



György Vaszil, Claudio Zandron,
Gexiang Zhang (Eds.)

ICMC 2021

International Conference on Membrane Computing

Chengdu, China and Debrecen, Hungary, 2021

Proceedings



ICMC 2021. International Conference on Membrane Computing, Chengdu, China and Debrecen, Hungary, 2021, Proceedings [PDF]

Published by the Faculty of Informatics of the University of Debrecen, Kassai út 26, 4028 Debrecen

Edited by © György Vaszil, Claudio Zandron, Gexiang Zhang, 2021

Copyright © Authors of the contributions, 2021

Published in August 2021

ISBN 978-963-490-329-1

Preface

Due to the continuing of the pandemic caused by the Corona virus, the two main conferences organized annually by the Membrane Computing community through IMCS, the *10th Asian Conference on Membrane Computing (ACMC 2021)* and the *22nd (European) Conference on Membrane Computing (CMC 2021)* are united in one joint conference also in 2021.

The *International Conference on Membrane Computing 2021 (ICMC 2021)* is organized by the Chengdu University of Information Technology, Chengdu, China and by the Faculty of Informatics of the University of Debrecen, Hungary as hybrid event with face-to-face participation in Chengdu for attendees from within China, and as an electronic conference with the possibility of online participation for attendees from the rest of the world outside of China.

This volume contains the abstracts of the invited presentations and the papers submitted to ICMC 2021 and accepted (as regular or short papers) on the basis of three referee reports, along with some abstracts of talks intended to be presented at the conference. We thank the members of the program committee

Henry Adorna, Quezon City, Philippines
 Artiom Alhazov, Chiinu, Rep. of Moldova
 Bogdan Aman, Iai, Romania
 Péter Battyányi, Debrecen, Hungary
 Francis George C. Cabarle, Quezon City, Philippines
 Lucie Cencialová, Opava, Czech Republic
 Erzsébet Csuhaj-Varjú, Budapest, Hungary
 Rudolf Freund, TU Wien, Austria
 Giuditta Franco, Verona, Italy
 Xiaoju Dong, Shanghai, China
 Zsolt Gazdag, Szeged, Hungary
 Marian Gheorghe, Bradford, U.K.
 Ping Guo, Chongqi, China
 Juanjuan He, Wuhan, China
 Thomas Hinze, Jena, Germany
 Florentin Ipate, Bucharest, Romania
 Shankara N. Krishna, Bombay, India
 Tseren-Onolt Ishdorj, Ulaanbaatar, Mongolia
 Savas Konur, Bradford, UK
 Alberto Loporati, Milano, Italy
 Jia Li, Chongqing, China
 Xiangrong Liu, Xiamen, China
 Xiyu Liu, Jinan, China
 Ravie Chandren Muniyandi, Bangi, Malaysia
 Ferrante Neri, Nottingham, UK
 Radu Nicolescu, Auckland, New Zealand
 Taishin Nishida, Toyama, Japan
 David Orellana-Martín, Sevilla, Spain
 Yunyun Niu, Beijing, China
 Linqiang Pan, Wuhan, China,
 Andrei Păun, Bucharest, Romania
 Gheorghe Păun, Bucharest, Romania

Hong Peng, Chengdu, China
Mario Pérez-Jiménez, Sevilla, Spain
Antonio E. Porreca, Marseille, France
Agustín Riscos-Núñez, Sevilla, Spain
Haina Rong, Chengdu, China
José M. Sempere, Valencia, Spain
Bosheng Song, Changsha, China
Tao Song, Qingdao, China
Petr Sosík, Opava, Czech Republic
K.G. Subramanian, Chennai, India
D.G. Thomas, Chennai, India
György Vaszil, Debrecen, Hungary, co-chair
Sergey Verlan, Paris, France
Jun Wang, Chengdu, China
Tingfang Wu, Suzhou, China
Jianhua Xiao, Tianjing, China
Jie Xue, Jinan, China
Hsu-Chun Yen, Taiwan, R.O.C
Jianying Yuan, Chengdu, China
Claudio Zandron, Milano, Italy, co-chair
Xiangxiang Zeng, Xiamen University
Gexiang Zhang, Chengdu, China, co-chair
Xingyi Zhang, Anhui, China
Xue Zhang, Boston, USA
Xuncaizhang, Zhengzhou, China
Ming Zhu, Chengdu, China

for participating in the evaluation process.

The European Branch of ICMC is held every year from the year of 2000 in different European Countries, while the Asian Branch of ICMC is held since 2012 in the Asian region. Due to the pandemic situation, the European and the Asian conferences were organized as one joint, online-only conference in 2020.

ICMC 2021 is intended to be a special event honoring the 10th anniversary of the Asian conference series ACMC, while also bringing together researchers from all over the world working in Membrane Computing and related areas by providing the possibility of online participation enhancing communication and cooperation in the virtual place of an electronic conference.

György Vaszil, Claudio Zandron, Gexiang Zhang
(Program Committee Co-Chairs)

Chengdu and Debrecen, 2021

Contents

Abstracts of Invited Talks	9
Regular papers	19
Artiom Alhazov, Rudolf Freund, Sergiu Ivanov and Sergey Verlan: Variants of Simple P Systems with One Catalyst Being Computationally Complete . . .	21
Artiom Alhazov, Rudolf Freund, Sergiu Ivanov and Marion Oswald: Variants of Simple Purely Catalytic P Systems with Two Catalysts	39
Artiom Alhazov, Alberto Leporati, Luca Manzoni, Giancarlo Mauri and Claudio Zandron: Evaluating Space Measures in P Systems	54
Bogdan Aman: Solutions to SAT, Threshold-SAT and Unique-SAT in Synchronized P Systems	69
Korsie Ballesteros, Dionne Peter Cailipan, Ren Tristan de La Cruz, Francis George Cabarle and Henry Adorna: Matrix Representation and Simulations of Numerical Spiking Neural P Systems	84
Somnath Bera, Rodica Ceterchi, Sastha Sriram and K G Subramanian: Array P Systems and Pure 2D Context-free Grammars with Independent Mode of Rewriting	102
Huijian Chen and Xiyu Liu: Double Fusion of multiple graphs for Multi-view Clustering based on tissue-like P System	117
Gabriel Ciobanu and Eneia Nicolae Todoran: Spiking Neural P Systems and Their Semantics in Haskell	128
Jianping Dong, Gexiang Zhang, Biao Luo, Qiang Yang, Dequan Guo, Haina Rong and Ming Zhu: A Modified Optimization Spiking Neural P System for Knapsack Problems	145
Jianping Dong, Xingqiao Deng, Shisong Wang, Biao Luo, Huiling Feng and Gexiang Zhang: Multi-parameter Optimization of Reducer Lubrication Based on Optimization Spiking Neural P Systems	167
Yingying Duan, Haina Rong, Gexiang Zhang, Dunwu Qi, Luis Valencia-Cabrera and Mario J. Pérez-Jiménez: A Review of Computing Models for Giant Panda Ecosystems	191
Yingying Duan, Haina Rong, Gexiang Zhang, Dunwu Qi, Luis Valencia-Cabrera and Mario J. Pérez-Jiménez: Minimum Viable Population Estimations for Giant Pandas using Membrane Computing Models	243
Annyssia Glynis Dupaya, Anica Clarice Galano, Francis George Cabarle, Ren Tristan de La Cruz, Ivan Cedric Macababayao, Korsie Ballesteros and Prometheus Peter Lazo: A Web-based Visual Simulator for Spiking Neural P Systems . .	264
Alec Henderson, Radu Nicolescu, Michael J. Dinneen, TN Chan, Hendrik Happe and Thomas Hinze: Parallel Computing With Water	296
Javier Hernández-Tello, Miguel Á. Martínez-Del-Amor, David Orellana-Martín and Francis George Cabarle: Sparse matrix representation of Spiking Neural P systems on GPUs	316
James Immanuel S, Jayasankar S, Gnanaraj Thomas D, Robinson T and Atulya K Nagar: Comparison Results On Parallel Contextual Array P Systems and Parallel Contextual Array Insertion Deletion P Systems	323
Florentin Ipate, Marian Gheorghe and Ionuț Mihai Niculescu: Experimenting with Model Learning for Spiking Neural P Systems	343

Prometheus Peter Lazo, Ren Tristan de La Cruz, Ivan Cedric Macababayao, Francis George Cabarle and Henry Adorna: Extended Rules and Heterogeneous Derivation Modes in Spiking Neural P Systems with Stochastic Application of Rules	352
David Orellana-Martín, Luis Valencia-Cabrera and Mario J. Pérez-Jiménez: Minimal cell-like membrane systems solving PSPACE-complete problems	379
Hongping Song, Yourui Huang, Qi Song, Tao Han and Shanyong Xu: Feature Selection Algorithm Based on P systems	390
Arian Allenson Valdez, Filbert Wee, Francis George Cabarle and Miguel Á. Martínez-Del-Amor: GPU simulations of spiking neural P systems on modern web browsers	400
Qi Wu and Yuzhen Zhao: Targetindicating Spiking Neural P Systems with Polarizations and Numeric Value	413
Xiaojian Yang, Qian Liu and Xiyu Liu: An Improved Deep Echo State Network Inspired by Tissue-Like P System Forecasting for Non-stationary Time Series	432
Wenping Yu, Jieping Wu, Yubo Wu and Yufeng Chen: Fuzzy Tissue-like P Systems with Promoters and Their Application in Power Coordinated Control of Microgrid	443
Luping Zhang and Fei Xu: Time-free RSSNP systems with synaptic co-transmission	457
Regular Papers: Nature-Inspired Computing	473
Dequan Guo, Shenggui Ling, Tianxiang Li and Haoyuan Ma: Face Illumination Normalization based on Generative Adversarial Network	475
Mingshan Liu and Yunyun Niu: Attention-based deep residual network for ghost imaging	497
Tianqi Liu, Hua Yang, Jing Yu and Kang Zhou: A new global harmony search algorithm for solving numerical optimization	510
Jin Zhou, Kang Zhou, Gexiang Zhang and Jiuju Yin: Research on Deep Learning Model for Solving the Evaluation of Wine Quality	529
Short Papers	565
Yourui Huang, Qi Song, Hongping Song, Shanyong Xu and Tao Han: Application Of The Population Dynamics Membrane System In COVID-19 Propagation Model	567
James Immanuel S, Jayasankar S, Gnanaraj Thomas D, Gayathri Lakshmi M and Meenakshi Paramasivan: Some Results on Parallel Contextual Hexagonal Array Insertion Deletion P System	573
Jayakrishna V and Lisa Mathew: nc-eNCE Graph Grammars and Graph Rewriting P Systems	582
Kalyani T, Raman T.T., Thomas D.G., Bhuvanewari K and Ravichandran P: Variants of Tetrahedral Tile Pasting P System Model for 3D Patterns	590
Yuping Liu, Yuzhen Zhao and Xiyu Liu: Weighted spiking neural P systems with polarizations and anti-spikes	598
Sweety F, Annadurai S, Kalyani T, Thomas D G and Bhuvanewari K: Domino Recognizable Three Dimensional Picture Languages and P System	606
Short Papers: Nature-Inspired Computing	615
Jing Yu, Shuo Liu, Tianqi Liu, Kang Zhou, Yan Zhang, Wenbin Ma and Pinpin Wu: Prediction of Grain Temperature Based on Intelligent Algorithm Optimized BP Neural Network	617

Abstracts	625
James Cooper and Radu Nicolescu: Revisiting Sorting and Selection Applied to Median Filtering	627
Qianqian Ren and Xiyu Liu: Spiking neural P systems with energy and threshold .	630
Xiaoxiao Song: Two New Variants of Spiking Neural P Systems	632
Xiang Tian and Xiyu Liu: Enzymatic Spiking Neural P Systems	635
Liping Wang and Xiyu Liu: Evolution-Communication Spiking Neural P Systems with Energy on Neurons	636
Qiang Yang and Gexiang Zhang: A Review on the Research of Rough Set Mem- brane Computing	638
Xiu Yin, Xiyu Liu, Jie Xue and Yuzhen Zhao: Numerical P systems of directed graph structure with threshold and plasticity	639
Qiang Yang and Gexiang Zhang: A Review on the Research of Rough Set Mem- brane Computing	641
Author Index	643

Abstracts of Invited Talks



Bogdan Aman graduated Alexandru Ioan Cuza University of Iasi (Faculty of Mathematics) in 2007 and completed his Ph.D. thesis in 2009 under the supervision of Prof. Gabriel Ciobanu at the Romanian Academy (Iasi branch). He received a public recognition for his research with the 2013 Grigore Moisil Award of the Romanian Academy of Sciences and 2019 International Membrane Computer Society (IMCS) Prize for the Theoretical Result of the Year. His main research fields are membrane computing, natural computing, process algebra, type systems, and other theoretical aspects of computer science.

Talk: Evolution strategies in membrane computing

Abstract: Membrane systems consist of a set of rewriting rules over (multi)sets of objects, together with an initial (multi)set of objects; membrane systems are used to describe the dynamics of systems which involve parallel access to resources. The evolution of a membrane system consists of applying rules over available resources (objects) using various strategies. We give a survey of the computation power and efficiency of membrane systems using various evolution strategies.



Xiyu Liu, Full professor, business school, Shandong normal university, China, received Ph.D. degrees in Fundamental Mathematics from Shandong university, in 1990. He works in Shandong Normal University and currently serves as the Dean of the Academy of Management Science. He was selected as the “Taishan Scholar” Distinguished Professor, the Young and Middle-aged Experts with Outstanding Contribution in Shandong Province, and the Outstanding Postgraduate Tutor in Shandong Province. At present, he has presided over more than 10 national and provincial projects, including 3 projects of National Natural Science Foundation of China, and published more than 150 academic papers. He is mainly engaged in the research of

biological computing, computing intelligent and nonlinear analysis.

Talk: Complex topological membrane systems and application in medical image processing

Abstract: Membrane computation (also called a membrane system or P system) is a novel paradigm that has gained popularity in the past few years because it offers promising features such as the data encapsulation, simple information representation, and especially, parallelism. However, for computation purposes, the traditional membrane systems simplify real membrane structures and do not use the complex membrane structures to solve problems with complicated structures in the field of ecology, optimization and so on. Therefore, making use of the complex structures of membranes may improve the performance of membrane systems in real applications.

Based on the above consideration, the main motivation of this work is to use membrane systems to develop a learning framework for medical image processing. We propose a membrane system with hybrid structures, where the hybrid structures combine the advantages of classical membrane structures, such as tissue-like, cell-like and neural-like P systems.



Hong Peng, Full professor, School of Mathematics and Computer Science, Xihua University, China, received the Ph.D. degree in signal and information processing from the University of Electronic Science and Technology of China, Chengdu, China, in 2010. He has been a Professor with the School of Computer and Software Engineering, since 2005. He was a Visiting Scholar with the Research Group of Natural Computing, University of Seville, Spain, from 2011 to 2012. He has published over 100 scientific articles in international journals. His research interests include membrane computing, machine learning,

image processing, and computer vision.

Talk: Computer: Some variants of spiking neural P systems and application in image fusion

Abstract: Spiking neural P systems are a class of neural-like membrane computing models, inspired from the spiking mechanism of biological neurons. In the past years, some biological mechanisms have been introduced to propose many variants of spiking neural P systems. We first introduce several models recently proposed by our group, and then give the results of their computational completeness.

Image fusion is a basic task of image processing, especially multi-modal image fusion, including multi-focus images, multi-modality medical images, and optical and infrared images. We introduce application of two variants in multi-modal image fusion, including dynamic threshold neural P systems and coupled neural P systems.



Mario J. Perez Jimenez, Full Professor at the Department of Computer Science and Artificial Intelligence at Universidad de Sevilla, Spain, since 2009, and currently Emeritus Professor. From 2005 to 2007 he was a Guest Professor of the Huazhong University of Science and Technology, Wuhan, China. He is a numerary member of the Academia Europaea (The Academy of Europe) in the Section of Informatics. His main research interests include theory of computation, computational complexity theory, natural computing (DNA computing and membrane computing), bioinformatics and computational modelling for complex systems. He has published 19 books in computer science and mathematics, and over 300 scientific papers in international journals (collaborating with researchers worldwide) and he is a member of the Editorial Board of six ISI journals. He has been the first scientist awarded with “Important Contributions to Membrane Computing” under the auspices of the European Molecular Computing Consortium, Edinburgh, 2008. In 2014, he received the University of Sevilla’s FAMA award for his outstanding research career. He has been the main researcher in various European, Asian, Spanish and Andalusian research grants. From 2003 he is an expert reviewer of the Prospective and Evaluation National Agency of Spain. From May 2006 he is an European Science Foundation peer reviewer, from July 2008 he is an expert reviewer from the Romanian National University Research Council and from October 2015 he is an international expert from the Russian Science Foundation, invited by the Russian International Affairs Council.

Talk: Membrane systems breaking cryptosystems

Abstract: Cryptography is a scientific discipline that concerns information security in presence of possible intruders, as well as authentication and identification, providing privacy and integrity. The first ever published public key cryptosystem, named RSA, was developed by R. Rivest, A. Shamir and L. Adleman in 1978. The security that resides in this system is based on the apparent computational hardness of the integer factorization problem. More precisely, the semiprime factorization problem (given a natural number product of two prime numbers, find its decomposition) is used.

In this talk, the cited problem, among others, is studied from the Membrane Computing perspective, and a new kind of membrane systems with the ability to compute partial functions among natural numbers, are presented. This provides a new approach to attack RSA cryptosystems.



Bosheng Song, Professor, Department of Computer Science, School of Information Science and Engineering, Hunan University, China. He received the Ph.D. degree in control science and engineering from Huazhong University of Science and Technology, Wuhan, China, in 2015. He spent eighteen months working in the Research Group on Natural Computing, University of Seville, Seville, Spain, from November, 2013 to May, 2015. He was worked as a post-doctoral researcher with the School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China, from March, 2016 to February, 2019. He is currently an Associate Professor with the College of Information Science and Engineering, Hunan University, Changsha, China. His current research interests include membrane computing and bioinformatics.

Talk: Some variants of P systems and their computational properties

Abstract: Membrane computing is an unconventional computing area that aims to abstract computing ideas (e.g., computing models, data structures, data operations) from the structure and functioning of living cells, as well as from more complex biological entities, like tissues, organs and populations of cells. The computational models that are part of this paradigm are generically called P systems, which are distributed and parallel computing devices. In this talk, inspired by different biological facts, some variants of P systems are introduced and the obtained results of these P systems are presented; besides, some open problems are given.



Petr Sosik, Professor, Department of Computer Science, School of Philosophy and Science, Silesian University, Opava. He is a head of research unit and a guarantee of several study programs. His research covers the area of bio-inspired computing (DNA computing, membrane computing, morphogenetic systems) and lately also machine learning applications. He has (co-)authored over 100 book chapters, journal and refereed conference publications. He supervises the work of several PhD students. Several times he was awarded the Best Paper Award or Research Result of the Year.

Talk: Simulations of Bacteria with Morphogenetic Systems

Joint work with Martin Pavlíček

Abstract: Morphogenetic (M) systems is a computational model inspired by morphogenesis of living cells. Mathematically, it is based partly on the concept of P systems with proteins on membranes providing abstract metabolic processes, and partly on the algorithmic self-assembly of tiles. An M system, however, generalizes both concepts into a unique framework. It allows to self-assemble 1D or 2D primitives of arbitrary pre-defined shapes into 2D or 3D forms, while the process is controlled by flow of atomic objects due to P-system-like rules. It was shown that M systems are computationally universal, error-prone, with strong self-healing properties, and able to solve NP-hard problems in a polynomial time.

Here we show that M systems are also able to simulate key processes in bacteria on a high level of granularity, while following important qualitative and quantitative macro-properties of the simulated cells. Initial experiments simulating growth of cytoskeleton inside cells were extended to cell fission processes under changing environmental conditions and to resistance of cells (e.g., E.Coli) to antibiotic agents. We show that, in spite of relative simplicity of the designed models of both prokaryotic and eukaryotic cells, their results faithfully correspond to published biological observations.



Sergey Verlan, Full professor, Department of Computer Science, School of Science and Technology, University of Paris, France. He received his PhD in Computer Science at the University of Metz, France (2004). He obtained a habilitation in Computer Science (2010). Currently he is an associated professor (maître de conférences) at the University of Paris Est Créteil (France). His research interests belong to the area of theoretical computer science and natural computing. He has expertise in the area of formal language theory, DNA computing, membrane computing, modeling of biological systems and hardware design. He is particularly interested in the universality problem and provided several universal constructions which are the smallest known for the corresponding classes. He also introduced the formal framework for P systems that allows to explain, compare and extend different variants of P systems. He has more than 100 articles published in scientific journals and international conference proceedings. He edited 6 special journal issues and contributed to 12 book chapters.

Talk: A formal look at spiking neural P systems

Abstract: In this talk we will consider the model of spiking neural P systems (SNP) from the formal perspective. Based on the formal framework for P systems we decompose the model and its semantics in distinct bricks and then show how different existing variants of SNP can be constructed from them. This allows to compare different variants of SNP systems, to extend them and to provide bisimulations.

We will also discuss the vector notation for the description of SNP systems and the relations (bisimulations) to other variants of P systems and related models (like Petri nets).



Jin Xu, Full professor, Department of Computer Science and Technology, School of Electronics Engineering and Computer Science, Peking University, China, received Ph.D. degrees both in science and engineering. He has published more than 300 academic papers, including more than 200 SCI indexed articles as the first author or correspondent; 5 monographs and 1 translation manuscripts. He has presided over more than 10 national projects consists of the National Natural Science Foundation Key Project, International Cooperation Project, Instrument R&D Project, 973 Project, 863 Project, National Key R&D Program, National Defense Pre-research Project and so on. In

2013, he won the second prize of National Natural Science as the first complete person. In addition, he won 2 first prizes in Natural Science of the Ministry of Education both as the first person to be completed. In academic terms, he was the first, second, fourth, fifth, seventh, eighth International Biomedical Conference chairman, editor in several journals, expert of Military Commission of Science and Technology Committee, the vice chairman of China Circuit and System Society, the chairman of Biometrics and Bioprocessing Professional Committee of China, and the member of the Ministry of Education Cyberspace Security Advisory Committee. His main research areas are graph theory and combinatorial optimization, novel computing models, biological computing, information security, etc.

Talk: Computer: Past-Present-Future

Abstract: This report reviews a brief history of computing tools and gives a new definition of computer from the perspective of decomposability; points out the reasons why electronic computers cannot effectively deal with the combinatorial explosion problems, which leads to the current computer science in the "Warring States Era"; and gives the basic principles and research progress of DNA computing, in particular, inspired by DNA computing, the reporter discovered a kind of fully parallel mathematical computing model from the bottom—the Probe Machine, and gave two types of probe computers based on biological materials and electronic materials. Finally, the report points out the possible development trend of computer in the future is: the mathematical computing model of the computer is a series/parallel structure type, and its realization materials are mainly protein materials.

Regular Papers

Variants of Simple P Systems with One Catalyst Being Computationally Complete

Artiom Alhazov¹, Rudolf Freund², Sergiu Ivanov³, and Sergey Verlan⁴

¹ Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chişinău, MD-2028, Moldova
artiom@math.md

² Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

³ IBISC, Univ. Évry, Paris-Saclay University
23, boulevard de France 91034 Évry, France
sergiu.ivanov@ibisc.univ-evry.fr

⁴ Univ. Paris Est Creteil, LACL, 94010, Creteil, France
verlan@u-pec.fr

Abstract. Catalytic P systems are among the first variants of membrane systems ever considered in this area. This variant of systems also features some prominent computational complexity questions, and in particular the problem of using only one catalyst: is one catalyst enough to allow for generating all recursively enumerable sets of multisets? Several additional ingredients have been shown to be sufficient for obtaining even computational completeness with only one catalyst.

Last year we could show that the derivation mode *max_{objects}*, where we only take those multisets of rules which affect the maximal number of objects in the underlying configuration one catalyst is sufficient for obtaining computational completeness without any other ingredients. In this paper we follow this way of research and show that one catalyst is also sufficient for obtaining computational completeness when using specific variants of derivation modes based on non-extendable multisets of rules: we only take those non-extendable multisets whose application yields the maximal number of generated objects or else those non-extendable multisets whose application yields the maximal difference in the number of objects between the newly generated configuration and the current configuration.

1 Introduction

Two decades ago, membrane systems were introduced in [30] as a multiset-rewriting model of computing inspired by the structure and the functioning of the living cell. The development of this fascinating area of biologically motivated computing models is documented in two textbooks, see [31] and [32]. For actual

information see the P systems webpage [34] and the issues of the Bulletin of the International Membrane Computing Society and of the Journal of Membrane Computing.

One basic feature of P systems already presented in [30] is the maximally parallel derivation mode, i.e., using non-extendable multisets of rules in every derivation step. The result of a computation can be extracted when the system halts, i.e., when no rule is applicable any more. Catalysts are special symbols which allow only one object to evolve in its context (in contrast to promoters) and in their basic variant never evolve themselves, i.e., a catalytic rule is of the form $ca \rightarrow cv$, where c is a catalyst, a is a single object and v is a multiset of objects. In contrast, non-catalytic rules in catalytic P systems are non-cooperative rules of the form $a \rightarrow v$.

From the beginning, the question how many catalysts are needed for obtaining computational completeness has been one of the most intriguing challenges regarding (catalytic) P systems. In [19] it has already been shown that two catalysts are enough for generating any recursively enumerable set of multisets, without any additional ingredients like a priority relation on the rules as used in the original definition. As already known from the beginning, without catalysts only regular (semi-linear) sets can be generated when using the standard maximal derivation mode and the standard halting mode, i.e., a result is extracted when the system halts with no rule being applicable any more. As shown, for example, in [22], using various additional ingredients, i.e., additional control mechanisms, one catalyst can be sufficient: in P systems with label selection, only rules from one set of a finite number of sets of rules in each computation step are used; in time-varying P systems, the available sets of rules change periodically with time. For many other variants of P systems using specific control mechanism for the application of rules the interested reader is referred to the list of references, for example, see [1–9, 12, 10, 11, 13, 17, 16, 18, 21–24, 27, 28].

On the other hand, for such catalytic P systems with only one catalyst and using the standard maximally parallel derivation mode and the standard halting mode, a lower bound has been established in [25]: P systems with one catalyst can simulate partially blind register machines, i.e., they can generate more than just semi-linear sets.

In [6], we returned to the idea of using a priority relation on the rules, but took only a very weak form of such a priority relation: we only required that overall in the system catalytic rules have weak priority over non-catalytic rules. This means that the catalyst c must not stay idle if the current configuration contains an object a with which it may cooperate in a rule $ca \rightarrow cv$; all remaining objects evolve in the maximally parallel way with non-cooperative rules. On the other hand, if the current configuration does not contain an object a with which the catalyst c may cooperate in a rule $ca \rightarrow cv$, c may stay idle and *all* objects evolve in the maximally parallel way with non-cooperative rules. Even without using more than this weak priority of catalytic rules over the non-catalytic (non-cooperative) rules, we could establish computational completeness for catalytic P systems with only one catalyst. Moreover, starting from a result established

in [6], an even stronger result using a similar construction as in [6] has been established in [9] where we show computational completeness for catalytic P systems with only one catalyst using the derivation mode *max_objects*, i.e., we only take those multisets of rules which affect the maximal number of objects in the underlying configuration.

In this paper we now continue the research started in [9] and investigate several variants of derivation modes based on non-extendable multisets of rules and taking only those for which the difference of objects between the underlying configuration and the configuration after the application of the multisets of rules is maximal. We also consider the variants where the number of symbols generated by the application of a multiset of rules is maximal.

Finally, for the variants with maximal number of objects we can also take these multisets of rules without requesting them to fulfill the condition of the multisets to be non-extendable.

2 Definitions

For an alphabet V , by V^* we denote the free monoid generated by V under the operation of concatenation, i.e., containing all possible strings over V . The *empty string* is denoted by λ . A *multiset* M with underlying set A is a pair (A, f) where $f : A \rightarrow \mathbb{N}$ is a mapping. If $M = (A, f)$ is a multiset then its *support* is defined as $\text{supp}(M) = \{x \in A \mid f(x) > 0\}$. A multiset is empty (respectively finite) if its support is the empty set (respectively a finite set). If $M = (A, f)$ is a finite multiset over A and $\text{supp}(M) = \{a_1, \dots, a_k\}$, then it can also be represented by the string $a_1^{f(a_1)} \dots a_k^{f(a_k)}$ over the alphabet $\{a_1, \dots, a_k\}$, and, moreover, all permutations of this string precisely identify the same multiset M . The set of all multisets over V is denoted by V° . The cardinality of a set or multiset M is denoted by $|M|$. For further notions and results in formal language theory we refer to textbooks like [15] and [33].

2.1 Register Machines

Register machines are well-known universal devices for computing (or generating or accepting) sets of vectors of natural numbers. The following definitions and propositions are given as in [9].

Definition 1. *A register machine is a construct*

$$M = (m, B, l_0, l_h, P)$$

where

- m is the number of registers,
- P is the set of instructions bijectively labeled by elements of B ,
- $l_0 \in B$ is the initial label, and
- $l_h \in B$ is the final label.

The instructions of M can be of the following forms:

- $p : (ADD(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.
Increase the value of register r by one, and non-deterministically jump to instruction q or s .
- $p : (SUB(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.
If the value of register r is not zero then decrease the value of register r by one (decrement case) and jump to instruction q , otherwise jump to instruction s (zero-test case).
- $l_h : HALT$.
Stop the execution of the register machine.

A configuration of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed. M is called deterministic if the ADD -instructions all are of the form $p : (ADD(r), q)$.

In the *accepting* case, a computation starts with the input of an l -vector of natural numbers in its first l registers and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$ -instruction. Without loss of generality, we may assume all registers to be empty at the end of the computation.

In the *generating* case, a computation starts with all registers being empty and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$ -instruction and the output of a k -vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

In the *computing* case, a computation starts with the input of an l -vector of natural numbers in its first l registers and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$ -instruction and the output of a k -vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

For useful results on the computational power of register machines, we refer to [29]; for example, to prove our main theorem, we need the following formulation of results for register machines generating or accepting recursively enumerable sets of vectors of natural numbers with k components or computing partial recursive relations on vectors of natural numbers:

Proposition 1. *Deterministic register machines can accept any recursively enumerable set of vectors of natural numbers with l components using precisely $l + 2$ registers. Without loss of generality, we may assume that at the end of an accepting computation all registers are empty.*

Proposition 2. *Register machines can generate any recursively enumerable set of vectors of natural numbers with k components using precisely $k + 2$ registers.*

Without loss of generality, we may assume that at the end of a generating computation the first two registers are empty, and, moreover, on the output registers, i.e., the last k registers, no *SUB*-instruction is ever used.

Proposition 3. *Register machines can compute any partial recursive relation on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using precisely $l + 2 + k$ registers, where without loss of generality, we may assume that at the end of a successful computation the first $l + 2$ registers are empty, and, moreover, on the output registers, i.e., the last k registers, no *SUB*-instruction is ever used.*

In all cases it is essential that the output registers never need to be decremented.

Remark 1. For any register machine, without loss of generality we may assume that the first instruction is an *ADD*-instruction: in fact, given a register machine $M = (m, B, l_0, l_h, P)$ with having a *SUB*-instruction as its first instruction, we can immediately construct an equivalent register machine M' by starting with an increment immediately followed by a decrement of the first register:

$$\begin{aligned} M' &= (m, B', l'_0, l_h, P'), \\ B' &= B \cup \{l'_0, l''_0\}, \\ P' &= P \cup \{l'_0 : (ADD(1), l''_0, l''_0), l''_0 : (SUB(1), l_0, l_0)\}. \end{aligned}$$

2.2 Simple Catalytic P Systems

Taking into account the well-known flattening process, e.g., see [20], in this paper we only consider simple catalytic P systems, i.e., with the simplest membrane structure of only one membrane, and with only one catalyst:

Definition 2. *A simple catalytic P system with only one catalyst is a construct*

$$\Pi = (V, \{c\}, T, w, R)$$

where

- V is the alphabet of objects;
- $c \in V$ is the single catalyst;
- $T \subseteq (V \setminus \{c\})$;
- $w \in V^\circ$ is the multiset of objects initially present in the membrane region;
- R is a finite set of evolution rules over V ; these evolution rules are of the forms $ca \rightarrow cv$ or $a \rightarrow v$, where c is a catalyst, a is an object from $V \setminus \{c\}$, and v is a multiset over $V \setminus \{c\}$.

The multiset in the single membrane region of Π constitutes a *configuration* of the P system; the *initial configuration* is given by the initial multiset w . A transition between configurations is governed by the application of the evolution rules, which is done in a given derivation mode. The application of a rule $u \rightarrow v$ to a multiset M results in subtracting from M the multiset identified by u , and then in adding the multiset identified by v .

2.3 Variants of Derivation Modes

The definitions and the corresponding notions used in this subsection follow the definitions and notions elaborated in [26] and extend them for the purposes of this paper.

Given a P system Π , the set of multisets of rules applicable to a configuration C is denoted by $\text{Appl}(\Pi, C)$; this set also equals the set $\text{Appl}(\Pi, C, \text{asyn})$ of multisets of rules applicable in the *asynchronous derivation mode* (abbreviated *asyn*).

Given a multiset R of rules in $\text{Appl}(\Pi, C)$, we write $C \xrightarrow{R} C'$ if C' is the result of applying R to C . The number of symbols affected by applying R to C is denoted by $\text{Aff}(C, R)$. The number of symbols generated in C' by the right-hand sides of the rules applied to C with the multiset of rules R is denoted by $\text{Gen}(C, R)$. The difference between the number of objects in C' and C is denoted by $\Delta\text{obj}(C, R)$. In all cases, the catalysts are taken into account, too.

The set $\text{Appl}(\Pi, C, \text{sequ})$ denotes the set of multisets of rules applicable in the *sequential derivation mode* (abbreviated *sequ*), where in each derivation step exactly one rule is applied.

The standard parallel derivation mode used in P systems is the *maximally parallel derivation mode* (*max* for short). In the maximally parallel derivation mode, in any computation step of Π we choose a multiset of rules from \mathcal{R} in such a way that no further rule can be added to it so that the obtained multiset would still be applicable to the existing objects in the configuration, i.e., in simple P systems we only take applicable multisets of rules which cannot be extended by further rules and are to be applied to the objects in the single membrane region:

$$\text{Appl}(\Pi, C, \text{max}) = \{R \in \text{Appl}(\Pi, C) \mid \text{there is no } R' \in \text{Appl}(\Pi, C) \\ \text{such that } R' \supset R\}.$$

As already introduced for multisets of rules in [14], we now consider the variant where the maximal number of rules is chosen. In the derivation mode $\text{max}_{\text{rules}}\text{max}$ only a maximal multiset of rules is allowed to be applied. But it can also be seen as the variant of the basic mode *max* where we just take a multiset of applicable rules with the maximal number of rules in it, hence, we will also call it the $\text{max}_{\text{rules}}$ derivation mode. Formally we have:

$$\text{Appl}(\Pi, C, \text{max}_{\text{rules}}) = \{R \in \text{Appl}(\Pi, C, \text{asyn}) \mid \\ \text{there is no } R' \in \text{Appl}(\Pi, C, \text{asyn}) \\ \text{such that } |R'| > |R|\}.$$

We also consider the derivation mode $\text{max}_{\text{objects}}\text{max}$ where from the multisets of rules in $\text{Appl}(\Pi, C, \text{max})$ only those are taken which affect the maximal number of objects. As with affecting the maximal number of objects, such multisets of rules are non-extendable anyway, we will also use the notation $\text{max}_{\text{objects}}$. Formally we may write:

$$\begin{aligned} Appl(\Pi, C, max_{objects}max) = \{ & R \in Appl(\Pi, C, max) \mid \\ & \text{there is no } R' \in Appl(\Pi, C, max) \\ & \text{such that } \text{Aff}(C, R) < \text{Aff}(C, R') \} \end{aligned}$$

and

$$\begin{aligned} Appl(\Pi, C, max_{objects}) = \{ & R \in Appl(\Pi, C, async) \mid \\ & \text{there is no } R' \in Appl(\Pi, C, async) \\ & \text{such that } \text{Aff}(C, R) < \text{Aff}(C, R') \}. \end{aligned}$$

As already mentioned, both definitions yield the same multiset of rules.

In addition to these well-known derivation modes, in this paper we also consider several new variants of derivation modes.

Remark 2. The inherent possibility of mimicking the weak priority of catalytic rules over non-catalytic rules taken from the set of applicable non-extendable multisets of rules in simple catalytic P systems using the derivation mode $max_{objects}$ allowed us to show that simple catalytic P systems with only one catalyst are computationally complete when using the derivation mode $max_{objects}$, see [9] .

We now define new derivation modes starting from the non-extendable multisets of rules applicable to the current configuration C as for the derivation mode max , which instead of looking at the number of affected symbols take into account the number of generated symbols and the difference of symbols between the derived configuration and the current configuration, respectively.

$max_{GENobjects}max$ a non-extendable multiset of rules R applicable to the current configuration C is only taken if the number of objects generated by the application of the rules in R to the configuration C is maximal with respect to the number of objects generated by the application of the rules in any other non-extendable multiset of rules R' to the configuration C :

$$\begin{aligned} Appl(\Pi, C, max_{GENobjects}max) = \{ & R \in Appl(\Pi, C, max) \mid \\ & \text{there is no } R' \in Appl(\Pi, C, max) \\ & \text{such that } \text{Gen}(C, R) < \text{Gen}(C, R') \}. \end{aligned}$$

$max_{\Delta objects}max$ a non-extendable multiset of rules R applicable to the current configuration C is only taken if the difference $\Delta C = |C'| - |C|$ between the number of objects in the configuration C' obtained by the application of R and the number of objects in the underlying configuration C is maximal with respect to the differences in the number of objects obtained by applying any other non-extendable multisets of rules:

$$\begin{aligned} Appl(\Pi, C, max_{\Delta objects}max) = \{ & R \in Appl(\Pi, C, max) \mid \\ & \text{there is no } R' \in Appl(\Pi, C, max) \text{ such that} \\ & \Delta obj(C, R) < \Delta obj(C, R') \}. \end{aligned}$$

We illustrate the difference between these new derivation modes in the following example:

Example 1. To illustrate the derivation modes $max_{GENobjects}max$ as well as $max_{\Delta objects}max$, consider a simple P system with the initial configuration caa and the following rules:

1. $a \rightarrow b$
2. $ca \rightarrow cd$

In case of the derivation mode $max_{GENobjects}max$, only the multiset of rules $\{ca \rightarrow cd, a \rightarrow b\}$ can be applied, as $Gen(caa, \{ca \rightarrow cd\}) = 2$ and $Gen(cab, \{a \rightarrow b\}) = 1$ and therefore $Gen(caa, \{ca \rightarrow cd, a \rightarrow b\}) = 3$, whereas $Gen(caa, \{a \rightarrow b, a \rightarrow b\}) = 2$. Hence, the only possible derivation with the derivation mode $max_{GENobjects}max$ is $caa \xrightarrow{\{ca \rightarrow cd, a \rightarrow b\}} cdb$. In this special case,

$$Appl(\Pi, caa, max_{GENobjects}max) = Appl(\Pi, caa, max_{objects}).$$

On the other hand, with the derivation mode $max_{\Delta objects}max$ both rules yield the same difference of 0, i.e.,

$$\Delta obj(caa, \{ca \rightarrow cd\}) = \Delta obj(caa, \{a \rightarrow b\}) = 0,$$

which yields all two non-extendable multisets of rules $\{ca \rightarrow cd, a \rightarrow b\}$ and $\{a \rightarrow b, a \rightarrow b\}$ to be applicable to the underlying configuration caa , i.e.,

$$Appl(\Pi, caa, max_{\Delta objects}max) = Appl(\Pi, caa, max).$$

Now let us take a slightly different set of rules:

1. $a \rightarrow bb$
2. $ca \rightarrow cd$

Observing that $Gen(caa, \{a \rightarrow bb\}) = 2$ and $\Delta obj(caa, \{a \rightarrow bb\}) = 1$, we obtain the following sets of applicable multisets of rules in the two derivation modes considered here in this example:

$$\begin{aligned} Appl(\Pi, caa, max_{GENobjects}max) &= \{\{a \rightarrow bb, a \rightarrow bb\}, \{a \rightarrow bb, ca \rightarrow cd\}\}, \\ Appl(\Pi, caa, max_{\Delta objects}max) &= \{\{a \rightarrow bb, a \rightarrow bb\}\}. \end{aligned}$$

Finally, let us take the following set of rules:

1. $a \rightarrow \lambda$
2. $ca \rightarrow cd$

Observing that $Gen(caa, \{a \rightarrow \lambda\}) = 0$ and $\Delta obj(caa, \{a \rightarrow \lambda\}) = -1$, we obtain the following sets of applicable multisets of rules in the two derivation modes considered here in this example:

$$\begin{aligned} Appl(\Pi, caa, max_{GENobjects}max) &= \{\{a \rightarrow \lambda, ca \rightarrow cd\}\}, \\ Appl(\Pi, caa, max_{\Delta objects}max) &= \{\{a \rightarrow \lambda, ca \rightarrow cd\}\}. \end{aligned}$$

2.4 Computations in a P System

The P system continues with applying multisets of rules according to the derivation mode until there remain no applicable rules in the single region of Π , i.e., as usual, with all these variants of derivation modes as defined above, we consider *halting computations*.

We may generate or accept or even compute functions or relations. The inputs/outputs may be multisets or strings, defined in the well-known way. When the system *halts*, in case of computing with multisets we consider the number of objects from T contained in the membrane region at the moment when the system halts as the *result* of the underlying computation of Π .

We would like to emphasize that as results we only take the objects from the terminal alphabet T , especially the catalyst is not counted to the result of a computation. On the other hand, with all the proofs given in this paper, except for the single catalyst no other garbage remains in the membrane region at the end of a halting computation.

As already mentioned earlier, the following result was shown in [25], establishing a lower bound for the computational power of catalytic P systems with only one catalyst:

Proposition 4. *Catalytic P systems with only one catalyst working in the derivation mode \max have at least the computational power of partially blind register machines.*

Example 2. In [25] it was shown that the vector set

$$S = \{(n, m) \mid 0 \leq n, n \leq m \leq 2^n\}$$

(which is not semi-linear) can be generated by a P system working in the derivation mode \max with only one catalyst and 19 rules.

3 Computational Completeness for Simple P Systems Working in the Derivation Modes $\max_{GENobjects}$ and $\max_{\Delta objects}$

As already mentioned earlier, in [9] we have already shown that we can obtain computational completeness with simple P systems and only one catalyst when using the derivation mode $\max_{objects}$ instead of \max .

In this section we now study simple P systems with only one catalyst using the derivation modes $\max_{GENobjects}$ and $\max_{\Delta objects}$, respectively.

Theorem 1. *For any register machine with at least two decrementable registers we can construct a simple catalytic P system with only one catalyst, working in the derivation mode $\max_{\Delta objects}$ or in the derivation mode $\max_{GENobjects}$, which can simulate every step of the register machine in n steps where n is the number of decrementable registers.*

Proof. Given an arbitrary register machine $M = (m, B, l_0, l_h, P)$ we will construct a corresponding catalytic P system with one membrane and one catalyst $\Pi = (V, \{c\}, T, w, R)$ simulating M . Without loss of generality, we may assume that, depending on its use as an accepting or generating or computing device, the register machine M , as stated in Proposition 1, Proposition 2, and Proposition 3, fulfills the condition that on the output registers we never apply any *SUB*-instruction.

The following proof is given for the most general case of a register machine computing any partial recursive relation on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using precisely $l + 2 + k$ registers, where without loss of generality, we may assume that at the end of a successful computation the first $l + 2$ registers are empty, and, moreover, on the output registers, i.e., the last k registers, no *SUB*-instruction is ever used. In fact, the proof works for any number $n \geq 2$ of decrementable registers, no matter how many of them are the l input registers and the working registers, respectively.

The main idea behind our construction is that all the symbols except the catalyst c and the output symbols (representing the contents of the output registers) go through a cycle of length n where n is the number of decrementable registers of the simulated register machine. When the symbols are traversing the r -th section of the n sections, they “know” that they are to probably simulate a *SUB*-instruction on register r of the register machine M .

As in our construction the simulation of a *SUB*-instruction takes two steps, the second simulation step in the case of a *SUB*-instruction on register n is shifted to the first step of the next cycle. Yet in this case we have to guarantee that after a *SUB*-instruction on register n the next instruction to be simulated is not a *SUB*-instruction on register 1. Hence, we use a similar trick as already elaborated in Remark 1: we not only do not start with a *SUB*-instruction, but we also change the register machine program in such a way that after a *SUB*-instruction on register n two intermediate instructions are introduced, i.e., as in Remark 1, we use an *ADD*-instruction on register 1 immediately followed by a *SUB*-instruction on register 1, whose simulation will end at most in step n , as we have assumed $n \geq 2$.

The following construction is elaborated in such a way that it works both for the derivation mode $max_{\Delta objects} max$ and the derivation mode $max_{GEN objects} max$.

We now simulate the resulting register machine fulfilling these additional constraints $M = (m, B, l_0, l_h, P)$ by a corresponding simple P system with one catalyst $\Pi = (V, \{c\}, T, c(l_0, 1), R)$.

$$\begin{aligned}
 V = & \{a_r \mid n+1 \leq r \leq m\} \\
 & \cup \{(a_r, i) \mid 1 \leq r \leq n, 1 \leq i \leq n\} \\
 & \cup \{(p, i) \mid p \in B_{ADD}, 1 \leq i \leq n\} \\
 & \cup \{(p, i) \mid p \in B_{SUB(r)}, 1 \leq i \leq r+1\} \\
 & \cup \{(p, i)^-, (p, i)^0 \mid p \in B_{SUB(r)}, r+2 \leq i \leq n\} \\
 & \cup \{c, e, d\}.
 \end{aligned}$$

The construction includes the dummy symbol d which is erased by the rule $d \rightarrow \lambda$. The effect of applying these rules due to the requirement of the chosen multisets of rules to be non-extendable will be ignored in the following calculations for $\Delta obj(C, R)$ and $Gen(C, R)$.

The symbols a_r , $n+1 \leq r \leq m$, represent the output registers. For the decrementable registers, we use the symbols (a_r, i) , $1 \leq r \leq n, 1 \leq i \leq n$, which go through a loop of n steps. The main idea now is that the only case when such a symbol can be used to decrement register r is when $n = r$, i.e., in the r -th step of the simulation cycle.

$$(a_r, i) \rightarrow (a_r, i+1), 1 \leq r < n; \quad (a_r, n) \rightarrow (a_r, 1). \quad (1)$$

In the same way as the register symbols a_r , the program symbols (p, i) representing the label p from B undergo the same cycle of length n .

For simulating ADD -instructions we need the following rules:

Increment $p : (ADD(r), q, s)$:

$$c(p, i) \rightarrow c(p, i+1)d, \quad 1 \leq i < n. \quad (2)$$

The catalyst has to be used with the program symbol which otherwise would stay idle when the catalyst is used with a register symbol, and the difference of objects $\Delta obj(C, R')$ for this other non-extendable multiset of rules R' would be 0 whereas when using the program symbol for the catalyst, we obtain $\Delta obj(C, R) = 1$ because of the additional dummy symbol d .

In a similar way we can argue that in the case of the derivation mode $max_{GEN} objects max$ the number of generated symbols is maximal when using the catalyst together with the program symbol; in fact, if N is the total number of register symbols for decrementable registers in the underlying configuration C , then with applying the set of rules R described so far we get $Gen(C, R) = N + 3$ in contrast to $Gen(C, R') = N - 1 + 3 = N + 2$ where using the catalyst with the rule $c(a_r, r) \rightarrow ced$, as described below for the simulation of the SUB -Instruction, results in the multiset of rules R' .

If r is a decrementable register, we end the simulation using one of the following rules:

$$c(p, n) \rightarrow c(q, 1)(a_r, 1), \quad c(p, n) \rightarrow c(s, 1)(a_r, 1). \quad (3)$$

If r is an output register, we end the simulation using one of the following rules introducing output symbols not to be changed any more:

$$c(p, n) \rightarrow c(q, 1)a_r, \quad c(p, n) \rightarrow c(s, 1)a_r. \quad (4)$$

As in both cases, together with the program symbol a new register symbol is generated, we again have $\Delta obj(C, R) = 1$, thus guaranteeing that the catalyst must take (p, n) and cannot take (a_n, n) instead.

A similar argument again holds in the case of the derivation mode $max_{GENobjects}max$ as the number of generated symbols is only maximal when using the catalyst together with the program symbol; again we have $Gen(C, R) = N + 3$ with this multiset of rules R in contrast to $Gen(C, R') = N - 1 + 3 = N + 2$ when using the catalyst with the rule $c(a_r, r) \rightarrow ced$ results in the multiset of rules R' .

For simulating *SUB*-instructions we need the following rules:

Decrement and zero-test $p : (SUB(r), q, s)$:

$$c(p, i) \rightarrow c(p, i + 1)d, \quad 1 \leq i < r. \quad (5)$$

For $1 \leq i < r$, we again use the dummy symbol d to obtain $\Delta C = 1$ and thus also having one more symbol generated, to enforce the catalyst to take the program symbol.

$$(p, r) \rightarrow (p, r + 1), \quad c(a_r, r) \rightarrow ced. \quad (6)$$

In case that register r is empty, i.e., there is no object (a_r, r) , then the catalyst will stay idle as in this step there is no other object with which it could react. In case that register r is not empty, i.e., there is at least one object (a_r, r) , then one of these objects (a_r, r) must be used with the catalyst c as the rule $c(a_r, r) \rightarrow ced$ implies $\Delta obj(C, R) = 1$, whereas otherwise, if all register symbols are used with the rule $(a_r, r) \rightarrow (a_r, r + 1)$, then $\Delta obj(C, R) = 0$.

In the same way we argue that with using the rule $c(a_r, r) \rightarrow ced$ we get one symbol generated more than if we use the rule $(a_r, r) \rightarrow (a_r, r + 1)$ for that symbol (a_r, r) , i.e., $Gen(C, R) = N - 1 + 3 = N + 2$ in contrast to $Gen(C, R') = N$.

If $r < n$:

$$\begin{aligned} ce &\rightarrow cdddd, & (p, r + 1) &\rightarrow (p, r + 2)^-; \\ c(p, r + 1) &\rightarrow c(p, r + 2)^0dd. \end{aligned} \quad (7)$$

If in the first step of the simulation phase the catalyst did manage to decrement the register, it produced e . Thus, in the second simulation step, the catalyst has three choices:

1. the catalyst c correctly “erases” e using the rule $ce \rightarrow cdddd$, and to the program symbol $(p, r + 1)$ the rule $(p, r + 1) \rightarrow (p, r + 2)^-$ must be applied due to the fact that both derivation modes $max_{\Delta objects} max$ and $max_{GEN objects} max$ only allow for non-extendable multisets of rules; all register symbols evolve in the usual way; in total we get $\Delta obj(C, R) = 3$ and $Gen(C, R) = N + 6$;
2. the catalyst c takes the program symbol $(p, r + 1)$ using the rule $c(p, r + 1) \rightarrow c(p, r + 2)^0 dd$, and all register symbols evolve in the usual way; in total we get $\Delta obj(C, R) = 2$ and $Gen(C, R) = N + 4$;
3. the catalyst c takes a register object, the program symbol $(p, r + 1)$ evolves with the rule $(p, r + 1) \rightarrow (p, r + 2)^-$, and all other register objects evolve in the usual way; in total we get $\Delta obj(C, R) = 1$ and $Gen(C, R) = (N - 1 + 3) + 1 = N + 3$.

In total, only variant 1 fulfills the condition given by the derivation mode $max_{\Delta objects} max$ that $\Delta obj(C, R)$ is maximal, and therefore is the only possible continuation of the computation if register r is not empty.

A similar argument holds for the derivation mode $max_{Gen objects} max$ with respect to the number of generated symbols $\Delta obj(C, R)$.

On the other hand, if register r is empty, no object e is generated, and the catalyst c has only two choices:

1. the catalyst c takes the program symbol $(p, r + 1)$ using the rule $c(p, r + 1) \rightarrow c(p, r + 2)^0 dd$, and all register symbols evolve in the usual way; in total we get $\Delta obj(C, R) = 2$ and $Gen(C, R) = N + 4$;
2. the catalyst c takes a register object $(a_{r+1}, r + 1)$ thereby generating ed , the program symbol $(p, r + 1)$ evolves with the rule $(p, r + 1) \rightarrow (p, r + 2)^-$, and all other register objects evolve in the usual way; this variant leads to $\Delta obj(C, R) = 1$ and $Gen(C, R) = (N - 1 + 3) + 1 = N + 3$.

In total, variant 1 is the only possible continuation of the computation if register r is empty.

$$\begin{aligned}
 c(p, i)^- &\rightarrow c(p, i + 1)^- d, & r + 2 \leq i < n, & & c(p, n)^- &\rightarrow c(q, 1) d, \\
 c(p, i)^0 &\rightarrow c(p, i + 1)^0 d, & r + 2 \leq i < n, & & c(p, n)^0 &\rightarrow c(s, 1) d.
 \end{aligned} \tag{8}$$

Again the catalyst has to be used with the program symbol to get $\Delta obj(C, R) = 1$ and $Gen(C, R) = N + 3$, which otherwise would stay idle when the catalyst is used with a register symbol, and the multiset of rules applied in this way would only yield $\Delta obj(C, R) = 0$ and $Gen(C, R') = N - 1 + 3 = N + 2$.

If $r = n$:

$$\begin{aligned}
 ce &\rightarrow cdddd, \\
 (p, n + 1) &\rightarrow (q, 2), \quad c(p, n + 1) \rightarrow c(s, 2) dd.
 \end{aligned} \tag{9}$$

In this case, the second step of the simulation is already the first step of the next cycle, which means that in this case of $r = n$ the next instruction to be simulated is an *ADD*-instruction on register 1.

To complete the proof we have to implement the final *HALT*-instruction $l_h : HALT$. In an easy way, we can do this by introducing d instead of $(l_h, 1)$ or $(l_h, 2)$ as done for other labels. In this way, finally no program symbol is present any more in the configuration. As we have assumed all decrementable registers to be empty when the register machine halts, this means the constructed simple *P* system will also halt after having erased the dummy symbols d in the next step.

We finally observe that the proof construction given above is even deterministic if the underlying register machine to be simulated is deterministic. \square

As the number of decrementable registers in generating register machines needed for generating any recursively enumerable set of (vectors of) natural numbers is only two, from the theorem above we obtain the following result:

Corollary 1. *For any generating register machine with two decrementable registers we can construct a simple *P* system with only one catalyst and working in the derivation mode $\max_{\Delta\text{objects}}\max$ or in the derivation mode $\max_{GEN\text{objects}}\max$ which can simulate every step of the register machine in 2 steps, and therefore such catalytic *P* systems with only one catalyst and working in the derivation mode $\max_{\Delta\text{objects}}\max$ or in the derivation mode $\max_{GEN\text{objects}}\max$ can generate any recursively enumerable set of (vectors of) natural numbers.*

For accepting register machines, in addition to the two working registers, we have at least one input register and therefore immediately infer the following result:

Corollary 2. *For any recursively enumerable set of d -vectors of natural numbers given by a register machine with $d + 2$ decrementable registers we can construct an accepting simple *P* system with only one catalyst and working in the derivation mode $\max_{\Delta\text{objects}}\max$ or in the derivation mode $\max_{GEN\text{objects}}\max$ which can simulate every step of the register machine in $d + 2$ steps, and therefore such catalytic *P* systems with only one catalyst and working in the derivation mode $\max_{\Delta\text{objects}}\max$ or in the derivation mode $\max_{GEN\text{objects}}\max$ can accept any recursively enumerable set of (vectors of) natural numbers.*

In a similar way, assuming at least one input register to be present, we also infer a similar result for register machines computing partial recursive relations on natural numbers and therefore computational completeness in its widest sense:

Corollary 3. *For any partial recursive relation $f : \mathbb{N}^d \rightarrow \mathbb{N}^k$ on vectors of natural numbers (i.e., with d components as input and k components as output) given by a register machine with $d + 2$ decrementable registers we can construct a simple *P* system with only one catalyst and working in the derivation*

mode $max_{\Delta objects}max$ or in the derivation mode $max_{GENobjects}max$ which can simulate every step of the register machine in $d + 2$ steps, and therefore such catalytic P systems with only one catalyst and working in the derivation mode $max_{\Delta objects}max$ or in the derivation mode $max_{GENobjects}max$ can compute any partial recursive relation $f : \mathbb{N}^d \longrightarrow \mathbb{N}^k$ on vectors of natural numbers.

4 Conclusion

In this paper we have continued our research on revisiting a classic problem of computational complexity in membrane computing: can catalytic P systems with only one catalyst already generate all recursively enumerable sets of multisets? This problem has been standing tall for many years, and nobody has yet managed to give it a positive or a negative answer. Already in [6] and in [9] we could show that adding the ingredient of weak priority of catalytic rules over non-catalytic rules or only taking the derivation mode $max_{objects}$ we can obtain computational completeness with only one catalyst.

In this paper, we have added some similar results how to obtain computational completeness with only one catalyst when using one of the newly defined derivation modes $max_{\Delta objects}max$ and $max_{GENobjects}max$.

The results obtained in this paper can also be extended to P systems dealing with strings, following the definitions and notions used in [25], thus showing computational completeness for computing with strings.

Acknowledgements

The authors gratefully thank the referees for their useful comments. Artiom Alhazov acknowledges project 20.80009.5007.22 “Intelligent information systems for solving ill-structured problems, processing knowledge and big data” by the National Agency for Research and Development. Sergiu Ivanov is partially supported by the Paris region via the project DIM RFSI n°2018-03 “Modèles informatiques pour la reprogrammation cellulaire”.

References

1. Alhazov, A., Aman, B., Freund, R.: P systems with anti-matter. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing – 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8961, pp. 66–85. Springer (2014). https://doi.org/10.1007/978-3-319-14370-5_5
2. Alhazov, A., Aman, B., Freund, R., Păun, Gh.: Matter and anti-matter in membrane systems. In: Jürgensen, H., Karhumäki, J., Okhotin, A. (eds.) Descriptive Complexity of Formal Systems – 16th International Workshop, DCFS 2014, Turku, Finland, August 5–8, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8614, pp. 65–76. Springer (2014). https://doi.org/10.1007/978-3-319-09704-6_7

3. Alhazov, A., Freund, R.: P systems with toxic objects. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) *Membrane Computing – 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers*. Lecture Notes in Computer Science, vol. 8961, pp. 99–125. Springer (2014). https://doi.org/10.1007/978-3-319-14370-5_7
4. Alhazov, A., Freund, R., Ivanov, S.: Variants of energy-controlled P systems. In: *Proceedings of NIT 2016* (2016)
5. Alhazov, A., Freund, R., Ivanov, S.: Variants of P systems with activation and blocking of rules. *Nat. Comput.* **18**(3), 593–608 (2019). <https://doi.org/10.1007/s11047-019-09747-5>
6. Alhazov, A., Freund, R., Ivanov, S.: Catalytic P systems with weak priority of catalytic rules. In: Freund, R. (ed.) *Proceedings ICMC 2020, September 14–18, 2020*, pp. 67–82. TU Wien (2020)
7. Alhazov, A., Freund, R., Ivanov, S.: P systems with limiting the number of objects in membranes. In: Freund, R. (ed.) *Proceedings ICMC 2020, September 14–18, 2020*, pp. 83–98. TU Wien (2020)
8. Alhazov, A., Freund, R., Ivanov, S.: P systems with limited number of objects. *Journal of Membrane Computing* **3**, 1–9 (2021). <https://doi.org/10.1007/s41965-020-00068-6>
9. Alhazov, A., Freund, R., Ivanov, S.: When catalytic P systems with one catalyst can be computationally complete. *Journal of Membrane Computing* (2021). <https://doi.org/10.1007/s41965-021-00079-x>
10. Alhazov, A., Freund, R., Ivanov, S., Verlan, S.: (Tissue) P systems with vesicles of multisets. In: Csuhaj-Varjú, E., Dömösi, P., Vaszil, Gy. (eds.) *Proceedings 15th International Conference on Automata and Formal Languages, AFL 2017, Debrecen, Hungary, September 4-6, 2017. EPTCS*, vol. 252, pp. 11–25 (2017). <https://doi.org/10.4204/EPTCS.252.6>
11. Alhazov, A., Freund, R., Oswald, M., Verlan, S.: Partial halting and minimal parallelism based on arbitrary rule partitions. *Fundam. Inform.* **91**(1), 17–34 (2009). <https://doi.org/10.3233/FI-2009-0031>
12. Alhazov, A., Freund, R., Sosík, P.: Small P systems with catalysts or anti-matter simulating generalized register machines and generalized counter automata. *Comput. Sci. J. Moldova* **23**(3), 304–328 (2015), <http://www.math.md/publications/csjm/issues/v23-n3/11980/>
13. Alhazov, A., Freund, R., Verlan, S.: P systems working in maximal variants of the set derivation mode. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) *Membrane Computing – 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers*. Lecture Notes in Computer Science, vol. 10105, pp. 83–102. Springer (2017). https://doi.org/10.1007/978-3-319-54072-6_6
14. Ciobanu, G., Marcus, S., Păun, Gh.: New strategies of using the rules of a P system in a maximal way. power and complexity. *Romanian Journal of Information Science and Technology* **12**(2), 21–37 (2009)
15. Dassow, J., Păun, Gh.: *Regulated Rewriting in Formal Language Theory*. Springer (1989), <https://www.springer.com/de/book/9783642749346>
16. Freund, R.: Purely catalytic P systems: Two catalysts can be sufficient for computational completeness. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Yu. (eds.) *CMC14 Proceedings – The 14th International Conference on Membrane Computing, Chişinău, August 20–23, 2013*. pp. 153–166. Institute of Mathematics and Computer Science, Academy of Sciences of Moldova (2013), http://www.math.md/cmc14/CMC14_Proceedings.pdf

17. Freund, R.: P automata: New ideas and results. In: Bordihn, H., Freund, R., Nagy, B., Vaszil, Gy. (eds.) Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA 2016, Debrecen, Hungary, August 29–30, 2016. Proceedings. books@ocg.at, vol. 321, pp. 13–40. Österreichische Computer Gesellschaft (2016)
18. Freund, R.: How derivation modes and halting conditions may influence the computational power of P systems. *Journal of Membrane Computing* **2**(1), 14–25 (2020). <https://doi.org/10.1007/s41965-019-00028-9>
19. Freund, R., Kari, L., Oswald, M., Sosík, P.: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science* **330**(2), 251–266 (2005). <https://doi.org/10.1016/j.tcs.2004.06.029>
20. Freund, R., Leporati, A., Mauri, G., Porreca, A.E., Verlan, S., Zandron, C.: Flattening in (tissue) P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Yu., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, Lecture Notes in Computer Science*, vol. 8340, pp. 173–188. Springer (2014). https://doi.org/10.1007/978-3-642-54239-8_13
21. Freund, R., Oswald, M.: Catalytic and purely catalytic P automata: control mechanisms for obtaining computational completeness. In: Bensch, S., Drewes, F., Freund, R., Otto, F. (eds.) Fifth Workshop on Non-Classical Models for Automata and Applications – NCMA 2013, Umeå, Sweden, August 13 – August 14, 2013, Proceedings. books@ocg.at, vol. 294, pp. 133–150. Österreichische Computer Gesellschaft (2013)
22. Freund, R., Oswald, M., Păun, Gh.: Catalytic and purely catalytic P systems and P automata: Control mechanisms for obtaining computational completeness. *Fundam. Inform.* **136**(1–2), 59–84 (2015). <https://doi.org/10.3233/FI-2015-1144>
23. Freund, R., Păun, Gh.: How to obtain computational completeness in P systems with one catalyst. In: Neary, T., Cook, M. (eds.) *Proceedings Machines, Computations and Universality 2013, MCU 2013, Zürich, Switzerland, September 9–11, 2013. EPTCS*, vol. 128, pp. 47–61 (2013). <https://doi.org/10.4204/EPTCS.128.13>
24. Freund, R., Rogozhin, Yu., Verlan, S.: P systems with minimal left and right insertion and deletion. In: Durand-Lose, J., Jonoska, N. (eds.) *Unconventional Computation and Natural Computation – 11th International Conference, UCNC 2012, Orléan, France, September 3–7, 2012. Proceedings. Lecture Notes in Computer Science*, vol. 7445, pp. 82–93. Springer (2012). https://doi.org/10.1007/978-3-642-32894-7_9
25. Freund, R., Sosík, P.: On the power of catalytic P systems with one catalyst. In: Rozenberg, G., Salomaa, A., Sempere, J.M., Zandron, C. (eds.) *Membrane Computing – 16th International Conference, CMC 2015, Valencia, Spain, August 17–21, 2015, Revised Selected Papers. Lecture Notes in Computer Science*, vol. 9504, pp. 137–152. Springer (2015). https://doi.org/10.1007/978-3-319-28475-0_10
26. Freund, R., Verlan, S.: A formal framework for static (tissue) P systems. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, Lecture Notes in Computer Science*, vol. 4860, pp. 271–284. Springer (2007). https://doi.org/10.1007/978-3-540-77312-2_17
27. Freund, R., Verlan, S.: (tissue) P systems working in the k -restricted minimally or maximally parallel transition mode. *Nat. Comput.* **10**(2), 821–833 (2011). <https://doi.org/10.1007/s11047-010-9215-z>
28. Krithivasan, K., Păun, Gh., Ramanujan, A.: On controlled P systems. *Fundam. Inform.* **131**(3–4), 451–464 (2014). <https://doi.org/10.3233/FI-2014-1025>
29. Minsky, M.L.: *Computation. Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ (1967)

30. Păun, Gh.: Computing with membranes. *Journal of Computer and System Sciences* **61**(1), 108–143 (2000). <https://doi.org/10.1006/jcss.1999.1693>
31. Păun, Gh.: *Membrane Computing: An Introduction*. Springer (2002). <https://doi.org/10.1007/978-3-642-56196-2>
32. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press (2010)
33. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages*. Springer (1997). <https://doi.org/10.1007/978-3-642-59136-5>
34. The P Systems Website. <http://ppage.psystems.eu/>

Variants of Simple Purely Catalytic P Systems with Two Catalysts

Artiom Alhazov¹, Rudolf Freund², Sergiu Ivanov³, and Marion Oswald²

¹ Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chişinău, MD-2028, Moldova
artiom@math.md

² Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
{rudi,marion}@emcc.at

³ Université Évry, Paris-Saclay, Université Évry,
IBISC, 91020, Évry-Courcouronnes, France
sergiu.ivanov@ibisc.univ-evry.fr

Abstract. Catalytic P systems and purely catalytic P systems are among the first variants of membrane systems ever considered in this area. These variants of systems also feature some prominent computational complexity questions, and in particular the problem if only one catalyst in catalytic P systems and two catalysts in purely catalytic P systems are enough to allow for generating all recursively enumerable sets of multisets? Several additional ingredients have been shown to be sufficient for obtaining such results.

Last year we could show that using the derivation mode *maxobjects*, where we only take those multisets of rules which affect the maximal number of objects in the underlying configuration, one catalyst is sufficient for obtaining computational completeness without any other ingredients in catalytic P systems. In this paper we investigate the question whether we can obtain a similar result for purely catalytic P systems, i.e., we show that two catalysts in purely catalytic P systems are enough to allow for generating all recursively enumerable sets of multisets when using specific variants of the maximally parallel derivation mode: we take only those applicable multisets of rules which (i) generate the maximal number of objects, or (ii) yield the maximal difference in the number of objects between the newly generated configuration and the current configuration.

1 Introduction

Membrane systems were introduced more than two decades ago, see [34], as a multiset-rewriting model of computing inspired by the structure and the functioning of the living cell. The development of this fascinating area of biologically motivated computing models is documented in two textbooks, see [35] and [36]. For actual information see the P systems webpage [38] and the issues of the

Bulletin of the International Membrane Computing Society and of the Journal of Membrane Computing.

One basic feature of P systems already presented in [34] is the maximally parallel derivation mode, i.e., using non-extendable multisets of rules in every derivation step. The result of a computation can be extracted when the system halts, i.e., when no rule is applicable any more. Catalysts are special objects which allow only one object to evolve in its context (in contrast to promoters) and in their basic variant never evolve themselves, i.e., a catalytic rule is of the form $ca \rightarrow cv$, where c is a catalyst, a is a single object and v is a multiset of objects. On the other hand, non-catalytic rules in catalytic P systems are non-cooperative rules of the form $a \rightarrow v$. In this paper, we focus on purely catalytic P systems, which use catalytic rules only.

From the beginning (see [34] and [35]), the question how many catalysts are needed for obtaining computational completeness has been one of the most intriguing challenges regarding catalytic and purely catalytic P systems. In [21] it has already been shown that for (purely) catalytic P systems two (three) catalysts are enough for generating any recursively enumerable set of multisets, without any additional ingredients like a priority relation on the rules as used in the original definition.

As already known from the beginning (see [35]), without catalysts only regular (semi-linear) sets can be generated when using the standard maximal derivation mode and the standard halting mode, i.e., a result is extracted when the system halts with no rule being applicable any more. As shown, for example, in [25], using various additional ingredients, i.e., additional control mechanisms, one (two) catalysts can be sufficient for (purely) catalytic P systems: In P systems with label selection, only rules from one set of a finite number of sets of rules in each computation step are used; in time-varying P systems, the available sets of rules change periodically with time. For many other variants of P systems using specific control mechanism for the application of rules the interested reader is referred to the list of references, for example, see [1–8, 10–15, 17–20, 23–28, 31, 32].

In this paper we now continue the research started for catalytic P systems in [9] and investigate the following variants of the maximally parallel derivation mode: we take only those applicable multisets of rules which

1. generate the maximal number of objects, or
2. yield the maximal difference in the number of objects between the newly generated configuration and the current configuration.

For the variants with the maximal number of objects we can take those multisets of rules from the applicable multisets of rules which are non-extendable, but we can also take those sets with the maximal number of generated or maximal difference of objects without requesting the multisets of rules to fulfill the condition to be non-extendable.

2 Definitions

For an alphabet V , by V^* we denote the free monoid generated by V under the operation of concatenation, i.e., containing all possible strings over V . The *empty string* is denoted by λ . A *multiset* M with underlying set A is a pair (A, f) where $f : A \rightarrow \mathbb{N}$ is a mapping. If $M = (A, f)$ is a multiset then its *support* is defined as $\text{supp}(M) = \{x \in A \mid f(x) > 0\}$. A multiset is empty (respectively finite) if its support is the empty set (respectively a finite set). If $M = (A, f)$ is a finite multiset over A and $\text{supp}(M) = \{a_1, \dots, a_k\}$, then it can also be represented by the string $a_1^{f(a_1)} \dots a_k^{f(a_k)}$ over the alphabet $\{a_1, \dots, a_k\}$, and, moreover, all permutations of this string precisely identify the same multiset M . The set of all multisets over V is denoted by V° . The cardinality of a set or multiset M is denoted by $|M|$. For further notions and results in formal language theory we refer to textbooks like [16] and [37].

2.1 Register Machines

Register machines are well-known universal devices for computing on (or generating or accepting) sets of vectors of natural numbers. In this paper we only consider the generating case. The following definitions and propositions are given as in [10].

Definition 1. *A register machine is a construct*

$$M = (m, B, l_0, l_h, P) \quad \text{where}$$

- m is the number of registers,
- P is the set of instructions bijectively labeled by elements of B ,
- $l_0 \in B$ is the initial label, and
- $l_h \in B$ is the final label.

The instructions of M can be of the following forms:

- $p : (ADD(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.
Increase the value of register r by one, and non-deterministically jump to instruction q or s .
- $p : (SUB(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.
If the value of register r is not zero then decrease the value of register r by one (decrement case) and jump to instruction q , otherwise jump to instruction s (zero-test case).
- $l_h : HALT$.
Stop the execution of the register machine.

A configuration of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed. M is called deterministic if the ADD-instructions all are of the form $p : (ADD(r), q)$.

Throughout the paper, B_{ADD} denotes the set of labels of *ADD*-instructions $p : (ADD(r), q, s)$ of arbitrary registers r , and $B_{SUB(r)}$ denotes the set of labels of all *SUB*-instructions $p : (SUB(r), q, s)$ of a decrementable register r . Moreover, for any $p \in B \setminus \{l_h\}$, $Reg(p)$ denotes the register affected by the *ADD*- or *SUB*-instruction labeled by p .

In the *generating* case, a computation starts with all registers being empty and by executing the first instruction of P (labeled with l_0); it terminates with reaching the *HALT*-instruction and the output of a k -vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

For useful results on the computational power of register machines, we refer to [33]; for example, to prove our main theorem, we need the following formulation of results for register machines generating recursively enumerable sets of vectors of natural numbers with k components:

Proposition 1. *Register machines can generate any recursively enumerable set of vectors of natural numbers with k components using precisely $k + 2$ registers. Without loss of generality, we may assume that at the end of a generating computation the first two registers are empty, and, moreover, on the output registers, i.e., the last k registers, no *SUB*-instruction is ever used.*

Remark 1. For any register machine, without loss of generality we may assume that the first instruction is an *ADD*-instruction on register 1: in fact, given a register machine $M = (m, B, l_0, l_h, P)$ with having a another instruction as its first instruction, we can immediately construct an equivalent register machine M' which starts with an increment immediately followed by a decrement of the first register:

$$\begin{aligned} M' &= (m, B', l'_0, l_h, P'), \\ B' &= B \cup \{l'_0, l''_0\}, \\ P' &= P \cup \{l'_0 : (ADD(1), l''_0, l''_0), l''_0 : (SUB(1), l_0, l_0)\}. \end{aligned}$$

2.2 Simple Purely Catalytic P Systems

Taking into account the well-known flattening process, which means that computations in a P system with an arbitrary membrane structure can be simulated in a P system with only one membrane, e.g., see [22], in this paper we only consider simple purely catalytic P systems, i.e., with the simplest membrane structure of only one membrane:

Definition 2. *A simple purely catalytic P system is a construct*

$$\Pi = (V, C, T, w, \mathcal{R}) \quad \text{where}$$

- V is the alphabet of objects;
- $C \subset V$ is the set of catalysts;

- $T \subseteq (V \setminus C)$ is the alphabet of terminal objects;
- $w \in V^\circ$ is the multiset of objects initially present in the membrane region;
- \mathcal{R} is a finite set of evolution rules over V ; these evolution rules are catalytic rules of the forms $ca \rightarrow cv$, where $c \in C$ is a catalyst, a is an object from $V \setminus C$, and v is a multiset over $V \setminus C$.

The multiset in the single membrane region of Π constitutes a *configuration* of the P system. The *initial configuration* is given by the initial multiset w ; in case of accepting or computing P systems the input multiset w_0 is assumed to be added to w , i.e., the initial configuration then is ww_0 .

A transition between configurations is governed by the application of the evolution rules, which is done in a given derivation mode. The application of a rule $u \rightarrow v$ to a multiset M results in subtracting from M the multiset identified by u , and then in adding the multiset identified by v .

2.3 Variants of Derivation Modes

The definitions and the corresponding notions used in this subsection follow the definitions and notions elaborated in [30] and extend them for the purposes of this paper.

Given a P system $\Pi = (V, C, T, w, \mathcal{R})$, the set of multisets of rules applicable to a configuration C is denoted by $Appl(\Pi, C)$; this set also equals the set $Appl(\Pi, C, \text{asyn})$ of multisets of rules applicable in the *asynchronous derivation mode* (abbreviated *asyn*).

Given a multiset R of rules in $Appl(\Pi, C)$, we write $C \xrightarrow{R} C'$ if C' is the result of applying R to C . The number of objects affected by applying R to C is denoted by $Aff(C, R)$. The number of objects generated in C' by the right-hand sides of the rules applied to C with the multiset of rules R is denoted by $Gen(C, R)$. The difference between the number of objects in C' and C is denoted by $\Delta obj(C, R)$.

The set $Appl(\Pi, C, \text{sequ})$ denotes the set of multisets of rules applicable in the *sequential derivation mode* (abbreviated *sequ*), where in each derivation step exactly one rule is applied.

The standard parallel derivation mode used in P systems is the *maximally parallel derivation mode* (*max* for short). In the maximally parallel derivation mode, in any computation step of Π we choose a multiset of rules from \mathcal{R} in such a way that no further rule can be added to it so that the obtained multiset would still be applicable to the existing objects in the configuration, i.e., in simple P systems we only take applicable multisets of rules which cannot be extended by further (copies of) rules and are to be applied to the objects in the single membrane region:

$$Appl(\Pi, C, \text{max}) = \{R \in Appl(\Pi, C) \mid \text{there is no } R' \in Appl(\Pi, C) \text{ such that } R' \supset R\}.$$

We first consider the derivation mode $max_{objects}max$ where from the multisets of rules in $Appl(\Pi, C, max)$ only those are taken which affect the maximal number of objects. As with affecting the maximal number of objects, such multisets of rules are non-extendable anyway, we will also use the notation $max_{objects}$. Formally we may write:

$$Appl(\Pi, C, max_{objects}max) = \{R \in Appl(\Pi, C, max) \mid \\ \text{there is no } R' \in Appl(\Pi, C, max) \\ \text{such that } \text{Aff}(C, R) < \text{Aff}(C, R')\}$$

and

$$Appl(\Pi, C, max_{objects}) = \{R \in Appl(\Pi, C, async) \mid \\ \text{there is no } R' \in Appl(\Pi, C, async) \\ \text{such that } \text{Aff}(C, R) < \text{Aff}(C, R')\}.$$

As already mentioned, both definitions yield the same multiset of rules.

In addition to these well-known derivation modes, in this paper we also consider several new variants of derivation modes as already introduced in [9], where instead of looking at the number of affected objects we take into account the number of generated objects and the difference of objects between the derived configuration and the current configuration, respectively.

$max_{GENobjects}max$ a non-extendable multiset of rules R applicable to the current configuration C is only taken if the number of objects generated by the application of the rules in R to the configuration C is maximal with respect to the number of objects generated by the application of the rules in any other non-extendable multiset of rules R' to the configuration C :

$$Appl(\Pi, C, max_{GENobjects}max) = \{R \in Appl(\Pi, C, max) \mid \\ \text{there is no } R' \in Appl(\Pi, C, max) \\ \text{such that } \text{Gen}(C, R) < \text{Gen}(C, R')\}.$$

$max_{\Delta objects}max$ a non-extendable multiset of rules R applicable to the current configuration C is only taken if the difference $\Delta C = |C'| - |C|$ between the number of objects in the configuration C' obtained by the application of R and the number of objects in the underlying configuration C is maximal with respect to the differences in the number of objects obtained by applying any other non-extendable multiset of rules:

$$Appl(\Pi, C, max_{\Delta objects}max) = \{R \in Appl(\Pi, C, max) \mid \\ \text{there is no } R' \in Appl(\Pi, C, max) \\ \text{such that } \Delta obj(C, R) < \Delta obj(C, R')\}.$$

As in purely catalytic P system we only have catalytic rules, which on the left-hand side of the rule have exactly two objects, in both cases we only have to consider the right-hand sides of the rules when comparing rules.

Like for $max_{objects}max$ in comparison with $max_{objects}$ we now can also consider the variants of the other maximal derivation modes where we do not start with imposing the restriction of being non-extendable on the applicable multisets:

$max_{GENobjects}$ a multiset of rules R applicable to the current configuration C is only taken if the number of objects generated by the application of the rules in R to the configuration C is maximal with respect to the number of objects generated by the application of the rules in any other multiset of rules R' to the configuration C :

$$\begin{aligned} Appl(\Pi, C, max_{GENobjects}) = \{ & R \in Appl(\Pi, C, asyn) \mid \\ & \text{there is no } R' \in Appl(\Pi, C, asyn) \\ & \text{such that } Gen(C, R) < Gen(C, R') \}. \end{aligned}$$

$max_{\Delta objects}$ a multiset of rules R applicable to the current configuration C is only taken if the difference $\Delta C = |C'| - |C|$ between the number of objects in the configuration C' obtained by the application of R and the number of objects in the underlying configuration C is maximal with respect to the differences in the number of objects obtained by applying any other multisets of rules:

$$\begin{aligned} Appl(\Pi, C, max_{\Delta objects}) = \{ & R \in Appl(\Pi, C, asyn) \mid \\ & \text{there is no } R' \in Appl(\Pi, C, asyn) \\ & \text{such that } \Delta obj(C, R) < \Delta obj(C, R') \}. \end{aligned}$$

We illustrate the difference between these new derivation modes in the following example:

Example 1. Consider a simple purely catalytic P system with the initial configuration c_1c_2aa and the following rules:

1. $c_1a \rightarrow c_1$
2. $c_2a \rightarrow c_2b$
3. $c_2a \rightarrow c_2bb$

We immediately observe the following:

1. $Gen(c_1c_2aa, \{c_1a \rightarrow c_1\}) = 1$,
2. $Gen(c_1c_2aa, \{c_2a \rightarrow c_2b\}) = 2$,
3. $Gen(c_1c_2aa, \{c_2a \rightarrow c_2bb\}) = 3$.

In case of the derivation mode $max_{GENobjects}max$, the multiset of rules $\{c_1a \rightarrow c_1, c_2a \rightarrow c_2bb\}$ has to be applied. Hence, the only possible derivation with the derivation mode $max_{GENobjects}max$ is $c_1c_2aa \xrightarrow{\{c_1a \rightarrow c_1, c_2a \rightarrow c_2bb\}} c_1c_2bb$.

In this special case,

$$Appl(\Pi, c_1c_2aa, max_{GENobjects}max) = Appl(\Pi, c_1c_2aa, max_{objects}).$$

If we do not start from non-extendable multisets of rules, we obtain the same results, i.e., in the derivation mode $max_{GENobjects}$, the multiset of rules $\{c_1a \rightarrow c_1, c_2a \rightarrow c_2bb\}$ has to be applied, and the only possible derivation with the derivation mode $max_{GENobjects}$ is $c_1c_2aa \xrightarrow{\{c_1a \rightarrow c_1, c_2a \rightarrow c_2bb\}} c_1c_2bb$.

In the same way, for the difference of generated and consumed objects we obtain:

1. $\Delta obj(c_1c_2aa, \{c_1a \rightarrow c_1\}) = -1$,
2. $\Delta obj(c_1c_2aa, \{c_2a \rightarrow c_2b\}) = 0$,
3. $\Delta obj(c_1c_2aa, \{c_2a \rightarrow c_2bb\}) = 1$.

As for the derivation mode $max_{\Delta objects}max$, also for the derivation mode $max_{GENobjects}max$ we obtain that the multiset of rules $\{c_1a \rightarrow c_1, c_2a \rightarrow c_2bb\}$ has to be applied and that the only possible derivation with the derivation mode $max_{\Delta objects}max$ is $c_1c_2aa \xrightarrow{\{c_1a \rightarrow c_1, c_2a \rightarrow c_2bb\}} c_1c_2bb$.

On the other hand, if we do not start from non-extendable multisets of rules, now the rule $c_1a \rightarrow c_1$ must not be applied because it would decrease the number of objects, i.e., in the derivation mode $max_{\Delta objects}$ we obtain that the – *not* non-extendable – multiset of rules $\{c_2a \rightarrow c_2bb\}$ has to be applied, and the only possible derivation with the derivation mode $max_{\Delta objects}$ is $c_1c_2aa \xrightarrow{\{c_2a \rightarrow c_2bb\}} c_1c_2abb$.

2.4 Computations in a P System

The P system continues with applying multisets of rules according to the derivation mode until there remain no applicable rules in the single region of Π , i.e., as usual, with all these variants of derivation modes as defined above, we consider *halting computations*.

When the system *halts*, we consider the number of objects from T contained at that moment in the single membrane region as the *result* of the underlying computation of Π .

We would like to emphasize that as results we only take the objects from the terminal alphabet T , especially the catalysts are not counted to the result of a computation. On the other hand, with all the proofs given in this paper, except for the catalysts no other “garbage” remains in the membrane region at the end of a halting computation, i.e., we could even omit T .

3 Simple Purely Catalytic P Systems Working in the Derivation Modes $max_{\Delta objects}max$, $max_{GEN objects}max$, $max_{\Delta objects}$, or $max_{GEN objects}$

In this section we show how the new derivation modes allow for simulating register machines by purely catalytic P systems with one catalyst less than in the original proofs given in [21], which are the first results obtained for purely catalytic P systems of that kind when using specific variants of derivation modes themselves without in addition using specific control mechanisms as, for example, in [25].

Theorem 1. *For any register machine with two decrementable registers we can construct a simple purely catalytic P system with only two catalysts, working in one of the derivation modes $max_{\Delta objects}max$, $max_{GEN objects}max$, $max_{\Delta objects}$, or $max_{GEN objects}$, which can simulate any computation of the register machine.*

Proof. As in purely catalytic P systems we only have a bounded number of rules – bounded by the number of catalysts – which can be executed in one derivation step, we cannot apply the proof idea used in the proofs elaborated in [9], where in some sense the weak priority of catalytic rules over non-catalytic rules taken from the set of applicable non-extendable multisets of rules in simple catalytic P systems was mimicked when using the derivation mode $max_{objects}$, see [10].

Instead, we have to go back to the original construction as elaborated in [21], yet we also take into consideration ideas related to energy-controlled P systems as described in [4], using dummy objects like energy to control the correct application of rules. Now having the new variants of derivation modes, we can avoid the trap rules and even more, instead of three catalysts, we only need two catalysts to obtain the desired completeness result.

Given an arbitrary register machine $M = (m, B, l_0, l_h, P)$ with two decrementable registers we will construct a corresponding simple purely catalytic P system with two catalysts

$$\Pi = (V, \{c_1, c_2\}, T, c_1c_2l_0l'_0, \mathcal{R})$$

simulating M . Without loss of generality, we may assume that, depending on its use as an accepting or generating or computing device, the register machine M , as stated in Proposition ??, Proposition 1, and Proposition ??, fulfills the condition that on the output registers we never apply any *SUB*-instruction. Moreover, according to Remark 1 we may assume that the first instruction is an *ADD*-instruction on the first register. Finally, we assume the n decrementable registers to be the first ones.

The following proof is elaborated for all the derivation modes $max_{\Delta objects}max$, $max_{GEN objects}max$, $max_{\Delta objects}$, and $max_{GEN objects}$; only a few subtle technical details have to be mentioned additionally.

The main part of the proof is to show how to simulate the instructions of M in Π ; in all cases we have to take care that both catalysts are kept busy to

guarantee that the simulation is executed in a correct way. The extensive use of the dummy object d guarantees that one of the rules using the catalysts c_1 and c_2 must be used if possible, i.e., a catalyst can only stay idle if the underlying configuration does not contain any object which can evolve together with the catalyst. On the other hand, the priority between different rules for a catalyst is guarded by the number of objects on the right-hand side of the rules, which argument applies for all the derivation modes under consideration, as every rule in a purely catalytic P system has exactly two objects on its left-hand side.

The only special detail which arises is that in the derivation mode $max_{\Delta objects}$, where, as in Example 1, the rules

$$c_1d \rightarrow c_1 \quad \text{and} \quad c_2d \rightarrow c_2$$

erasing the “energy” object d can only be used at the end of a computation when no other rules can be applied any more.

Before giving the whole construction of the simple purely catalytic P system, we mention that the main basis for choosing the right number of the objects d on the right-hand side of the rules is based on the importance and rôle of the rules

$$c_1a_1 \rightarrow c_1\hat{a}_1dd \quad \text{and} \quad c_2a_2 \rightarrow c_2\hat{a}_2dd$$

which should only be applicable in the first step of the simulation of a decrement instruction on register 1 and register 2, respectively. As usually done in corresponding proofs, the number of objects a_r in a configuration represents the number stored in register r at that moment of the computation. Objects a_r for $r > 2$ are never changed again, as they represent output registers.

$$\begin{aligned} V &= \{a_r \mid 1 \leq r \leq m\} \cup \{\hat{a}_r \mid 1 \leq r \leq 2\} \\ &\cup \{p, p' \mid p \in B_{ADD} \cup \{l_h\}\} \\ &\cup \{p, p', \bar{p}, \hat{p} \mid p \in B_{SUB(r)}, 1 \leq r \leq 2\} \\ &\cup \{c_1, c_2, d\}, \\ T &= \{a_r \mid 3 \leq r \leq m\}. \end{aligned}$$

B_{ADD} and $B_{SUB(r)}$ denote the set of labels of ADD- and SUB-instructions of the register machine M .

The set \mathcal{R} of catalytic rules can be captured from the description of how the simulation of the register machine instructions works as described in the following:

- $p : (ADD(r), q, s)$, with $p \in B_{ADD}$, $q, s \in B$, $1 \leq r \leq m$.

An ADD-instruction can be simulated in one step by letting every catalyst make one evolution step:

$$\begin{aligned} c_1p &\rightarrow c_1qq'a_r d \quad \text{or} \quad c_1p \rightarrow c_1ss'a_r d, \\ c_2p' &\rightarrow c_2d^4. \end{aligned}$$

The dummy objects d are used to guarantee that the rules given above, with in sum 5 objects on their right-hand sides, have priority over the rules $c_1 a_1 \rightarrow c_1 \hat{a}_1 d^2$ and $c_2 a_2 \rightarrow c_2 \hat{a}_2 d^2$, respectively, with in sum only 4 objects on their right-hand sides.

- $p : (SUB(r), q, s)$, with $p \in B_{SUB}$, $q, s \in B$, $1 \leq r \leq 2$.

decrement case:

If the value of register r (denoted by $|reg(r)|$) is not zero then the number of register objects a_r is decreased by one using the corresponding rule $c_r a_r \rightarrow c_r \hat{a}_r d^2$ in the first step of the simulation. In sum three steps are needed for the simulation, see table below.

zero-test case:

If the value of register r is zero, then the corresponding catalyst is already free for eliminating the label object p so that already in the second step of the simulation the simulation of the next instruction s can be initiated. In sum only two steps are needed for the simulation of this case, see Table 1.

The following table summarizes the rules to be used for the simulation of the SUB-instruction on register r , $r \in \{1, 2\}$, i.e., we use the following rules, resulting in different sets of objects depending on the value of $|reg(r)|$, thereby neglecting the objects d ; we emphasize that the simulation is *deterministic*.

Table 1. Simulation of SUB-instruction

step	$ reg(r) $	rule for c_r	rule for c_{3-r}	resulting objects
first	> 0	$c_r a_r \rightarrow c_r \hat{a}_r d^2$	$c_{3-r} p' \rightarrow c_{3-r} \bar{p} d^{10}$	p, \bar{p}, \hat{a}_r
	$= 0$	$c_r p \rightarrow c_r d^2$	$c_{3-r} p' \rightarrow c_{3-r} \bar{p} d^{10}$	\bar{p}
second	> 0	$c_r \bar{p} \rightarrow c_r \hat{p} d^3$	$c_{3-r} p \rightarrow c_{3-r} d^9$	\hat{a}_r, \hat{p}
	$= 0$	$c_r d \rightarrow c_r (*)$	$c_{3-r} \bar{p} \rightarrow c_{3-r} s s' d^6$	s, s'
third	> 0	$c_r \hat{p} \rightarrow c_1 q q' d^2$	$c_{3-r} \hat{a}_r \rightarrow c_{3-r} d^4$	q, q'

The rule $c_r d \rightarrow c_r$ marked with (*) is only applied in the derivation modes $max_{\Delta objects} max$ and $max_{GEN objects} max$ as well as $max_{GEN objects}$, whereas in the derivation mode $max_{\Delta objects}$ it will not be applied as it would decrease the difference between generated and consumed objects.

- $l_h : HALT$.

When a computation of the register machine M ends with reaching the HALT-instruction, the simulating P system Π uses the following rules:

$$c_1 l_h \rightarrow c_1 d d \quad \text{and} \quad c_2 l'_h \rightarrow c_2 d d.$$

After the register machine has halted (with the first two registers being empty), which is simulated by the rules above, finally all dummy objects generated during the simulation steps before are deleted by using the rules

$$c_1d \rightarrow c_1 \quad \text{and} \quad c_2d \rightarrow c_2.$$

Whereas in the derivation modes $max_{\Delta objects}max$ and $max_{GEN objects}max$ as well as $max_{GEN objects}$ some of these objects d can already be erased during the simulation of SUB-instructions, see above, in the derivation mode $max_{\Delta objects}$, these erasing rules are only executed at the end of the computation.

The construction has been chosen in such a way that it works for all these derivation modes. Yet for the derivation mode $max_{\Delta objects}$ it is essential that the right-hand side contains at least three objects to enforce the application of all rules except the deletion rules $c_1d \rightarrow c_1$ and $c_2d \rightarrow c_2$. Otherwise the given construction of rules would not work correctly because of the missing condition for the multisets to be applied of being non-extendable. On the other hand, for the other derivation modes, i.e., $max_{\Delta objects}max$ and $max_{GEN objects}max$ as well as $max_{GEN objects}$, the construction would still work correctly if on the right-hand side of *all* rules, obviously again except the deletion rules $c_1d \rightarrow c_1$ and $c_2d \rightarrow c_2$, one dummy object d less is used.

These observations complete the proof. \square

Corollary 1. *Any recursively enumerable set of (vectors of) natural numbers can be generated by a simple purely catalytic P system with only two catalysts working in one of the derivation modes $max_{\Delta objects}max$, $max_{GEN objects}max$, $max_{\Delta objects}$, or $max_{GEN objects}$.*

The results elaborated in this section can be generalized to the case of $k \geq 2$ decrementable registers, yet we leave the challenging technical details, following the ideas in [21], to the interested reader.

In sum, we then obtain the following result:

Proposition 2. *Purely catalytic P systems working in any of the derivation modes $max_{\Delta objects}max$, $max_{GEN objects}max$, $max_{\Delta objects}$, or $max_{GEN objects}$ are computationally complete, i.e., they can compute any partial recursive relation on natural numbers.*

4 Conclusion

In this paper we have continued our investigation of the new derivation modes $max_{GEN objects}max$, $max_{\Delta objects}max$, $max_{GEN objects}$, and $max_{\Delta objects}$ as introduced in [9], now applied in simple purely catalytic P systems. In contrast to the proof technique used there for catalytic P systems, we here had to go back to the original construction as elaborated in [21], yet also using ideas related to energy-controlled P systems as described in [4] in order to be able to show that *two* catalysts are enough for generating all recursively enumerable sets of multisets. The results obtained in this paper can also be extended to P systems dealing with strings, following the definitions and notions as, for example, used in [29], thus showing computational completeness for computing with strings.

Acknowledgements

Artiom Alhazov acknowledges project 20.80009.5007.22 “Intelligent information systems for solving ill-structured problems, processing knowledge and big data” by the National Agency for Research and Development. Sergiu Ivanov is partially supported by the Paris region via the project DIM RFSI n°2018-03 “Modèles informatiques pour la reprogrammation cellulaire”.

References

1. Alhazov, A., Aman, B., Freund, R.: P systems with anti-matter. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) *Membrane Computing – 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers*. Lecture Notes in Computer Science, vol. 8961, pp. 66–85. Springer (2014). https://doi.org/10.1007/978-3-319-14370-5_5
2. Alhazov, A., Aman, B., Freund, R., Păun, Gh.: Matter and anti-matter in membrane systems. In: Jürgensen, H., Karhumäki, J., Okhotin, A. (eds.) *Descriptive Complexity of Formal Systems – 16th International Workshop, DCFS 2014, Turku, Finland, August 5–8, 2014. Proceedings*. Lecture Notes in Computer Science, vol. 8614, pp. 65–76. Springer (2014). https://doi.org/10.1007/978-3-319-09704-6_7
3. Alhazov, A., Freund, R.: P systems with toxic objects. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) *Membrane Computing – 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers*. Lecture Notes in Computer Science, vol. 8961, pp. 99–125. Springer (2014). https://doi.org/10.1007/978-3-319-14370-5_7
4. Alhazov, A., Freund, R., Ivanov, S.: Variants of energy-controlled P systems. In: *Proceedings of NIT 2016* (2016)
5. Alhazov, A., Freund, R., Ivanov, S.: Variants of P systems with activation and blocking of rules. *Nat. Comput.* **18**(3), 593–608 (2019). <https://doi.org/10.1007/s11047-019-09747-5>
6. Alhazov, A., Freund, R., Ivanov, S.: Catalytic P systems with weak priority of catalytic rules. In: Freund, R. (ed.) *Proceedings ICMC 2020, September 14–18, 2020*, pp. 67–82. TU Wien (2020)
7. Alhazov, A., Freund, R., Ivanov, S.: P systems with limiting the number of objects in membranes. In: Freund, R. (ed.) *Proceedings ICMC 2020, September 14–18, 2020*, pp. 83–98. TU Wien (2020)
8. Alhazov, A., Freund, R., Ivanov, S.: P systems with limited number of objects. *Journal of Membrane Computing* **3**, 1–9 (2021). <https://doi.org/10.1007/s41965-020-00068-6>
9. Alhazov, A., Freund, R., Ivanov, S.: Variants of simple P systems with one catalyst being computationally complete. In: Vaszil, Gy. (ed.) *Proceedings ICMC 2021* (2021)
10. Alhazov, A., Freund, R., Ivanov, S.: When catalytic P systems with one catalyst can be computationally complete. *Journal of Membrane Computing* (2021). <https://doi.org/10.1007/s41965-021-00079-x>
11. Alhazov, A., Freund, R., Ivanov, S., Verlan, S.: (Tissue) P systems with vesicles of multisets. In: Csuhaaj-Varjú, E., Dömösi, P., Vaszil, Gy. (eds.) *Proceedings*

- 15th International Conference on Automata and Formal Languages, AFL 2017, Debrecen, Hungary, September 4-6, 2017. EPTCS, vol. 252, pp. 11–25 (2017). <https://doi.org/10.4204/EPTCS.252.6>
12. Alhazov, A., Freund, R., Leporati, A., Oswald, M., Zandron, C.: (Tissue) P systems with unit rules and energy assigned to membranes. *Fundam. Informaticae* **74**(4), 391–408 (2006)
 13. Alhazov, A., Freund, R., Oswald, M., Verlan, S.: Partial halting and minimal parallelism based on arbitrary rule partitions. *Fundam. Inform.* **91**(1), 17–34 (2009). <https://doi.org/10.3233/FI-2009-0031>
 14. Alhazov, A., Freund, R., Sosík, P.: Small P systems with catalysts or anti-matter simulating generalized register machines and generalized counter automata. *Comput. Sci. J. Moldova* **23**(3), 304–328 (2015)
 15. Alhazov, A., Freund, R., Verlan, S.: P systems working in maximal variants of the set derivation mode. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) *Membrane Computing – 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers. Lecture Notes in Computer Science*, vol. 10105, pp. 83–102. Springer (2017). https://doi.org/10.1007/978-3-319-54072-6_6
 16. Dassow, J., Păun, Gh.: *Regulated Rewriting in Formal Language Theory*. Springer (1989)
 17. Freund, R.: Energy-controlled P systems. In: Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) *Membrane Computing*, pp. 247–260. Springer (2003)
 18. Freund, R.: Purely catalytic P systems: Two catalysts can be sufficient for computational completeness. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Yu. (eds.) *CMC14 Proceedings – The 14th International Conference on Membrane Computing, Chişinău, August 20–23, 2013*, pp. 153–166. Institute of Mathematics and Computer Science, Academy of Sciences of Moldova (2013)
 19. Freund, R.: P automata: New ideas and results. In: Bordihn, H., Freund, R., Nagy, B., Vaszil, Gy. (eds.) *Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA 2016, Debrecen, Hungary, August 29–30, 2016. Proceedings. books@ocg.at*, vol. 321, pp. 13–40. Österreichische Computer Gesellschaft (2016)
 20. Freund, R.: How derivation modes and halting conditions may influence the computational power of P systems. *Journal of Membrane Computing* **2**(1), 14–25 (2020). <https://doi.org/10.1007/s41965-019-00028-9>
 21. Freund, R., Kari, L., Oswald, M., Sosík, P.: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science* **330**(2), 251–266 (2005). <https://doi.org/10.1016/j.tcs.2004.06.029>
 22. Freund, R., Leporati, A., Mauri, G., Porreca, A.E., Verlan, S., Zandron, C.: Flattening in (tissue) P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Yu., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, Lecture Notes in Computer Science*, vol. 8340, pp. 173–188. Springer (2014). https://doi.org/10.1007/978-3-642-54239-8_13
 23. Freund, R., Oswald, M.: Partial halting in P systems. *Int. J. Found. Comput. Sci.* **18**(6), 1215–1225 (2007). <https://doi.org/10.1142/S0129054107005261>
 24. Freund, R., Oswald, M.: Catalytic and purely catalytic P automata: control mechanisms for obtaining computational completeness. In: Bensch, S., Drewes, F., Freund, R., Otto, F. (eds.) *Fifth Workshop on Non-Classical Models for Automata and Applications – NCMA 2013, Umeå, Sweden, August 13 – August 14, 2013, Proceedings. books@ocg.at*, vol. 294, pp. 133–150. Österreichische Computer Gesellschaft (2013)

25. Freund, R., Oswald, M., Păun, Gh.: Catalytic and purely catalytic P systems and P automata: Control mechanisms for obtaining computational completeness. *Fundam. Inform.* **136**(1–2), 59–84 (2015). <https://doi.org/10.3233/FI-2015-1144>
26. Freund, R., Păun, Gh.: How to obtain computational completeness in P systems with one catalyst. In: Neary, T., Cook, M. (eds.) *Proceedings Machines, Computations and Universality 2013, MCU 2013*, Zürich, Switzerland, September 9–11, 2013. *EPTCS*, vol. 128, pp. 47–61 (2013). <https://doi.org/10.4204/EPTCS.128.13>
27. Freund, R., Păun, Gh., Pérez-Jiménez, M.J.: Polarizationless P systems with active membranes working in the minimally parallel mode. In: Akl, S.G., Calude, C.S., Dinneen, M.J., Rozenberg, G., Wareham, T. (eds.) *Unconventional Computation, 6th International Conference, UC 2007*, Kingston, Canada, August 13–17, 2007, *Proceedings. Lecture Notes in Computer Science*, vol. 4618, pp. 62–76. Springer (2007). https://doi.org/10.1007/978-3-540-73554-0_8
28. Freund, R., Rogozhin, Yu., Verlan, S.: P systems with minimal left and right insertion and deletion. In: Durand-Lose, J., Jonoska, N. (eds.) *Unconventional Computation and Natural Computation – 11th International Conference, UCNC 2012*, Orléan, France, September 3–7, 2012. *Proceedings. Lecture Notes in Computer Science*, vol. 7445, pp. 82–93. Springer (2012). https://doi.org/10.1007/978-3-642-32894-7_9
29. Freund, R., Sosík, P.: On the power of catalytic P systems with one catalyst. In: Rozenberg, G., Salomaa, A., Sempere, J.M., Zandron, C. (eds.) *Membrane Computing – 16th International Conference, CMC 2015*, Valencia, Spain, August 17–21, 2015, *Revised Selected Papers. Lecture Notes in Computer Science*, vol. 9504, pp. 137–152. Springer (2015). https://doi.org/10.1007/978-3-319-28475-0_10
30. Freund, R., Verlan, S.: A formal framework for static (tissue) P systems. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, Lecture Notes in Computer Science*, vol. 4860, pp. 271–284. Springer (2007). https://doi.org/10.1007/978-3-540-77312-2_17
31. Freund, R., Verlan, S.: (Tissue) P systems working in the k -restricted minimally or maximally parallel transition mode. *Nat. Comput.* **10**(2), 821–833 (2011). <https://doi.org/10.1007/s11047-010-9215-z>
32. Krithivasan, K., Păun, Gh., Ramanujan, A.: On controlled P systems. *Fundam. Inform.* **131**(3–4), 451–464 (2014). <https://doi.org/10.3233/FI-2014-1025>
33. Minsky, M.L.: *Computation. Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ (1967)
34. Păun, Gh.: Computing with membranes. *Journal of Computer and System Sciences* **61**(1), 108–143 (2000). <https://doi.org/10.1006/jcss.1999.1693>
35. Păun, Gh.: *Membrane Computing: An Introduction*. Springer (2002). <https://doi.org/10.1007/978-3-642-56196-2>
36. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press (2010)
37. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages*. Springer (1997). <https://doi.org/10.1007/978-3-642-59136-5>
38. The P Systems Website. <http://ppage.psystems.eu/>

Evaluating Space Measures in P Systems

Artiom Alhazov¹, Alberto Leporati², Luca Manzoni³, Giancarlo Mauri², and Claudio Zandron²

¹ Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chişinău, MD-2028, Moldova
`artiom@math.md`

² Dipartimento di Informatica, Sistemistica e Comunicazione (DISCo)
Università degli Studi di Milano-Bicocca
Viale Sarca 336, 20126 Milan, Italy
`{alberto.leporati,giancarlo.mauri,claudio.zandron}@unimib.it`

³ Dipartimento di Matematica e Geoscienze
Università degli Studi di Trieste
`lmanzoni@units.it`

Abstract. P systems with active membranes allow the creation of an exponential amount of resources in a polynomial number of computation steps, by dividing existing membranes. The resources can then be used in parallel to attack computationally hard problems. Time and space complexity classes for active membrane systems have been introduced, to characterize classes of problems that can be solved by different membrane systems making use of different resources.

Initially, space complexity classes were introduced assuming that every single object or membrane requires some constant physical space, corresponding to store information in a unary notation. A different definition, based on the use binary numbers, was also considered having in mind possible implementation of P systems in silico. In both cases, the elements contributing to the definition of the space required by a system (namely, the total number of membranes, the total number of objects, the types of different membranes, and the types of different objects) was considered as a whole. In this paper, we consider a different definition for space complexity classes in the framework of P systems, where each of the previous elements is considered independently. We review the principal results related to the solution of different computationally hard problems presented in the literature, highlighting the requirement of every single resource in each solution. A short discussion concerning possible alternative solutions requiring different resources is presented.

Keywords: Membrane Systems, Computational Complexity, Space Complexity

1 Introduction

P systems with active membranes have been introduced in [28], considering the idea of generating new membranes through division of existing ones. The

exponential amount of resources that can be obtained in this way, in a polynomial number of computation steps, naturally leads to the definition of new complexity classes to be compared with the standard ones.

Initially, the research activity focused on the investigation of time complexity. It was proved that, in order to go beyond the complexity class **P**, the creation of new membranes is a necessary feature to gain enough computation efficiency [43], unless non-confluent systems are used [36]. In [37] it was proved that P systems with active membranes can solve all problems in the class **PSPACE** in polynomial time, a result which is valid also for uniform systems, as proved in [7]. Relations with the classes **EXP** and **EXSPACE** were investigated in [35].

A series of works then defined various complexity classes characterized by P systems that make use of different features. For instance, the works [13,14] focused on the crucial role of membrane dissolution; polarizationless systems have been investigated in [5,6,15,12]; constraints on membrane division [23] or on depth of membrane structure [17] have been the subjects of other papers, while [39,40] focused on the role of cooperation.

More recently, other aspects have also been studied. In [1,26] a different kind of membrane division, called separation (since objects are separated between new membranes, rather than duplicated) is considered in the framework of P systems with active membranes; in [25] such kind of rules are applied in a different variant of P systems, having proteins on membranes. In [8,11] solutions for **SAT** are proposed which use different strategies than previously proposed solutions. Systems of a shallow depth are the subject of [18,19,20]. A recent survey on different strategies to approach computationally hard problems by P systems with active membranes can be found in [38].

A natural research topic that has been approached after the first works on time complexity concerns space complexity, a notion introduced for the first time in the framework of P systems in [30]. The definition was based on a hypothetical real implementation by means of biochemical materials such as cellular membranes and chemical molecules. Under this assumption, it was assumed that every single object or membrane requires some constant physical space, and this is equivalent to using a unary encoding to represent multiplicities. The relations between standard computational complexity classes and the space complexity classes defined in these terms have been studied, both when at least a linear amount of space is used [31,32], as well as when only sublinear [34] or even constant amount of space [16] is available. A recent survey concerning results obtained by considering different bounds on space can be found in [42].

A different approach to define space complexity for P systems was considered in [2], focusing the definition of space on the simulative point of view. In fact, by considering an implementation of P systems in silico (like the ones in, e.g., [9,10]), it is not strictly necessary to store information concerning every single object: the multiplicity of each object in each membrane can be stored using binary numbers, thus reducing the amount of needed space.

In both cases, the definition of the space required by a system was considered as a unique total measure obtained by considering all the elements contributing

to it: the total number of membranes, the total number of objects, the types of different membranes, and the types of different objects.

In this paper, we introduce a different definition for space complexity classes in the framework of P systems, where each of the previous elements is considered independently. This allows to consider the amount of each element separately, thus highlighting the requirement of each of them in every solution considered. We review the principal results present in the literature and we discuss possible alternative solutions, requiring different resources balances.

The paper is organized as follows. In Section 2 we recall some definitions concerning P systems with active membranes and space requirements in P systems computations. In Section 3, we introduce a definition of space to measure the contribution by each component, namely the total number of membranes, the total number of objects, the types of different membranes, and the types of different objects. Moreover, we survey some main result concerning complexity in the framework of P systems, highlighting the use of each single resource. Section 4 presents some conclusions and future research topics.

2 Basic definitions

In this section, we shortly recall some definitions that will be useful while reading the rest of the paper. For a complete introduction to P systems, we refer the reader to *The Oxford Handbook of Membrane Computing* [29].

Definition 1. A P system with active membranes having initial degree $d \geq 1$ is a tuple $\Pi = (\Gamma, \Lambda, \mu, w_{h_1}, \dots, w_{h_d}, R)$, where:

- Γ is an alphabet, i.e., a finite non-empty set of symbols, usually called objects; in the following, we assume $\Gamma = \{O_1, O_2, \dots, O_n\}$;
- Λ is a finite set of labels for the membranes;
- μ is a membrane structure (i.e., a rooted unordered tree, usually represented by nested brackets) consisting of d membranes, labelled by elements of Λ , defining regions (the space between a membrane and all membranes immediately inside it, if any);
- w_{h_1}, \dots, w_{h_d} , with $h_1, \dots, h_d \in \Lambda$, are strings over Γ describing the initial multisets of objects placed in the d regions of μ ;
- R is a finite set of rules over Γ .

Membranes are polarized, that is, they have an attribute called *electrical charge*, which can be neutral (0), positive (+) or negative (–).

A P system can make a computation step by applying its rules to modify the membrane structure and/or the membrane content. The following types of rules can be used during the computation:

- *Object evolution rules*, of the form $[a \rightarrow w]_h^\alpha$
They can be applied inside a membrane labelled by h , having charge α and containing at least an occurrence of the object a ; the copy of the object a to which the rule is applied is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by the objects in w).

- *Send-in communication rules*, of the form $a []_h^\alpha \rightarrow [b]_h^\beta$
They can be applied to a membrane labelled by h , having charge α and such that the external region contains at least an occurrence of the object a ; the copy of the object a to which the rule is applied is sent into h becoming b and, simultaneously, the charge of h is changed to β .
- *Send-out communication rules*, of the form $[a]_h^\alpha \rightarrow []_h^\beta b$
They can be applied to a membrane labelled by h , having charge α and containing at least an occurrence of the object a ; the copy of the object a to which the rule is applied is sent out from h to the outside region becoming b and, simultaneously, the charge of h is changed to β .
- *Dissolution rules*, of the form $[a]_h^\alpha \rightarrow b$
They can be applied to a membrane labelled by h , having charge α and containing at least an occurrence of the object a ; the copy of the object a to which the rule is applied is replaced by b , the membrane h is dissolved and its contents are left in the surrounding region.
- *Elementary division rules*, of the form $[a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma$
They can be applied to a membrane labelled by h , having charge α , containing at least an occurrence of the object a but having no other membrane inside (in this case the membrane is said to be *elementary*); the membrane is divided into two membranes having both label h and charges β and γ , respectively; the copy of the object a to which the rule is applied is replaced, respectively, by b and c in the two new membranes, while the other objects in the initial multiset are copied to both membranes.
- *(Weak) Non-elementary division rules*, of the form $[a]_h^\alpha \rightarrow [b]_h^\beta [c]_h^\gamma$
These rules operate just like division for elementary membranes, but they can be applied to non-elementary membranes, containing membrane substructures and having a label h . Like the objects, the substructures inside the dividing membrane are replicated in the two new copies of it.

A configuration of a P system with active membranes is described by the current membrane structure (including the electrical charge of each membrane) and the multisets located in the corresponding regions. A computation step changes the current configuration according to the following set of principles:

- Each object and membrane can be subject to at most one rule per step, except for object evolution rules: this means that inside each membrane several evolution rules can be applied simultaneously, but each membrane can be involved only in a single communication, dissolution, or division rule per step.
- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, dissolution or division rules must be subject to exactly one of them (unless the current charge of the membrane prohibits it, and according to the fact that a membrane can be involved in a single communication, dissolution, or division rule per step). The same principle applies to each membrane that can be involved in communication, dissolution, or division rules. In other words, the only objects and membranes

that do not evolve are those associated with no rule, or only to rules that are not applicable due to the electrical charges.

- When several conflicting rules can be applied at the same time, a nondeterministic choice is performed; this implies that, in general, multiple possible configurations can be reached as a result of a computation step.
- In each computation step, all the chosen rules are applied simultaneously (in an atomic way). However, in order to clarify the operational semantics, each computation step is conventionally described as a sequence of micro-steps as follows. First, all evolution rules are applied inside the elementary membranes, followed by all communication, dissolution and division rules involving the membranes themselves; this process is then repeated on the membranes containing them, and so on towards the root (outermost membrane). In other words, the membranes evolve only after their internal configuration has been updated. For instance, before a membrane division occurs, all chosen object evolution rules must be applied inside it; in this way, the objects that are duplicated during the division are already the final ones.
- The outermost membrane cannot be divided or dissolved, and any object sent out from it cannot re-enter the system again.

A *halting computation* of the P system Π is a finite sequence of configurations $\mathcal{C} = (\mathcal{C}_0, \dots, \mathcal{C}_k)$, where \mathcal{C}_0 is the initial configuration, every \mathcal{C}_{i+1} is reachable from \mathcal{C}_i via a single computation step, and no rules of Π are applicable in \mathcal{C}_k . If this last condition is never reached (that is, in each configuration of the sequence there is at least one applicable rule), then a *non-halting computation* $\mathcal{C} = (\mathcal{C}_i : i \in \mathbb{N})$ is obtained, that consists of infinitely many configurations, again starting from the initial one and generated by successive computation steps.

P systems can be used as language *recognizers* by employing two distinguished objects *yes* and *no*; exactly one of these must be sent out from the outermost membrane, and only in the last step of each computation, in order to signal acceptance or rejection, respectively; we also assume that all computations are halting.

In order to solve decision problems (i.e., recognize languages over an alphabet Σ), we use *families* of recognizer P systems $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^*\}$. Each input x is associated with a P system Π_x in the family $\mathbf{\Pi}$ that decides the membership of x in the language $L \subseteq \Sigma^*$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for each input length [24].

These families of *recognizer* P systems can be used to solve decision problems as follows.

Definition 2. *Let Π be a P system whose alphabet contains two distinct objects *yes* and *no*, such that every computation of Π is halting and during each computation exactly one of the objects *yes*, *no* is sent out from the skin to signal acceptance or rejection. If all the computations of Π agree on the result, then Π is said to be confluent; if this is not necessarily the case, then it is said to be non-confluent and the global result is acceptance if and only if there exists at least an accepting computation.*

Definition 3. Let $L \subseteq \Sigma^*$ be a language, \mathcal{D} a class of P systems (i.e. a set of P systems using a specific subset of features) and let $\Pi = \{\Pi_x \mid x \in \Sigma^*\} \subseteq \mathcal{D}$ be a family of P systems, either confluent or non-confluent. We say that Π decides L when, for each $x \in \Sigma^*$, $x \in L$ if and only if Π_x accepts.

Complexity classes for P systems are defined by imposing a uniformity condition on Π and restricting the amount of time or space available for deciding a language.

Definition 4. Consider a language $L \subseteq \Sigma^*$, a class of recognizer P systems \mathcal{D} , and let $f: \mathbb{N} \rightarrow \mathbb{N}$ be a proper complexity function (i.e. a "reasonable" one, that can be computed by a Turing machine using a limited amount of resources. See [27, Definition 7.1], page 140). We say that L belongs to the complexity class $\mathbf{MC}_{\mathcal{D}}^*(f)$ if and only if there exists a family of confluent P systems $\Pi = \{\Pi_x \mid x \in \Sigma^*\} \subseteq \mathcal{D}$ deciding L such that:

- Π is semi-uniform, i.e. there exists a deterministic Turing machine which, for each input $x \in \Sigma^*$, constructs the P system Π_x in polynomial time with respect to $|x|$;
- Π operates in time f , i.e. for each $x \in \Sigma^*$, every computation of Π_x halts within $f(|x|)$ steps.

In particular, we say that a language $L \subseteq \Sigma^*$ belongs to the complexity class $\mathbf{PMC}_{\mathcal{D}}^*$ if and only if there exists a semi-uniform family of confluent P systems $\Pi = \{\Pi_x \mid x \in \Sigma^*\} \subseteq \mathcal{D}$ deciding L in polynomial time. Otherwise stated, $\mathbf{PMC}_{\mathcal{D}}^* = \bigcup_{k \in \mathbb{N}} \mathbf{MC}_{\mathcal{D}}^*(n^k)$.

The analogous complexity classes for non-confluent P systems are denoted by $\mathbf{NMC}_{\mathcal{D}}^*(f)$ and $\mathbf{NPMC}_{\mathcal{D}}^*$.

Another set of complexity classes is defined in terms of uniform families of recognizer P systems:

Definition 5. Consider a language $L \subseteq \Sigma^*$, a class of recognizer P systems \mathcal{D} , and let $f: \mathbb{N} \rightarrow \mathbb{N}$ be a proper complexity function. We say that L belongs to the complexity class $\mathbf{MC}_{\mathcal{D}}(f)$ if and only if there exists a family of confluent P systems $\Pi = \{\Pi_x \mid x \in \Sigma^*\} \subseteq \mathcal{D}$ deciding L such that:

- Π is uniform, i.e. for each $x \in \Sigma^*$ deciding whether $x \in L$ is performed as follows: first, a polynomial-time deterministic Turing machine, given the length $n = |x|$ as a unary integer, constructs a P system Π_n with a distinguished input membrane; then, another polynomial-time deterministic Turing machine computes an encoding of the string x as a multiset w_x , which is finally added to the input membrane of Π_n , thus obtaining a P system Π_x that accepts if and only if $x \in L$.
- Π operates in time f , i.e. for each $x \in \Sigma^*$, every computation of Π_x halts within $f(|x|)$ steps.

In particular, a language $L \subseteq \Sigma^*$ belongs to the complexity class $\mathbf{PMC}_{\mathcal{D}}$ if and only if there exists a uniform family of confluent P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^*\} \subseteq \mathcal{D}$ deciding L in polynomial time.

The analogous complexity classes for non-confluent P systems are denoted by $\mathbf{NMC}_{\mathcal{D}}(f)$ and $\mathbf{NPMC}_{\mathcal{D}}$.

As stated in the Introduction, the first definition of space complexity for P systems introduced in [30] considered a possible real implementation with biochemical materials, thus assuming that every single object and membrane requires some constant physical space. Such a definition (in the improved version from [21], taking into account also the space required by the labels for membranes and the alphabet of symbols) is the following (we stress the fact that the number of membranes in a computation step can be smaller, equal, or greater than the initial number of membranes, due to dissolution and duplication of membranes; we also stress the fact that we do not need to store unique IDs for membranes having the same label as we can, for example, indicate multisets of objects inside a string-like bracketed expression):

Definition 6. Considering a configuration \mathcal{C} of a P system Π , its size $|\mathcal{C}|$ is the number of membranes in the current membrane structure multiplied by $\log |A|$, plus the total number of objects from Γ they contain multiplied by $\log |\Gamma|$. If $\mathcal{C} = (\mathcal{C}_0, \dots, \mathcal{C}_k)$ is a computation of Π , then the space required by \mathcal{C} is defined as

$$|\mathcal{C}| = \max\{|\mathcal{C}_0|, \dots, |\mathcal{C}_k|\}.$$

The space required by Π itself is defined as the supremum of the space required by all computations of Π :

$$|\Pi| = \sup\{|\mathcal{C}| : \mathcal{C} \text{ is a computation of } \Pi\}.$$

Finally, let $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^*\}$ be a family of recognizer P systems, and let $s: \mathbb{N} \rightarrow \mathbb{N}$. We say that $\mathbf{\Pi}$ operates within space bound s if and only if $|\Pi_x| \leq s(|x|)$ for each $x \in \Sigma^*$.

Following what has been done for time complexity classes, we can define space complexity classes. By $\mathbf{MCSPACE}_{\mathcal{D}}(f(n))$ (resp. $\mathbf{MCSPACE}_{\mathcal{D}}^*(f(n))$) we denote the class of languages which can be decided by uniform (resp. semi-uniform) families, $\mathbf{\Pi}$, of confluent P systems of type \mathcal{D} , where each $\Pi_x \in \mathbf{\Pi}$ operates within space bound $f(|x|)$. In the following, we will consider P systems with active membranes using both types of division for elementary and non-elementary membranes (and we denote this by setting $\mathcal{D} = AM$), using only division for elementary membranes ($\mathcal{D} = EAM$), and not using division of membranes ($\mathcal{D} = NAM$).

In particular, the class of problems solvable in a polynomial space by uniform confluent systems is denoted by $\mathbf{PMCSpace}_{\mathcal{D}}$, and the class of problems solvable in an exponential space by uniform confluent systems is denoted by $\mathbf{EXPMCSpace}_{\mathcal{D}}$ (adding a star in case of semi-uniform classes).

The corresponding classes for non-confluent systems are $\mathbf{NPMCSPACE}_{\mathcal{D}}$ and $\mathbf{NEXPMCSPACE}_{\mathcal{D}}$.

A different approach to define space complexity for P systems was introduced in [2], considering the information stored in the objects of the systems, and not the single objects themselves. Binary notation, instead of unary, was used to store the amount of objects in each region, with the following definition of *binary space*:

Definition 7. Consider a configuration \mathcal{C} of a P system Π . Let us denote by h_1, h_2, \dots, h_z the membranes of the current membrane structure, and by $|O_{i,j}|$ the multiplicity of object i within region j . The binary size $|\mathcal{C}|_B$ of a configuration \mathcal{C} is defined as:

$$|\mathcal{C}|_B = z \cdot \log |A| + \sum_{j=1}^z \sum_{i=1}^n \left(\lceil \log(|O_{i,j}| + 1) \rceil + \log |T| \right)$$

that is the number of membranes in the current membrane structure multiplied by $\log |A|$, plus the number of bits required to store the description of the multiset in each region.

If $\mathcal{C} = (\mathcal{C}_0, \dots, \mathcal{C}_k)$ is a computation of Π , then the binary space required by \mathcal{C} is defined as

$$|\mathcal{C}|_B = \max\{|\mathcal{C}_0|_B, \dots, |\mathcal{C}_k|_B\}.$$

The binary space required by Π itself is then obtained by computing the binary space required by all computations of Π and taking the supremum:

$$|\Pi|_B = \sup\{|\mathcal{C}|_B : \mathcal{C} \text{ is a computation of } \Pi\}.$$

Finally, let $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^*\}$ be a family of recognizer P systems, and let $s: \mathbb{N} \rightarrow \mathbb{N}$. We say that $\mathbf{\Pi}$ operates within binary space bound s if and only if $|\Pi_x|_B \leq s(|x|)$ for each $x \in \Sigma^*$.

Corresponding space complexity classes, that concern this different size measure, can be introduced.

3 Considering separate feature contributions to space definition

In both cases of space definition described in the previous Section, the amount of space required by a system was a single total measure obtained, in different ways, by considering all the elements contributing to it: the total number of membranes, the total number of objects, the types of different membranes, and the types of different objects.

We introduce now a different definition of space in the framework of P systems, where each of the previous elements is considered independently, thus allowing to consider the contribution given by each element separately, and highlighting

the requirement of each of them in every solution to complex computational problems considered in the literature.

The new measure is a vector refinement of previously defined scalar measurements. Refinements of this type have been considered in many different areas to exploit the advantages they offer. For example, in the framework of distributed systems, the partial order of event defined by a vector clock [22] (a vector of N logical clocks, one per each process of the system) allows to detect concurrency issues, that cannot be detected by using linear Lamport timestamps.

Definition 8. *Consider a P system Π with active membranes, and a computation $\mathcal{C} = (C_0, \dots, C_k)$ of Π . Let us denote the set of membrane labels of Π by Λ , and the alphabet of objects by Γ . Moreover, let us denote by MaxMe the maximum number of membranes present at the same time in Π during some steps of \mathcal{C} , and by MaxOb the maximum number of objects present at the same time in Π during some steps of \mathcal{C} .*

We say that the computation \mathcal{C} of Π is bounded by $\text{Space}(me, ob, met, obt)$ if and only if $\text{MaxMe} \leq me$, $\text{MaxOb} \leq ob$, $|\Lambda| \leq met$, and $|\Gamma| \leq obt$, where me , ob , met , and obt are some positive integers.

The space required by Π itself is then obtained by computing the corresponding space required by all computations of Π and taking the supremum.

Finally, let $\mathbf{\Pi} = \{\Pi_x : x \in \Sigma^\}$ be a family of recognizer P systems, and let $s : \mathbb{N} \rightarrow \mathbb{N}$. We say that $\mathbf{\Pi}$ operates within $\text{Space}(me, ob, met, obt)$ if and only if each member Π_x of the family operates within the same space.*

Of course, complexity classes corresponding to this definition of space can be defined in a similar way as already done in the previous Section; as an example, by $\text{MCSPACE}_{\mathcal{D}}(me, ob, met, obt)$ we denote the class of problems solved by P systems with active membranes of type \mathcal{D} having features limited according to me, ob, met, obt .

Having defined the contribution of each element to the space requirements, we are ready to survey some main results present in the literature, to analyze the requirements in terms of single feature required by each proposed solution.

The first results we analyze are from [30].

Proposition 1. $\mathbf{P} \subseteq \text{MCSPACE}_{NAM}^*(O(1))$, and $\mathbf{P} \subseteq \text{MCSPACE}_{NAM}(O(1))$.

In this solution, it is proved that semi-uniform systems with active membranes that do not use membrane division can solve all problems in \mathbf{P} in constant time. In fact, all the work is done by the deterministic Turing machine used to build the system (a deterministic polynomial time uniformity condition is considered). The obtained system simply sends out a *yes* or *no* answer, in one step. Similar considerations remain valid also for the uniform case. As a consequence, in both cases we have $me = ob = met = obt = O(1)$.

Proposition 2. $\mathbf{NP} \cup \text{coNP} \subseteq \text{EXPSPACE}_{EAM}^*$.

This proposition considers the results from [43], in particular concerning the problems *SAT* and *Hamiltonian Path*.

In the first case, considering an instance with n variables and m clauses, 2^n membranes are generated, each containing one possible truth assignment to be checked. We have: $me = \mathcal{O}(2^n)$, $ob = \mathcal{O}((n * m) * (2^n))$, $met = 2$, and $obt = 4n + 2m + 4$.

The solution for the Hamiltonian Path problem tries all possible paths, to check if at least one of them satisfies the required conditions. In this case, we have $me = \mathcal{O}(n^n)$, $ob = \mathcal{O}(n * me) = \mathcal{O}(n^{n+1})$, $met = 2$, $obt = 6n + 4$.

In the paper [37], it was shown how to exploit membrane division to solve the **PSPACE**-complete problem Satisfiability of Quantified Boolean Formulas (QBF), using semi-uniform P systems with active membranes:

Proposition 3. $\mathbf{PSPACE} \subseteq \mathbf{EXSPACE}_{AM}^*$.

The solution makes use of an exponential number of membranes and objects: $me = \mathcal{O}(2^n)$, $ob = \mathcal{O}(2^n)$, $met = m + n + 2$, and $obt = 5n + m + 4$.

In [4] the same result was proved for uniform systems.

Proposition 4. $\mathbf{PSPACE} \subseteq \mathbf{EXSPACE}_{AM}$.

An analysis of resources used for this solution shows that $me = \mathcal{O}(2^{2n}) = \mathcal{O}(4^n)$, $ob = \mathcal{O}(2^{2n}) = \mathcal{O}(4^n)$, $met = \mathcal{O}(m + n)$, and $obt = \mathcal{O}(m * n)$.

In [33], systems with limited power were considered; in particular, division rules for non-elementary membranes and dissolution rules were avoided. It was proved that such systems can solve in polynomial time all problems in the complexity class **PP** (the class of languages decided by polynomial time probabilistic Turing machines with error probability strictly less than $1/2$). In fact, a solution for the **PP**-complete problem *SQRT-3SAT* was proposed.

Proposition 5. $\mathbf{PP} \subseteq \mathbf{PMC}_{AM}(-d, -n)$.

The features of that solution are the following: $me = \mathcal{O}(2^n)$, $ob = \mathcal{O}(2^n)$, $met = 3$, and $obt = \mathcal{O}(n)$.

In the paper [41], it was shown that a deterministic Turing machine working in polynomial space, with respect to the input length, can be efficiently simulated (both in terms of time and space) by a semi-uniform family of P systems with active membranes, using only communication rules.

Proposition 6. *A deterministic Turing machine M working in space $s(n)$ and time $t(n)$ can be simulated by semiuniform P systems in space $\mathcal{O}(s(n))$ and time $\mathcal{O}(t(n))$.*

The main idea to prove this result, was to store information bits by using polarizations associated to membranes, instead of objects inside them. As a consequence, the required amount of resourced needed is very low: $me = s(n) + 2$, $ob = \mathcal{O}(n)$ initially, then 1, $met = 3$, $obt = 5 + 3 * |Q|$, where Q is the set of states of the Turing machine M .

In the paper [3], similar results for uniform families of P systems with active membranes were proved, showing that a cubic slowdown and a quadratic space overheads are required:

Proposition 7. *A DTM M working in space $s(n)$ and time $t(n)$ can be simulated by uniform confluent or non-confluent P systems within polynomial bounds for space and time.*

In both cases we have the following: $me = \mathcal{O}(s(n))$, $ob = \mathcal{O}(s(n)^2)$, $met = 7$, and $obt = 3$.

We conclude this analysis by recalling a result from [32]. In this work, it was proved that recognizer P systems with active membranes that use polynomial space characterize the complexity class **PSPACE**. In particular, the result holds for both confluent and nonconfluent systems, and independently of the use of membrane division rules.

This generic result, allows to relate the number of computation steps to the maximum number of objects and the maximum number of membranes that can be obtained after those steps. In particular, let us consider a non confluent P system Π with active membranes, having a description of length m . After t steps of computation we have the following: $me = \mathcal{O}(2^{t*m+m*\log(m)})$, and $ob = \mathcal{O}(2^{t*t*m*\log(m)})$.

4 Conclusions

We introduced a definition for space complexity classes in the framework of P systems, where each element contributing to the definition of space used by the system (that is, the total number of membranes, the total number of objects, the types of different membranes, and the types of different objects) is considered independently. In this way, the contribution of each element in defining the total space required by the system to execute a computation can be highlighted independently.

We conclude the Section by pointing out that many different constructions by various authors have been presented in the literature. After presenting here some of them, we think it will be interesting to check if some of them open the possibility to propose alternate solutions where the considered parameters are different (i.e., of a different asymptotic order) with respect to those already published. As an example, would it be possible to find solutions similar to those of Proposition 3 and 4 by using a constant amount of object types or of membrane types?

Moreover, we want to stress the fact that, under the usual uniformity condition considered, a polynomial time precomputing by a deterministic Turing machine is allowed and, in this case, membrane types and object types are forced to be at most polynomial. However, different uniformity conditions can also be considered, to highlight their impact on these features. For instance, how to factor the input for sublinear space complexity classes if we would like to have more object types with respect to those allowed by the input size (maybe not all of them present since the beginning of the computation), but the uniformity condition forbids that?

It would also be useful to clarify how to proceed when the size of the problem instance is given by a few numbers like, e.g., the number of clauses and variables for **SAT**.

Acknowledgements

This work was partially supported by Università degli Studi di Milano-Bicocca, Fondo di Ateneo per la Ricerca 2019, project 2019-ATE-0454. The first author acknowledges project 20.80009.5007.22 “Intelligent information systems for solving ill-structured problems, processing knowledge and big data” by the National Agency for Research and Development.

References

1. Alhazov, A., Ishdorj, T.: Membrane operations in P systems with active membranes. In: Păun, Gh., Riscos-Núñez, A., Romero-Jiménez, A., Sancho-Caparrini, F. (eds.) Second Brainstorming Week on Membrane Computing. pp. 37–44. No. 1/2004 in RGNC Reports, Fénix Editora (2004)
2. Alhazov, A., Leporati, A., Manzoni, L., Mauri, G., Zandron, C.: Alternative space definitions for P systems with active membranes. *Journal of Membrane Computing* **3**(2), 87–96 (2021), <https://doi.org/10.1007/s41965-021-00074-2>
3. Alhazov, A., Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: Space complexity equivalence of P systems with active membranes and Turing machines. *Theoretical Computer Science* **529**, 69–81 (2014), <https://doi.org/10.1016/j.tcs.2013.11.015>
4. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundamenta Informaticae* **58**(2), 67–77 (2003), <http://iospress.metapress.com/content/99n72anvn6bkl4mm/>
5. Alhazov, A., Pan, L.: Polarizationless P systems with active membranes. *Grammars* **7**, 141–159 (2004)
6. Alhazov, A., Pan, L., Păun, Gh.: Trading polarizations for labels in P systems with active membranes. *Acta Informatica* **41**(2-3), 111–144 (2004), <https://doi.org/10.1007/s00236-004-0153-z>
7. Alhazov, A., Pérez-Jiménez, M.J.: Uniform solution to QSAT using polarizationless active membranes. In: Durand-Lose, J., Margenstern, M. (eds.) *Machines, Computations, and Universality*, 5th International Conference, MCU 2007, Lecture Notes in Computer Science, vol. 4664, pp. 122–133. Springer (2007), https://doi.org/10.1007/978-3-540-74593-8_11
8. Buño, K., Adorna, H.: Distributed computation of a kP system with active membranes for SAT using clause completion. *Journal of Membrane Computing* **2**(2), 108–120 (2020), <https://doi.org/10.1007/s41965-020-00040-4>
9. Cecilia, J., García, J., Guerrero, G., Martínez-del Amor, M., Pérez-Hurtado, I., Pérez-Jiménez, M.: Simulating a P system based efficient solution to SAT by using GPUs. *Journal of Logic and Algebraic Programming* **79**(6), 317–325 (2010)
10. García-Quismondo, M., Gutiérrez-Escudero, R., Martínez-del Amor, M.A., Orejuela-Pinedo, E., Pérez-Hurtado, I.: P-Lingua 2.0: A software framework for cell-like P systems. *International Journal of Computers, Communications & Control* **4**(3), 234–243 (2009)

11. Gazdag, Z., Kolonits, G.: A new approach for solving SAT by P systems with active membranes. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) *Membrane Computing, 13th International Conference, CMC 2012. Lecture Notes in Computer Science*, vol. 7762, pp. 195–207. Springer (2013)
12. Gazdag, Z., Kolonits, G.: A new method to simulate restricted variants of polarizationless P systems with active membranes. *Journal of Membrane Computing* **1**(4), 251–261 (2019)
13. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: On the power of dissolution in P systems with active membranes. In: Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, 6th International Workshop, WMC 2005. Lecture Notes in Computer Science*, vol. 3850, pp. 224–240. Springer (2006), <https://doi.org/10.1007/11603047>
14. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Campero, F.J.: Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics* **83**(7), 593–611 (2006), <https://doi.org/10.1080/00207160601065413>
15. Leporati, A., Ferretti, C., Mauri, G., Zandron, C.: Complexity aspects of polarizationless membrane systems. *Natural Computing* **4**(8), 703–717 (2008)
16. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Constant-space P systems with active membranes. *Fundamenta Informaticae* **134**(1–2), 111–128 (2014), <https://doi.org/10.3233/FI-2014-1094>
17. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Membrane division, oracles, and the counting hierarchy. *Fundamenta Informaticae* **138**(1–2), 97–111 (2015), <https://doi.org/10.3233/FI-2015-1201>
18. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Characterizing PSPACE with shallow non-confluent P systems. *Journal of Membrane Computing* **1**(2), 75–84 (2019), <https://doi.org/10.1007/s41965-019-00011-4>
19. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Shallow laconic P systems can count. *Journal of Membrane Computing* **2**(1), 49–58 (2020), <https://doi.org/10.1007/s41965-020-00032-4>
20. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: A Turing machine simulation by P systems without charges. *Journal of Membrane Computing* **2**(2), 71–79 (2020), <https://doi.org/10.1007/s41965-020-00031-5>
21. Leporati, A., Mauri, G., Porreca, A.E., Zandron, C.: A gap in the space hierarchy of P systems with active membranes. *Journal of Automata, Languages and Combinatorics* **19**(1–4), 173–184 (2014), http://theo.cs.ovgu.de/jalc/search/j19_i.html
22. Liskov, B., Ladin, R.: Highly-available distributed services and fault-tolerant distributed garbage collection. In: *Proceedings of the 5th Symposium on the Principles of Distributed Computing*. pp. 29–39. ACM (1986)
23. Murphy, N., Woods, D.: Active membrane systems without charges and using only symmetric elementary division characterise P. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, 8th International Workshop, WMC 2007. Lecture Notes in Computer Science*, vol. 4860, pp. 367–384 (2007), https://doi.org/10.1007/978-3-540-77312-2_23
24. Murphy, N., Woods, D.: The computational power of membrane systems under tight uniformity conditions. *Natural Computing* **10**(1), 613–632 (2011), <https://doi.org/10.1007/s11047-010-9244-7>
25. Orellana-Martín, D., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: P systems with proteins: a new frontier when membrane division disappears. *Journal of Membrane Computing* **1**(1), 29–39 (2019), <https://doi.org/10.1007/s41965-018-00003-w>

26. Pan, L., Alhazov, A., Ishdorj, T.O.: Further remarks on P systems with active membranes, separation, merging, and release rules. *Soft Computing* **9**(9), 686–690 (2005), <https://doi.org/10.1007/s00500-004-0399-y>
27. Papadimitriou, C.H.: *Computational Complexity*. Addison-Wesley (1993)
28. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics* **6**(1), 75–90 (2001)
29. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press (2010)
30. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Introducing a space complexity measure for P systems. *International Journal of Computers, Communications & Control* **4**(3), 301–310 (2009), <http://univagora.ro/jour/index.php/ijccc/article/view/2779>
31. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes: Trading time for space. *Natural Computing* **10**(1), 167–182 (2011), <https://doi.org/10.1007/s11047-010-9189-x>
32. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes working in polynomial space. *International Journal of Foundations of Computer Science* **22**(1), 65–73 (2011), <https://doi.org/10.1142/S0129054111007836>
33. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with elementary active membranes: Beyond NP and coNP. In: Gheorghe, M., Hinze, T., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, 11th International Conference, CMC 2010. Lecture Notes in Computer Science*, vol. 6501, pp. 338–347. Springer (2011), https://doi.org/10.1007/978-3-642-18123-8_26
34. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Sublinear-space P systems with active membranes. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) *Membrane Computing, 13th International Conference, CMC 2012. Lecture Notes in Computer Science*, vol. 7762, pp. 342–357. Springer (2013), https://doi.org/10.1007/978-3-642-36751-9_23
35. Porreca, A.E., Mauri, G., Zandron, C.: Complexity classes for membrane systems. *RAIRO Theoretical Informatics and Applications* **40**(2), 141–162 (2006), <https://doi.org/10.1051/ita:2006001>
36. Porreca, A.E., Mauri, G., Zandron, C.: Non-confluence in divisionless P systems with active membranes. *Theoretical Computer Science* **411**(6), 878–887 (2010), <https://doi.org/10.1016/j.tcs.2009.07.032>
37. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing* **2**(3), 287–298 (2003), <https://doi.org/10.1023/A:1025401325428>
38. Sosík, P.: P systems attacking hard problems beyond NP: a survey. *Journal of Membrane Computing* **1**(3), 198–208 (2019), <https://doi.org/10.1007/s41965-019-00017-y>
39. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Computational efficiency of minimal cooperation and distribution in polarizationless P systems with active membranes. *Fundamenta Informaticae* **153**(1–2), 147–172 (2017), <https://doi.org/10.3233/FI-2017-1535>
40. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Reaching efficiency through collaboration in membrane systems: Dissolution, polarization and cooperation. *Theoretical Computer Science* **701**, 226–234 (2017), <https://doi.org/10.1016/j.tcs.2017.04.015>
41. Valsecchi, A., Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: An efficient simulation of polynomial-space Turing machines by P systems with active membranes. In: Păun, Gh., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G.,

- Salomaa, A. (eds.) Membrane Computing, 10th International Workshop, WMC 2009, Lecture Notes in Computer Science, vol. 6501, pp. 461–478. Springer (2010), https://doi.org/10.1007/978-3-642-11467-0_31
42. Zandron, C.: Bounding the space in P systems with active membranes. *Journal of Membrane Computing* **2**(2), 137–145 (2020), <https://doi.org/10.1007/s41965-020-00039-x>
43. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-complete problems using P systems with active membranes. In: Antoniou, I., Calude, C.S., Dinneen, M.J. (eds.) *Unconventional Models of Computation, UMC'2K, Proc. Second Int. Conference*, pp. 289–301. Springer (2001), https://doi.org/10.1007/978-1-4471-0313-4_21

Solutions to SAT, Threshold-SAT and Unique-SAT in Synchronized P Systems

Bogdan Aman

Romanian Academy, Institute of Computer Science, Iași, Romania
`bogdan.aman@iit.academiaromana-is.ro`

Abstract. We consider synchronized membrane systems extended with communication and division rules, using the maximal parallelism evolution strategy together with synchronization between non-cooperating rewriting rules of length at most three. We prove that in this framework we can obtain in polynomial time solutions to the NP-complete problems SAT, Threshold-SAT and Unique-SAT.

1 Introduction

Membrane systems (also often mentioned as P systems) are parallel distributed systems able to model the behaviour and organization of living cells [21]. Graphically, a membrane system can be depicted as a structure formed of non-intersecting regions (membranes) in which multisets of objects and rules are placed. In what follows we will use synchronized P systems [9] in which the evolution is done by using several types of rules (evolution, communication and division) that are applied using maximal parallelism [21] and respecting the synchronization relation imposed for some sets of rules [9].

The maximal parallel manner of applying the rules ensures that in each computational step the multiset of rules to be used is chosen non-deterministically and cannot be contained in another multiset of applicable rules. This way of applying the rules turned out to be an essential feature when checking if a class of membrane systems is computational complete. Over the years the way of choosing the maximal applicable multiset of rules was extended with additional restrictions (e.g., synchronization among rules ensuring that a non-empty set of rules is applicable if each rule from the set of rules is applicable at least once [9]) or value-based criteria (e.g., guards used in kernel P systems [16] and adaptive P systems [8]). Several classes of membrane systems, defined for theoretical or practical purposes, appeared in several books during the last decade [5, 14, 22, 29].

The research done in membrane computing usually concerns one of the following aspects: modelling power [3, 12], computational completeness for various amounts and types of resources [4], and efficiency in solving NP complete problems (strong [7] or weak [6]) by proposing algorithms that trade time for space, namely by creating an exponential working space that is useful to generate a polynomial time solution. In this paper we show that, for synchronized P systems extended with communication and division rules, the synchronization is

powerful enough such that only by using rules of length at most three polynomial time solutions are provided for the NP-complete problems SAT, Threshold-SAT and Unique-SAT.

The paper is structured as follows. Section 2 briefly introduces some notions of the synchronized P systems. Section 3 represents the substantial and original part of our approach; we show that there exists synchronized P systems that solve, in polynomial number of steps, the NP-complete problems SAT, Threshold-SAT and Unique-SAT. The paper ends with conclusion and references.

2 Synchronized P Systems

Let $O = \{a_1, \dots, a_n\}$ be a finite alphabet. A multiset is a mapping $u : O \rightarrow \mathbb{N}$ from the set O to the set of non-negative integers \mathbb{N} . The multiset λ , defined such that for all $a \in O$ it holds that $\lambda(a) = 0$, is called the empty multiset. Using strings to represent multisets is a common practice in the membrane computing community; all the permutation of a string can be used to represent the same multiset. As an example, the multiset defined as $u(a_1) = 4$, $u(a_2) = 1$ and $u(a_3) = 1$ can be represented by any permutation of the string $a_1a_1a_1a_1a_2a_3$. The set O^* contains all multisets over O , while $O^+ = O^* - \{\lambda\}$ contains all non-empty multisets over O . For two multisets u and v their union and difference are defined for all $a \in O$ as $(u + v)(a) = u(a) + v(a)$ and $(u - v)(a) = \max\{0, u(a) - v(a)\}$, respectively. Also, the multiset inclusion is defined if for all $a \in O$ it holds that $u(a) \leq v(a)$. More details are available in [27].

Till now, the concept of synchronization was used in different ways in membrane computing:

- (i) synchronization between membranes in which the maximal parallel strategy of applying the rules is used [23];
- (ii) synchronization of computations among various membranes [1, 13];
- (iii) synchronization of rules having unknown evolution time by using additional mechanisms to control the evolution (e.g., bi-stable catalysts, promoters) [10];
- (iv) synchronization between the rules of a membrane by allowing a set of synchronized rules to be applicable if each of its rules is applicable at least once [9, 28].

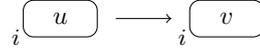
On the other hand, the maximal parallel strategy of selecting the applicable rules was controlled by using additional restrictions (e.g., prioritizing some rules) or some criteria based on existing resources (e.g., adaptive P systems [8] and kernel P systems [16] using guarded rules).

Next we define synchronized P systems [9] extended with communication and division rules.

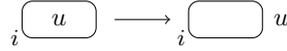
Definition 1. *A synchronized P system of degree $n + 1$ is a tuple $\Pi = (O, H, \mu, w_0, \dots, w_n, (R_0, \rho_0), \dots, (R_n, \rho_n))$, where*

- O is a finite non-empty set of objects;
- H is a finite non-empty set of labels for membranes;
- μ is a membrane structure given as a hierarchical arrangement of membranes, all of them placed in a main membrane 0, called the skin membrane, that delimits the system from its environment;
- $w_i \in O^*$, with $0 \leq i \leq n$, is a multiset of objects initially placed in the membrane i ;
- R_i , with $0 \leq i \leq n$, is a finite set of rules placed in membrane i and having rules of the following forms, where $u \in O^+$ and $v, u', v' \in O^*$:

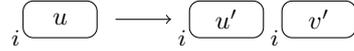
- ★ $u \rightarrow v$ (object evolution rules)
A multiset of objects u is replaced by a multiset of objects v ; the multiset v can also be the empty multiset λ . Graphically this can be depicted as:



- ★ $[u]_i \rightarrow []_i u$ (communication rules)
A multiset of objects u is sent out of membrane i and placed in the parent membrane, or in the environment if the rule is used for the skin membrane. Graphically this can be depicted as:



- ★ $[u]_i \rightarrow [u']_i [v']_i$ (division rules)
A multiset of objects u is used to divide membrane i , and is replaced in the two new instances of membrane i by the multisets of objects u' and v' ; the multisets u' and v' can also be the empty multiset λ . The other objects placed inside membrane i are duplicated and placed in the new instances of membrane i . Note that the division rules cannot be used for the skin membrane. Graphically this can be depicted as:



- ρ_i , with $0 \leq i \leq n$, is the synchronization relation over the rules of R_i and is defined as a partial relation.

Synchronization means that if there exists $r_1 \otimes \dots \otimes r_n \in \rho_i$, the rules from the set $\{r_1, \dots, r_n\}$ can be applied in a computational step only if each rule r_i , with $1 \leq i \leq n$, is used at least once.

Example 1. Consider a synchronized P system

$$\Pi = (O, \{0\}, []_0, w_0, (R_0, \rho_0)),$$

where R_0 contains the object evolution rules $r_1 : u_1 \rightarrow v_1$, $r_2 : u_2 \rightarrow v_2$ and $r_3 : u_1 \rightarrow v_3$, while $\rho_0 = \{r_1 \otimes r_2\}$ is a synchronization relation. Depending on the multiset of objects w_1 there exist different applicable multisets of rules:

- ★ if $u_1^+ \leq w_1$ and $u_1^+ u_2^+ \not\leq w_1$, then the multiset of rules r_3^+ is applicable; this is due to the fact that there does not exist the multiset u_2 for rule r_2 to be applicable, and also $r_1 \otimes r_2 \in \rho_0$ implies that the applicability of rule r_1 is conditioned by the applicability of rule r_2 ;

- ★ if $u_2^+ \leq w_1$ and $u_1^+u_2^+ \not\leq w_1$, then no rule is applicable also due to the synchronization relation ρ_0 ;
- ★ if $u_1u_2^+ \leq w_1$ and $u_1^2u_2^+ \not\leq w_1$, then the applicable multisets of rules are r_1r_2 and r_3 ;
- ★ if $u_1^2u_2^+ \leq w_1$, then the applicable multisets of rules are $r_1^+r_2$ and $r_1^+r_2r_3^+$.

3 Efficiency of Synchronized P Systems

Depending on the size of their input the NP-complete problems can be classified into two categories: *weak* (e.g., Partition, Knapsack, Subset Sum) and *strong* (e.g., SAT, Common Algorithmic Problem, Clique, Bin Packing) [15]. A survey of several solutions obtained in P systems with active membranes that also use membrane electrical charges, membrane creation and division rules is given in [25].

A solution to an NP-complete problem can be obtained if the constructed membrane system satisfies the following requirements:

- (i) a halting configuration is reached after all computations;
- (ii) in order to mark the response given by the membrane system two fresh objects *yes* and *no* are used beside those from the working alphabet;
- (iii) in the halting configuration only one of these two objects must be present in the environment: a successful computation is indicated by the availability of object *yes*, while an unsuccessful computation is indicated by the availability of object *no*.

The SAT problem is an NP-complete problem that asks about the satisfiability of a propositional logic formula $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_m$ given in conjunctive normal form, where $C_i = y_1 \vee y_2 \vee \dots \vee y_r$, for $1 \leq i \leq m$ and $1 \leq r \leq n$, and either $y_j = x_k$ or $y_j = \neg x_k$, for the set of propositional variables $X = \{x_1, x_2, \dots, x_n\}$.

The next theorem illustrates how to solve the SAT problem by means of a synchronized P systems using the maximal parallelism evolution strategy equipped with a synchronization between non-cooperating rewriting rules of length at most three. The synchronization is powerful enough to get the satisfiability answer without the need for additional parameters (cooperation, catalysts, promoters, inhibitors, etc.).

Theorem 1. *There exists a synchronized P systems that solves SAT in polynomial number of steps.*

Proof. We present the steps for constructing a synchronized P system Π , that uses the maximal parallelism evolution strategy and a synchronization between non-cooperating rewriting rules of length at most three, and that is able to provide an answer to the satisfiability of formula φ from the SAT problem. The membrane system Π is defined as

$$(O, H, \mu, w_0, w_1, (R_0, \rho_0), (R_1, \rho_1)),$$

where the components are defined as:

- $O = \{a_i, t_i, f_i \mid 1 \leq i \leq n\} \cup \{yes, yes', no, no'\}$
 $\cup \{x_{i,j} \mid x_i \in C_j, 1 \leq i \leq n, 1 \leq j \leq m\}$
 $\cup \{\bar{x}_{i,j} \mid \bar{x}_i \in C_j, 1 \leq i \leq n, 1 \leq j \leq m\}$
 $\cup \{d_i \mid 0 \leq i \leq n+m+1\} \cup \{e_i \mid 0 \leq i \leq n+m+2\}$
 $\cup \{g_i \mid 0 \leq i \leq n+m+4\} \cup \{h_i \mid 0 \leq i \leq n+m+3\};$
- $H = \{0, 1\};$
- $\mu = [[]_1]_0;$
- $w_0 = g_0 h_0$, and w_1 contains the following objects:
 - ★ $\{a_i \mid 1 \leq i \leq n\}$ - used for generating all possible truth assignments of the n variable of φ ;
 - ★ $\{x_{i,j} \mid x_i \in C_j, 1 \leq i \leq n, 1 \leq j \leq m\}$ - used to indicate that variable x_i is used in clause C_j ;
 - ★ $\{\bar{x}_{i,j} \mid \bar{x}_i \in C_j, 1 \leq i \leq n, 1 \leq j \leq m\}$ - used to indicate that variable \bar{x}_i is used in clause C_j ;
 - ★ d_0, e_0 - counters.

The set of rules R_1 and the synchronization relation ρ_1 are as follows:

- (i) $[a_i]_1 \rightarrow [t_i]_1 [f_i]_1$, for $1 \leq i \leq n$
 These rules create all the possible 2^n truth assignments over the propositional variables x_1, \dots, x_n .
- (ii) $d_i \rightarrow d_{i+1}$, for $0 \leq i \leq n+m$
 $e_i \rightarrow e_{i+1}$, for $0 \leq i \leq n+m+1$
 These rules increment the counter objects of membrane 1 and are used at certain times during evolution to generate yes' or no' objects for all 2^n assignments.
- (iii) $t_i \rightarrow t_i c_{i,j} \otimes x_{i,j} \rightarrow c_j$
 $f_i \rightarrow f_i c_{i,j} \otimes \bar{x}_{i,j} \rightarrow c_j$, for $1 \leq i \leq n, 1 \leq j \leq m$
 These rules perform the satisfiability check of all m conjunctions.
- (iv) $c_1 \rightarrow \lambda \otimes \dots \otimes c_m \rightarrow \lambda \otimes d_{n+m+1} \rightarrow yes'$
 $e_{n+m+2} \rightarrow \lambda \otimes d_{n+m+1} \rightarrow no'$
 These rules perform the satisfiability check of the formula φ . If the formula is satisfied then the object yes' is created; otherwise the object no' is created.
- (v) $[yes']_1 \rightarrow []_1 yes'$
 This rule is used to send, if exists, the yes' objects out of membrane 1.

The set of rules R_0 and the synchronization relation ρ_0 are as follows:

- (vi) $g_i \rightarrow g_{i+1}$, for $0 \leq i \leq n+m+3$
 $h_i \rightarrow h_{i+1}$, for $0 \leq i \leq n+m+2$
 These rules increment the counter objects of membrane 0 and are used at certain times during evolution to generate the final unique yes or no object.
- (vii) $yes' \rightarrow \lambda \otimes h_{n+m+3} \rightarrow yes$
 $no' \rightarrow \lambda \otimes g_{n+m+4} \rightarrow no$
 These rules are used to generate in membrane 0 the unique yes or no answer.

- (viii) $[yes]_0 \rightarrow []_0 \text{ yes}$
 $[no]_0 \rightarrow []_0 \text{ no}$

These rules are used to send out of membrane 0 and into environment the unique *yes* or *no* answer. The computation stops after performing these rules.

Overview of the computation. We detail how the synchronized P systems perform the satisfiability check of the formula φ . The initial configuration is $[w_0[w_1]_1]_0$.

The generation phase happens in n steps. The purpose of the rules (i) is to generate all possible truth assignments over the propositional variables $\{x_1, \dots, x_n\}$. In parallel the rules (ii) are used to increase the counters d and e , while the rules (vi) are used to increase the counters g and h . Thus after n steps inside each membrane 1 are placed the counters d_n and e_n , while in membrane 0 are placed the counters g_n and h_n .

The checking phase happens in $m + 2$ steps. The purpose of the rules (iii) is to check which of the clauses in each assignment are satisfied. If all m clauses c_j , with $1 \leq j \leq m$, are satisfied by the current truth assignment placed inside a membrane 1, then the first of the rules (iv) generates an object *yes'* while consuming the counter d_{n+m+1} . Otherwise if one of the clauses is not satisfied the corresponding c_j object is not created and thus the first of the rules (iv) is not applicable. This means that in the next step, the objects e_{n+m+2} and d_{n+m+1} can be consumed by the second of the rules (iv) in order to generate an object *no'*. Also, in parallel the counters g and h are incremented by the rules (vi). This phase ends with the rules (v) that are used to send the *yes'* objects out of the membranes labelled by 1.

After performing the generation and checking phases, the rules (vii) can be applied in membrane 1 in order to create the unique answer to the problem: a *yes* or *no* object. Depending on how many *yes'* objects were created after the checking phase, there are two possibilities: (1) if there exist *yes'* objects, then the answer is generated by consuming all *yes'* objects and the h_{n+m+3} object, while creating a unique object *yes*; (2) if there are no *yes'* objects then only the counter g is incremented and then used to generate a unique *no* object. The computation stops after using the rules (viii) to send into the environment the unique object *yes* or *no* obtained in the previous step.

Thus, in the worst case scenario, it takes $n + m + 6$ steps to generate the unique *yes* or *no* answer and send it to the environment.

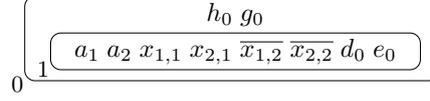
Example 2. We illustrate how the constructed synchronized P system looks and evolves, by considering a simple example from [20]. Consider a propositional formula with two variables and two clauses:

$$\varphi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$$

Note that the formula φ is satisfiable by two assignments:

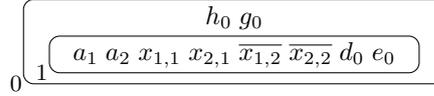
$$x_1 = \text{true}, x_2 = \text{false} \text{ and } x_1 = \text{false}, x_2 = \text{true}.$$

Thus $n = m = 2$ and the initial configuration can be depicted as:

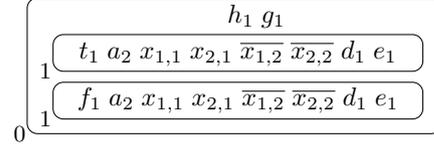


The computation proceeds as follows where we depict the configuration after each step:

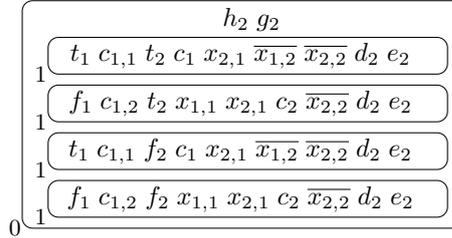
Step 0. The initial configuration is:



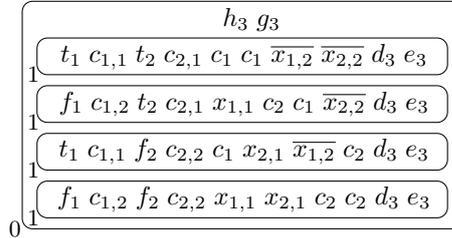
Step 1. In the first step the process of division is started. After applying rule (i) on object a_1 we obtain the configuration:



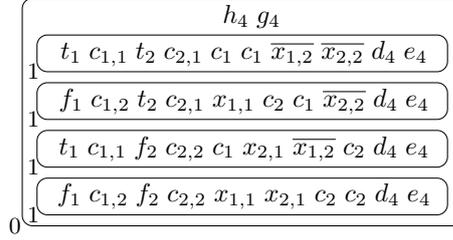
Step 2. After $n = 2$ steps all possible $2^n = 4$ assignments are created. In parallel, we start the checking phase and see which clauses are satisfied in each membrane 1 for truth values t_1 and f_1 , respectively.



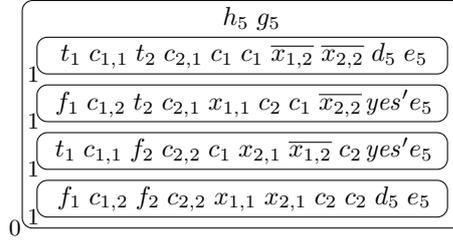
Step 3. In this step we continue the checking phase and see which clauses are satisfied in each membrane 1 for truth values t_2 and f_2 , respectively.



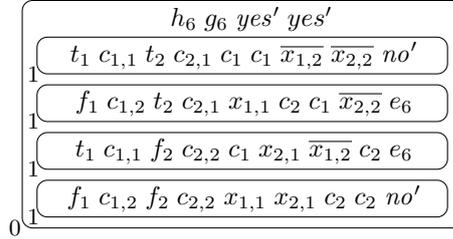
Step 4. As each variable appears only once in all clauses, in this step only the counters are incremented.



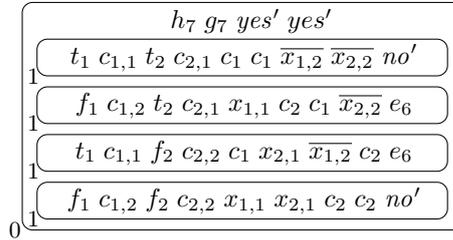
Step 5. If there are assignments that satisfy φ , then *yes'* objects are generated for those assignments.



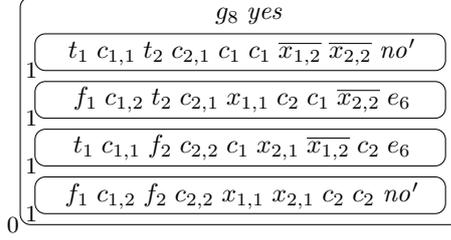
Step 6. If there are assignments that do not satisfy φ , then *no'* objects are generated for those assignments, while at the same time the *yes'* objects from membranes 1 are communicated to membrane 0.



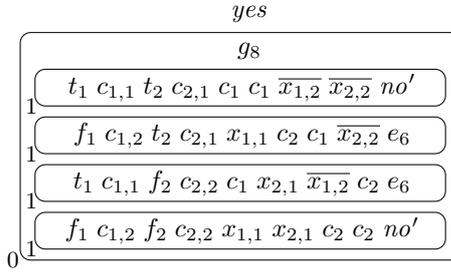
Step 7. In this step only the counters f and g are incremented.



Step 8. As there exist *yes'* objects in membrane 0, then the unique *yes* answer is generated inside membrane 0 after $n + m + 4 = 8$ steps.



Step 9. As there exist an unique *yes* object in membrane 0, then it will be sent to the environment and the computation stops after $n + m + 5 = 9$ steps.



In [26] the decision problem called Threshold-SAT is considered: given a Boolean formula φ with n variables and m clauses, exist more than k assignments, with $0 < k < 2^m$, satisfying it?

The next result illustrates how to solve the Threshold-SAT problem by means of a synchronized P systems using the maximal parallelism evolution strategy equipped with a synchronization between non-cooperating rewriting rules of length at most three. The synchronization allows to obtain the satisfiability answer without the need for additional parameters (cooperation, catalysts, promoters, inhibitors, etc.).

Corollary 1. *There exists a synchronized P systems that solves Threshold-SAT in polynomial number of steps.*

Proof. The proof proceeds in a similar manner as for Theorem 1, except that for membrane 0 the set of rules and the synchronization relation differ. The set of rules R_0 and the synchronization relation ρ_0 used in the current proof are as follows:

$$(vii') \quad yes' \rightarrow \lambda \otimes \dots (k \text{ times}) \dots \otimes yes' \rightarrow \lambda \otimes h_{n+m+3} \rightarrow yes$$

$$h_{n+m+3} \rightarrow \lambda \otimes g_{n+m+4} \rightarrow no$$

These rules are used to generate in membrane 0 the unique *yes* or *no* answer.

$$(viii) \quad [yes]_0 \rightarrow []_0 yes$$

$$[no]_0 \rightarrow []_0 no$$

These rules are used to send out of membrane 0 and into environment the unique *yes* or *no* answer. The computation stops after performing these rules.

Overview of the computation. We detail how the synchronized P systems perform the satisfiability check of the formula φ . The generation and checking phases are done in $n + m + 2$ steps as in the proof of Theorem 1.

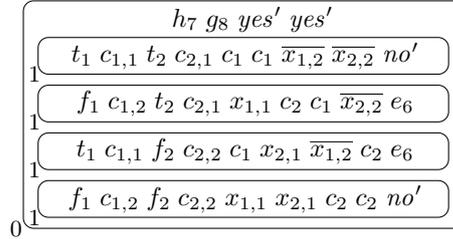
After performing the generation and checking phases, the rules (vii') can be applied in membrane 1 in order to create the unique answer to the problem: a *yes* or *no* object. Depending on how many *yes'* objects were created after the checking phase, there are two possibilities: (1) if there exist at least k *yes'* objects, then the answer is generated by consuming all *yes'* objects and the h_{n+m+3} object, while creating an unique object *yes*; (2) if there are less than k *yes'* objects then only the counter g is incremented and then used to generate an unique *no* object. The computation stops after using the rules $(viii)$ to send into the environment the unique object *yes* or *no* obtained in the previous step.

Thus, in the worst case scenario, it takes $n + m + 6$ steps to generate the unique *yes* or *no* answer and send it to the environment.

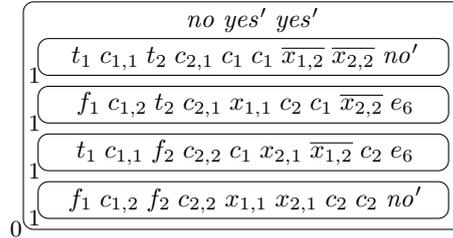
Example 3. Consider the propositional formula φ from Example 2. Depending on the value of the threshold we have two possible evolutions:

1. If $0 < k \leq 2$. Then the system behaves in a similar manner as in Example 2, except that in the last step are applied the rules (vii') instead of the rules (vii) . Note that for $k = 1$ the SAT problem is in fact an instance of the Threshold-SAT problem.
2. If $2 < k \leq 2^n$.

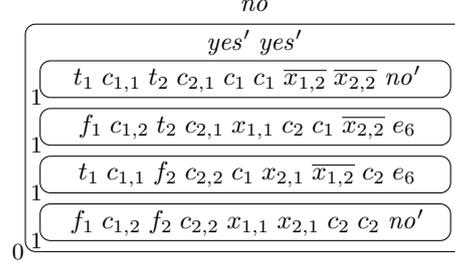
Step 8'. As there exist *yes'* objects in membrane 0, but their number is less than k then the unique *yes* answer cannot be generated, so only the counter g is incremented in this step.



Step 9'. As the objects h_7 and g_8 exist in membrane 0, then the unique *no* answer is generated after $n + m + 5 = 9$ steps.



Step 10'. As there exist an unique *no* object in membrane 0, then it will be sent to the environment and the computation stops after $n + m + 6 = 10$ steps.



In [18] the decision problem called Unique-SAT is considered: given a Boolean formula φ with n variables and m clauses, exactly one assignment (out of 2^n) satisfies it?

The next result illustrates how to solve the Unique-SAT problem by means of a synchronized P systems working in the maximal parallelism evolution strategy equipped with a synchronization between non-cooperating rewriting rules of length at most three. The synchronization allows to obtain the satisfiability answer without the need for additional parameters (cooperation, catalysts, promoters, inhibitors, etc.).

Corollary 2. *There exists a synchronized P systems that solves Unique-SAT in polynomial number of steps.*

Proof. The proof proceeds in a similar manner as for Theorem 1, except for the set of rules and the synchronization relation of membrane 0. The set of rules R_0 and the synchronization relation ρ_0 used in the current proof are as follows:

- (vi'') $g_i \rightarrow g_{i+1}$, for $0 \leq i \leq n + m + 3$
 $h_i \rightarrow h_{i+1}$, for $0 \leq i \leq n + m + 2$
 $l_i \rightarrow l_{i+1}$, for $0 \leq i \leq n + m + 4$

These rules increment the counter objects of membrane 0 and are used at certain times during evolution to generate the unique *yes* or *no* object. Note that a new counter l is needed when an exact number of solutions is needed.

- (vii'') $\text{yes}' \rightarrow \lambda \otimes \text{yes}' \rightarrow \lambda \otimes h_{n+m+3} \rightarrow \text{no}$
 $\text{yes}' \rightarrow \lambda \otimes g_{n+m+4} \rightarrow \text{yes}$
 $l_{n+m+5} \rightarrow \lambda \otimes h_{n+m+3} \rightarrow \lambda \otimes g_{n+m+4} \rightarrow \text{no}$

These rules are used to generate in membrane 0 the unique *yes* or *no* answer.

- (viii) $[\text{yes}]_0 \rightarrow []_0 \text{yes}$
 $[\text{no}]_0 \rightarrow []_0 \text{no}$

These rules are used to send out of membrane 0 and into environment the unique *yes* or *no* answer. The computation stops after performing these rules.

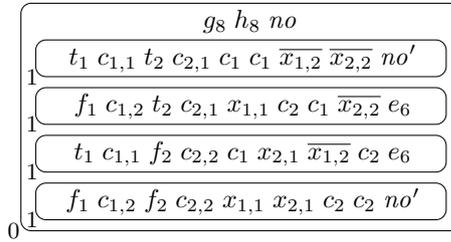
Overview of the computation. We detail how the synchronized P systems perform the satisfiability check of the formula φ . The generation and checking phases are performed in $n + m + 2$ steps in a similar manner as in the proof of Theorem 1. Also, in parallel the counter l is incremented.

After performing the generation and checking phases, the rules (vii'') can be applied in membrane 1 in order to create the unique answer to the problem: a *yes* or *no* object. Depending on how many *yes'* objects were created after the checking phase there are three possibilities: (1) if there exist at least two *yes'* objects, then the answer is generated by consuming all *yes'* objects and the h_{n+m+3} object, while creating an unique object *no*; (2) if there exists an unique *yes'* object, then only the counter g is incremented and then is used to generate an unique *yes* object; (3) if there exists no *yes'* objects, then only the counters g and l are incremented and used to generate a unique *no* answer. The computation stops after using the rules $(viii)$ to send into the environment the unique object *yes* or *no* obtained in the previous step.

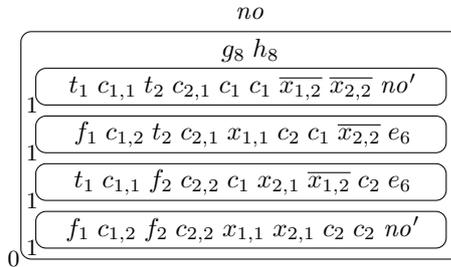
Thus, in the worst case scenario, it takes $n + m + 7$ steps to generate the unique *yes* or *no* answer and send it to the environment.

Example 4. Consider the propositional formula φ from Example 2. Then the system behaves in the first seven steps in a similar manner as in Example 2, except that also the counter l is indexed alongside counters g and h in membrane 0.

Step 8". As there exist more than one *yes'* objects in the previous step, then the unique *no* answer is generated. Note that in this step is applied the first of the rules (vii'') .



Step 9". In this step the counter l is incremented, the unique object *no* is sent to the environment and the computation stops after $n + m + 5 = 9$ steps.



4 Conclusion

We showed that synchronized P systems, extended with communication and division rules, can solve the SAT, Threshold-SAT and Unique-SAT NP-complete problems in polynomial time by using synchronization between non-cooperating rewriting rules of length at most three. This represents an improvement regarding the number and types of resources, because over the years it was shown that beside division and communication rules the SAT problem can be solved by using also: (i) membrane polarizations [11, 20, 24]; (ii) label changing [2]. Other results avoid membrane division rules by using instead: (i) membrane creation rules [17]; (ii) membrane separation rules [19]

References

1. Alhazov, A., Margenstern, M., Verlan, S.: Fast synchronization in P systems. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing - 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers*. Lecture Notes in Computer Science, vol. 5391, pp. 118–128. Springer (2008). https://doi.org/10.1007/978-3-540-95885-7_9
2. Alhazov, A., Pan, L., Păun, G.: Trading polarizations for labels in P systems with active membranes. *Acta Informatica* **41**(2-3), 111–144 (2004). <https://doi.org/10.1007/s00236-004-0153-z>
3. Aman, B., Ciobanu, G.: Describing the immune system using enhanced mobile membranes. *Electron. Notes Theor. Comput. Sci.* **194**(3), 5–18 (2008). <https://doi.org/10.1016/j.entcs.2007.12.003>
4. Aman, B., Ciobanu, G.: Turing completeness using three mobile membranes. In: Calude, C.S., Costa, J.F., Dershowitz, N., Freire, E., Rozenberg, G. (eds.) *Unconventional Computation, 8th International Conference, UC 2009, Ponta Delgada, Azores, Portugal, September 7-11, 2009. Proceedings*. Lecture Notes in Computer Science, vol. 5715, pp. 42–55. Springer (2009). https://doi.org/10.1007/978-3-642-03745-0_12
5. Aman, B., Ciobanu, G.: *Mobility in Process Calculi and Natural Computing*. Natural Computing Series, Springer (2011). <https://doi.org/10.1007/978-3-642-24867-2>
6. Aman, B., Ciobanu, G.: Solving a weak NP-complete problem in polynomial time by using mutual mobile membrane systems. *Acta Informatica* **48**(7-8), 409–415 (2011). <https://doi.org/10.1007/s00236-011-0144-9>
7. Aman, B., Ciobanu, G.: Efficiently solving the bin packing problem through bio-inspired mobility. *Acta Informatica* **54**(4), 435–445 (2017). <https://doi.org/10.1007/s00236-016-0264-3>
8. Aman, B., Ciobanu, G.: Adaptive P systems. In: Hinze, T., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) *Membrane Computing - 19th International Conference, CMC 2018, Dresden, Germany, September 4-7, 2018, Revised Selected Papers*. Lecture Notes in Computer Science, vol. 11399, pp. 57–72. Springer (2018). https://doi.org/10.1007/978-3-030-12797-8_5
9. Aman, B., Ciobanu, G.: Synchronization of rules in membrane computing. *J. Membr. Comput.* **1**(4), 233–240 (2019). <https://doi.org/10.1007/s41965-019-00022-1>

10. Cavaliere, M., Sburlan, D.: Time and synchronization in membrane systems. *Fundam. Informaticae* **64**(1-4), 65–77 (2005), <http://content.iospress.com/articles/fundamenta-informaticae/fi64-1-4-07>
11. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulating a P system based efficient solution to SAT by using GPUs. *J. Log. Algebraic Methods Program.* **79**(6), 317–325 (2010). <https://doi.org/10.1016/j.jlap.2010.03.008>
12. Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (eds.): *Applications of Membrane Computing*. Natural Computing Series, Springer (2006). <https://doi.org/10.1007/3-540-29937-8>
13. Dinneen, M.J., Kim, Y., Nicolescu, R.: Faster synchronization in P systems. *Nat. Comput.* **11**(1), 107–115 (2012). <https://doi.org/10.1007/s11047-011-9271-z>
14. Frisco, P., Gheorghe, M., Pérez-Jiménez, M.J. (eds.): *Applications of Membrane Computing in Systems and Synthetic Biology*. Emergence, Complexity and Computation, Springer (2014). <https://doi.org/10.1007/978-3-319-03191-0>
15. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman (1979)
16. Gheorghe, M., Ipate, F.: A kernel P systems survey. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing - 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers*. Lecture Notes in Computer Science, vol. 8340, pp. 1–9. Springer (2013). https://doi.org/10.1007/978-3-642-54239-8_1
17. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A uniform solution to SAT using membrane creation. *Theor. Comput. Sci.* **371**(1-2), 54–61 (2007). <https://doi.org/10.1016/j.tcs.2006.10.013>
18. Orellana-Martín, D., Valencia-Cabrera, L., Riscos-Nez, A., Prez-Jimenez, M.: The unique satisfiability problem from a membrane computing perspective. *Romanian Journal of Information Science and Technology* **21**, 288–297 (2018)
19. Pan, L., Alhazov, A.: Solving HPP and SAT by P systems with active membranes and separation rules. *Acta Informatica* **43**(2), 131–145 (2006). <https://doi.org/10.1007/s00236-006-0018-8>
20. Păun, G.: P systems with active membranes: Attacking NP-complete problems. *J. Autom. Lang. Comb.* **6**(1), 75–90 (2001). <https://doi.org/10.25596/jalc-2001-075>
21. Păun, G.: *Membrane Computing: An Introduction*. Natural computing series, Springer (2002). <https://doi.org/10.1007/978-3-642-56196-2>
22. Păun, G., Rozenberg, G., Salomaa, A.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., USA (2010)
23. Păun, G., Yu, S.: On synchronization in P systems. *Fundam. Informaticae* **38**(4), 397–410 (1999). <https://doi.org/10.3233/FI-1999-38404>
24. Pérez-Jiménez, M.J., Jiménez, Á.R., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. *Nat. Comput.* **2**(3), 265–285 (2003). <https://doi.org/10.1023/A:1025449224520>
25. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, Á., Woods, D.: Complexity: membrane division, membrane creation. *The Oxford Handbook of Membrane Computing* pp. 302–336 (2010)
26. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Elementary active membranes have the power of counting. *Int. J. Nat. Comput. Res.* **2**(3), 35–48 (2011). <https://doi.org/10.4018/jncr.2011070104>
27. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages*, 3 Volumes. Springer (1997)

28. Song, B., Pan, L.: Rule synchronization for tissue P systems. *Information and Computation* p. 104685 (2021). <https://doi.org/https://doi.org/10.1016/j.ic.2020.104685>
29. Zhang, G., Pérez-Jiménez, M.J., Gheorghe, M. (eds.): *Real-life Applications with Membrane Computing. Emergence, Complexity and Computation*, Springer (2017). <https://doi.org/10.1007/978-3-319-55989-6>

Matrix Representation and Simulations of Numerical Spiking Neural P Systems

Korsie J. Ballesteros, Dionne Peter P. Cailipan, Ren Tristan De La Cruz, Francis George C. Cabarle, and Henry N. Adorna

Algorithms and Complexity, Dept. of Computer Science, University of the Philippines Diliman, 1101 Quezon City, Philippines

Abstract. Spiking Neural P systems (SNP systems) are biologically inspired models of computation based on the firing behavior of neurons. Variations of these systems have been proposed to solve more specific problems. A more recent variation called the Numerical Spiking Neural SNP systems (NSNP systems) combines concepts from SNP systems and Numerical P systems to create a new model of computation. This model allows continuous production functions and in effect, allows for faster production. In this work, we propose a matrix representation and a corresponding simulation algorithm for NSNP systems. Having a matrix representation and a simulation algorithm allows for testing of solutions *in silico*. We also present an NSNP system that solves the subset sum problem, and use the matrix representation and simulation algorithm to obtain the solution.

Keywords: Spiking Neural P Systems, Numerical P Systems, Matrix Representation, Simulation

1 Introduction

Computers have evolved from being simple machines that can solve small problems to complex systems that make our society today more productive than ever. Even with the great strides of improvements of algorithms and hardware throughout the years, a large class of problems still remains difficult for traditional computers to solve. It is of great interest to find new ways to solve problems. In this pursuit the field of Natural Computing was developed. Some sub-fields include DNA computing, Quantum Computing and Membrane Computing introduced in [10]. These sub-fields of Natural Computing all investigate computational models and techniques derived from various natural processes. Membrane Computing in particular introduced the concept of SNP systems which are models of computation which are based on the structure of biological cells wherein computations represent the interaction of chemical across cell membranes. Much like its ancestors various sub-fields of Membrane Computing emerged one of which was the Spiking Neural P (SNP) systems introduced in [7]. SNP systems are a model of computation heavily influenced by the biological

neurons as it mimics the behavior of neurons which send electrical impulses (spikes) along axons to other neurons. Introduced in [11] is another class of P systems called Numerical P system (N P system) wherein numerical variables evolve by means of polynomial production functions and repartition protocols. It focuses on a hierarchical structure and is deterministic in nature.

Even though SNP systems have already been used to solve NP-complete problems (such as in [9]) and were used for image processing as in [13], optimization of such systems is important so that we can use it to solve more complex problems. Development of simulation algorithms for these systems allow us to further investigate the potential and limitations of these systems. To aid with the development of simulation algorithms matrix representation for these systems was first introduced in [16]. Matrix representations represented configurations and related information of given SNP systems so that transition from one state to another can be simplified as a sequence of matrix operations. These matrix representations will then be used by the simulation algorithms which focuses on creating the computation tree to account for the nondeterminism in these systems whether be it a CPU or GPU algorithm. A few works on simulation algorithms for some variants of SNP systems include [4], [6], [8], and [5]. In order to optimize the runtime of previously mentioned work [1] and [3] presented ideas and proof of concepts on how to use GPUs to improve runtime of these simulators.

While SNP systems have been proven to be equivalent to Turing Machines and there are works in applications of SNP systems some changes can be made to how it works so that it can potentially solve a larger instances of some problems.

To deal with some of the limitations posed by the previously mentioned P system, a new class of SNP systems was introduced called Numerical Spiking Neural P (NSNP) systems. NSNP systems combine some features of SNP systems and Numerical P systems. NSNP systems follow the network style architecture of SNP systems and use variables rather than the singleton alphabet used by SNP systems. It uses numerical variables to encode information and continuous production functions to process information [14]. Proved to be equivalent to Turing Machines, NSNP systems can also have non-deterministic computations by multiple production functions in one neuron. Execution of production functions are not controlled by virtue of regular expressions which alleviates the problem of solving an NP-Complete problem to check the applicability of rules. Its numerical nature paired with the threshold control strategy for neurons with multiple rules make it so that it is an interesting choice to solve real world applications that require quantitative modeling and a deterministic mechanism [14]. Lastly because of the continuous nature of the production functions, developing learning algorithms can be easier when compared to the discrete integrate-and-fire behavior of neurons in SNP systems [14].

In this work, a matrix representation for NSNP systems is introduced together with a simulation algorithm so that the advantages presented by the [14] can be verified through simulations of NSNP systems that are equivalent to those currently in literature. The study also has the same constraints as [14] thus we

only consider NSNP systems without delay and without constant values for the production functions. Lastly, a NSNP solution to Subset sum was introduced and use the matrix representation and simulation algorithm to obtain the solution.

2 Background

2.1 Spiking Neural P Systems (SNP systems)

An SNP system Π is a construct of form $\Pi = (O, \sigma_1, \sigma_2, \dots, syn, in, out)$ where:

1. $O = \{a\}$ is a singleton alphabet containing a single symbol (spike).
2. $\sigma_1, \sigma_2, \dots$ are the neurons with the form of $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, where
 - (a) $n_i \geq 0$ is the initial number of spikes in σ_i
 - (b) R_i is a finite set of rules of two forms:
 - i. **Spiking Rule** $E/a^c \rightarrow a^p; d$ where E is a regular expression over O and $c \geq p \geq 0, d \geq 0$
 - ii. **Forgetting Rule** $a^s \rightarrow \lambda$, for $s \geq 1$ with for each spiking rule in R_i $a^s \notin L(E)$;
3. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $i \neq j$ for all $(i, j) \in syn, 1 \leq i, j \leq m$
4. $in, out \in \{1, 2, \dots, m\}$ are the input and output neurons respectively if any.

Consider the spiking rules, if a neuron σ_i has a spiking rule in R_i and contains k spikes, where $k \geq c$ and $a^k \in L(E)$ this means that $E/a^c \rightarrow a^p; d$ can be applied. Remove c spikes from neuron σ_i resulting in the number of spikes in σ_i after application of a rule be equal to $k - c$. If $d = 0$ spikes are fired instantly otherwise spikes are fired after step $t + d$ where t is the current time step in the computation. Between steps t and $t + d$ the neuron will not fire the spikes to connected neurons and is *closed*, this means that σ_i cannot accept spikes from other neurons connected to it consequently rules cannot be applied during this period. At step $t + d + 1$ spikes are fired and σ_i is now open meaning that rules can be applied and that it can accept spikes from other neurons. Now consider forgetting rules, if a neuron σ_i has a forgetting rule in R_i and contains exactly s spikes, then the rule $a^s \rightarrow \lambda$ can be applied. All s spikes are removed from σ_i . Lastly, spiking rules where $E = a^c$ can be written as $a^c \rightarrow a^p; d$.

If at a given step in the computation, a neuron has 2 rules that can be applied, only one rule can be applied and is non-deterministically chosen. The *configuration* of an SNP system Π at step t is denoted as $C_t = \langle r_1/k_1, \dots, r_m/k_m \rangle$, where $1 \leq i \leq m$ and σ_i contains r_i spikes and remains closed for k_i more steps. A computation of an SNP system is a finite or infinite sequence of configurations. A computation halts if it reaches a configurations wherein no more rules can be applied. Outputs of SNP systems are commonly obtained by measuring the time interval between the first two spikes that the output neuron σ_{out} sent to the environment.

2.2 Numerical Spiking Neural P Systems

We use the conventions presented in [14] to define Numerical Spiking Neural P systems (NSNP systems).

An NSNP system containing $m \geq 1$ neurons is represented by a tuple Π where:

$$\Pi = (\sigma_1, \sigma_2, \dots, \sigma_m, \text{syn}, \text{in}, \text{out})$$

where:

1. $\sigma_1, \sigma_2, \dots, \sigma_m$ are the neurons with the form of

$$\sigma_i = (\text{Var}_i, \text{Prf}_i, \text{Var}_i(0)), 1 \leq i \leq m$$

where each element of a σ_i is defined by the following:

- (a) $\text{Var}_i = \{x_{q,i} | 1 \leq q \leq k_i\}$ refers to the variables in neuron σ_i
- (b) $\text{Var}_i(0) = \{x_{q,i}(0) | x_{q,i}(0) \in \mathbb{R}, 1 \leq q \leq k_i\}$ refers to the initial values assigned to the corresponding variables $x_{q,i}$ from Var_i and k_i represents the number of production functions inside σ_i
- (c) Prf_i is the set of production functions inside σ_i and are of the two following forms.
 - i. *Nonthreshold Form*: $f(x_{1,i}, \dots, x_{k_i,i})$
 - ii. *Threshold Form*: $f(x_{1,i}, \dots, x_{k_i,i}) |_{T_L}$ where $T_L \in \mathbb{Z}$ indicates the threshold of the corresponding production function
- (d) $\text{syn} \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ refers to the set synapses, for each $(i, j) \in \text{syn}, 1 \leq i, j \leq m$, and $i \neq j$
- (e) $\text{in}, \text{out} \in \{1, 2, \dots, m\}$ are the input and output neurons respectively if any.

The production functions in a given NSNP system are applied. The function could be any mathematical function using the variables present inside it but for this application we will only be using polynomials specifically with degree 1 because it already is equivalent to Turing Machines even with this constraint as shown in [14]. Steps in the application of production function could be separated into 2 stages called the production stage and distribution stage with the following behavior:

1. *Production Stage*: Computes or evaluates the value of the production function using $x_{1,i}(t), \dots, x_{k_i,i}(t)$ or simply the current values in the variables present in the neuron where the production function resides. Call this value $prv_{l,i}(t) = f_{l,i}(x_{1,i}(t), \dots, x_{k_i,i}(t))$.
2. *Distribution Stage*: The computed production value $prv_{l,i}$ from the production stage is transmitted to the post synaptic neurons σ_j such that $(i, j) \in \text{syn}$

Consider a variable $x_{q,i}$ included in the computation of $prv_{l,i}$, after computing the production value $prv_{l,i}$ the value inside $x_{q,i}$ will be reset to 0 but if it is a presynaptic neuron it will receive values from relevant production values. If it will receive multiple production values at the same time all these values will be added to $x_{q,i}$ during the distribution stage.

3 Matrix Representation of Numerical Spiking Neural P Systems

Matrix representations for SNP systems without delays was first introduced in [16] which was recently revisited in [2]. We only consider the case with no delays as these were proven to be already equivalent to Turing Machines. We use the example at Figure 1 to illustrate the matrix representations. Consider an NSNP system Π with m variables, g neurons and n production functions. The following definitions is the proposed possible matrix representation for NSNP systems.

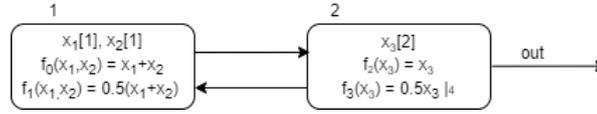


Fig. 1. Sample NSNP system

Definition 1: The **Configuration Vector** $C^{(k)} = \langle c_1^{(k)}, \dots, c_m^{(k)} \rangle$, where $c_i^{(k)}$ is the value of variable x_i at time step k . Similarly a **Configuration Matrix** $CM^{(k)}$ is a matrix containing rows of configuration vectors. Consider a Configuration Matrix with q Configuration Vectors,

$$CM^{(k)} = \begin{bmatrix} c_{ij}^{(k)} \end{bmatrix}_{q \times m} \quad (1)$$

where $c_{ij}^{(k)}$ is the value of the variable x_j on instance i at time step k and an "instance" is one of the possible resulting configurations for time step k obtained from the previous configuration at time step $k - 1$ where $k > 0$.

Consider the SNP system in Figure 1, the initial Configuration Matrix is

$$C^0 = [1 \ 1 \ 2]$$

The initial Configuration Matrix of any NSNP system will only have 1 row since we are only dealing with 1 configuration, rows of this matrix can increase for the succeeding computation steps.

Definition 2: The **Function Matrix** F is the matrix that represents the production functions in the system. Let b_i be the coefficient of the variable x_i in the production functions. The i th row of F is the vector representation for the production function f_i .

$$F = [f_{ij}]_{n \times m} \quad (2)$$

where n is the number of production functions in the systems and v is the number of variables.

$$f_{ij} = \begin{cases} b_j; & \text{if variable } x_j \text{ is present in } f_i \\ 0; & \text{otherwise} \end{cases} \quad (3)$$

For the example in Figure 1 we have the Function Matrix as follows:

$$F = \begin{bmatrix} 1 & 1 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0.5 \end{bmatrix}$$

The first row represents the function $f(x_1, x_2) = (1)x_1 + (1)x_2$ while the second row represents the function $f(x_1, x_2) = (0.5)x_1 + (0.5)x_2$

Definition 3: The **Function Location Matrix** L is the matrix that indicates in what neuron a given production function is located.

$$L = [l_{ij}]_{n \times g} \quad (4)$$

$$l_{ij} = \begin{cases} 1; & \text{if production function } f_i \text{ is present in } \sigma_j \\ 0; & \text{otherwise} \end{cases} \quad (5)$$

For the example in Figure 1 we have the Function Location Matrix as follows:

$$L = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Definition 4: The **Spiking Matrix** S of an NSNP system Π is a collection of Spiking Vectors, where a Spiking Vector is a valid combination of production functions that activates during a given time step

$$S^{(k)} = [s_{ij}^{(k)}]_{q \times n} \quad (6)$$

Where:

$$s_{ij}^{(k)} = \begin{cases} 1; & \text{if there is no threshold or the threshold is satisfied for} \\ & \text{function } f_j \text{ and is applied in spiking vector } i \\ 0; & \text{otherwise} \end{cases} \quad (7)$$

and any $S^{(k)}$ is a matrix where q is the number of valid spiking vectors for a given $C^{(k)}$ and n is the number of production functions.

For the example in Figure 1, we have the Spiking Matrix as follows:

$$S^0 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Rows represent the 2 following configurations that can result from $C^{(k)}$ since both rules in neuron 1 can be fired during the first time step. The first row $\langle 1, 0, 1, 0 \rangle$ represent the activation of the first production function in both of the neurons while the second row $\langle 0, 1, 1, 0 \rangle$ represent the activation of the second production function in neuron 1 and also the first production function from neuron 2.

Definition 5: Rows in the **Production Matrix** PM of an SNP system Π represent the new values of variables obtained from the current configuration and the production functions associated with the row. It is generated using the Function Matrix and a single Configuration Vector. Consider the r th row in the configuration matrix which is also the vector representation for the r th possible instance,

$$PM^{(k)} = \left[pm_{ij}^{(k)} \right]_{n \times v} \quad (8)$$

Where:

$$pm_{ij} = \begin{cases} \sum_{p=1}^v f_{ip} c_{rp} & \text{if function } f_i \text{ is in neuron } \sigma_s, \\ s \neq u \text{ and } (s, u) \in \text{syn} & \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

For the example in Figure 1, the Production Matrix is as follows:

$$PM^{(0)} = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 1 \\ 2 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The first row $\langle 0, 0, 2 \rangle$ represents the activation of the first production function which results in a value of 2 sent to the variables inside neuron 2. In this case the value is only sent to the third variable. The last row represents the non-activation of the last production function because the threshold is not satisfied.

Definition 6: The rows of the **Net Gain matrix** NG represent the possible outcomes from the application of the valid production functions. Since we are considering non-determinism multiple configurations may ensue. It is equal to:

$$NG_{q \times v}^{(k)} = S_{q \times n}^{(k)} \times PM_{n \times v}^{(k)} \quad (10)$$

For the example in Figure 1 we have the Net Gain Matrix as follows

$$NG^{(0)} = S_{q \times n}^{(0)} \times PM_{n \times v}^{(0)} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 1 \\ 2 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 1 \end{bmatrix}$$

Definition 7: The values inside variables are reset to zero when it is included in a function that is applied during a computation step but remains to be equal to its previous value when it is not used. With this we introduce the **Variable matrix**, having the same dimensions as the Net Gain Matrix it represents the base value to be added with the Net Gain Matrix to obtain the next configuration. Like the generation of the Production matrix we consider the r th configuration vector from the configuration matrix. It is defined as follows:

$$V^{(k)} = \left[v_{ij}^{(k)} \right]_{q \times v} \quad (11)$$

Where,

$$v_{ij} = \begin{cases} 0 & \text{if variable } j \text{ is in any of the used} \\ & \text{functions in } \sigma_i \\ c_{rj} & \text{otherwise} \end{cases} \quad (12)$$

For the given example there are no unused variables the Variable Matrix is,

$$V^{(0)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The configuration matrix for the next step $CM^{(k+1)}$ will be equal to the following.

$$CM_{q \times v}^{(k+1)} = V_{q \times v}^{(k)} + NG_{q \times v}^{(k)} \quad (13)$$

Finally we have $CM^{(1)}$ to be:

$$CM^{(1)} = V_{q \times v}^{(0)} + NG_{q \times v}^{(0)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 1 \end{bmatrix}$$

To show a case where there are 2 configuration vectors in the configuration matrix we also show $CM^{(2)}$. Details of F and L can be omitted as those matrices are static and not dependent on the current configuration of the system.

Consider the first configuration vector in $CM^{(1)}$ which in this case is $\langle 2, 2, 2 \rangle$ we call this $CM_0^{(1)}$. The Spiking Matrix and Production Matrix are as follows:

$$S_0^{(1)} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad PM_0^{(1)} = \begin{bmatrix} 0 & 0 & 4 \\ 0 & 0 & 2 \\ 2 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

With this the Net Gain matrix can be computed as follows:

$$NG_0^{(1)} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 4 \\ 0 & 0 & 2 \\ 2 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 4 \\ 2 & 2 & 2 \end{bmatrix}$$

The variable matrix for all configurations in $CM^{(1)}$ is,

$$V_0^{(1)} = V_1^{(1)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Lastly we can compute for the configuration matrix $CM_0^{(2)}$ from $CM^{(1)}$,

$$CM_{0(q \times v)}^{(2)} = V_{0(q \times v)}^{(1)} + NG_{0(q \times v)}^{(1)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 2 & 2 & 4 \\ 2 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 4 \\ 2 & 2 & 2 \end{bmatrix}$$

Now consider the second configuration vector in $CM^{(1)}$ which is $CM_1^{(1)} = \langle 2, 2, 1 \rangle$, the corresponding Spiking and Production matrix are,

$$S_1^{(1)} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} PM_1^{(1)} = \begin{bmatrix} 0 & 0 & 4 \\ 0 & 0 & 2 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

With this the Net Gain matrix can be computed as follows:

$$NG_1^{(1)} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 4 \\ 0 & 0 & 2 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 4 \\ 1 & 1 & 2 \end{bmatrix}$$

Finally the configuration matrix $CM_1^{(2)}$ from $CM^{(1)}$ can be obtained,

$$CM_{1(q \times v)}^{(2)} = V_{1(q \times v)}^{(1)} + NG_{1(q \times v)}^{(1)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 4 \\ 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 4 \\ 1 & 1 & 2 \end{bmatrix}$$

Combine configurations vector from $CM_0^{(2)}$ and $CM_1^{(2)}$ to obtain the next configuration from $CM^{(2)}$,

$$CM^{(2)} = \begin{bmatrix} 2 & 2 & 4 \\ 2 & 2 & 2 \\ 1 & 1 & 4 \\ 1 & 1 & 2 \end{bmatrix}$$

4 Simulation Algorithm

Algorithm 1 generates the computation tree from a given initial configuration using a breadth first search methodology. The algorithm ends after the specified depth is reached. The algorithm also checks all currently known states to avoid redundancy in the generation of configurations. The *computationHistory* tree object in line 4 of algorithm 1 represents the computation tree of a given NSNP system. Algorithms 2 and 3 handle the generation of the spiking and production matrix respectively.

Algorithm 1: General algorithm for NSNPS without delay

Require: $C^0, F, FL, syn, maxDepth$
Ensure: *ExploredStates* /* Resulting computation tree */

- 1: F, FL, T, syn are Global Constants
- 2: $UnexploredStates \leftarrow \{C^0\}$;
- 3: $ExploredStates \leftarrow \{\}$;
- 4: Create *computationHistory* tree;
- 5: *computationHistory.setRoot*(C^0);
- 6: $depth = 0$;
- 7: **while** $depth < max.depth$ **do**
- 8: $nextstates \leftarrow \{\}$;
- 9: **for** each *configuration* $\in UnexploredStates$ **do**
- 10: $S \leftarrow generateSM(configuration)$; /* shown in algorithm 2 */
- 11: $PM \leftarrow generatePM(configuration)$; /* shown in algorithm 3 */
- 12: $V \leftarrow checkActiveVars(S)$;
- 13: $NG \leftarrow S \times PM$;
- 14: $Cnext \leftarrow V + NG$;
- 15: **for** each $C_w \in Cnext$ **do**
- 16: **if** $C_w \notin ExploredStates$ **then**
- 17: Add C_w to the *nextstates* list;
- 18: Add *createNode*($C_w, configuration$) to *ComputationHistory*;
- 19: **end if**
- 20: **end for**
- 21: Add *configuration* to *ExploredStates* list;
- 22: Remove *configuration* from *UnexploredStates* list;
- 23: **end for**
- 24: Add elements of *nextstates* to *UnexploredStates* list;
- 25: $depth \leftarrow depth + 1$;
- 26: **end while**

Algorithm 2: *generateSM(configuration)*

Require: C, F, FL, syn, T
Ensure: $SM \rightarrow$ /* Spiking Matrix */

- 1: $Active \leftarrow FL$;
- 2: **for** each column j in FL **do**
- 3: $count \leftarrow 0$;
- 4: **for** each row i **do**
- 5: **if** $checkThreshold(C, i)$ **then**
- 6: $count \leftarrow count + 1$;
- 7: $Active_{ij} \leftarrow 1$;
- 8: **else**
- 9: $Active_{ij} \leftarrow 0$;
- 10: **end if**
- 11: **end for**
- 12: $n_j \leftarrow count$;
- 13: **end for**
- 14: $q \leftarrow 1$;
- 15: **for** each n_j **do**
- 16: **if** $n_j \neq 0$ **then**
- 17: $q \leftarrow q * n_j$;
- 18: **end if**
- 19: **end for**
- 20: Initialize a matrix $S_{q \times n}$ with each s_{ij} be equal to 0
- 21: **for** each neurom m **do**
- 22: $function \leftarrow getFunctions(m, Active)$;
- 23: **if** $n_m == 0$ **then**
- 24: **for** each element of $function$ say j **do**
- 25: **for** each row i in S **do**
- 26: $s_{ij} = 0$;
- 27: **end for**
- 28: **end for**
- 29: **else**
- 30: $i \leftarrow 1$;
- 31: $p \leftarrow q/n_m$;
- 32: **for** each j in $function$ **do**
- 33: $k \leftarrow 1$;
- 34: **while** $k \leq p$ **do**
- 35: $s_{ij} = 1$;
- 36: $k \leftarrow k + 1$;
- 37: $i \leftarrow i + 1$;
- 38: **end while**
- 39: **end for**
- 40: **end if**
- 41: $q = q/n_m$;
- 42: **end for**
- 43: $return(S)$;

Algorithm 3: *generatePM(configuration)*

Require: C, F, FL, syn, T
Ensure: $PM \rightarrow$ /* Production Matrix */

- 1: Initialize a matrix $PM_{n \times v}$ with each pm_{ij} be equal to 0
- 2: **for** each row i in F **do**
- 3: $sum \leftarrow 0$
- 4: **for** column j in F **do**
- 5: $sum \leftarrow sum + f_{ij} * c_j$
- 6: **end for**
- 7: $m \leftarrow getNeuronFromFunction(i)$
- 8: **for** each variable j **do**
- 9: **if** $(m, j) \in syn$ **then**
- 10: $pm_{ij} \leftarrow sum$
- 11: **end if**
- 12: **end for**
- 13: **end for**
- 14: *return*(PM)

Specifics of the implementations of the helper functions in algorithms 3 and 2 will not be discussed but a brief explanation of their use will be given. The *getNeuronFromFunction* function from line 7 in algorithm 3 takes as input the index of a given function and returns the index of the neuron where the function is located. The *checkThreshold* function checks whether or not the given function has satisfied the associated threshold value. The *getFunctions* takes as input a given neuron m and the Function Location matrix and returns a list of the index of functions that is present in neuron m . Lastly, *checkActiveVars* functions takes as input the obtained spiking matrix S and then creates a matrix with the same dimensions as the net gain matrix and outputs the variable matrix which identifies which variables have been used or not.

5 Simulations of universality modules

To demonstrate the matrix representation and simulation algorithm we presented, we simulate the example NSNP system in [14] as shown in Figure 2 using a Python implementation of the above mentioned algorithm to produce a computation tree from the required input of the algorithm. We also simulate the various NSNP modules in [14] that were created to show the Turing completeness of the NSNP model. The generated computation trees are shown in Figures 4 to 6, and cross validate the generated computation trees from computations trees presented in [14]. The computation tree for the example NSNP system was incomplete as shown in Figure 3 but [14] specified the halting configuration. For our implementation, we are able to generate a configuration tree from the sample configuration specifying a *max_depth* of 5. Notice the self looping configurations, for self looping states that have no children it can be said that those

configurations are already halting configurations since no other configurations other than the configuration itself can be produced.

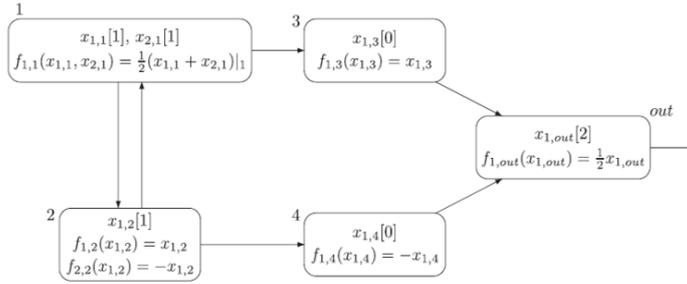


Fig. 2. NSNP system provided in [14]

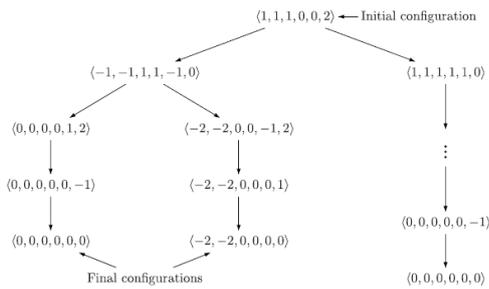


Fig. 3. Computation tree of NSNP system provided in [14]

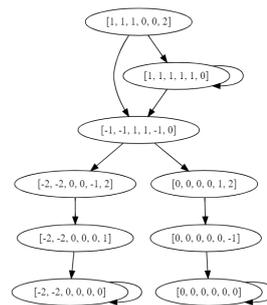


Fig. 4. Configuration tree generated using Algorithm 1

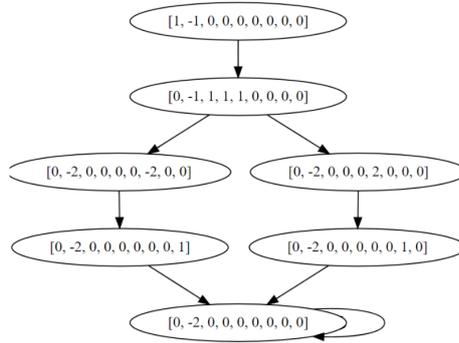


Fig. 5. Configuration tree of the ADD module presented in [14] generated using Algorithm 1

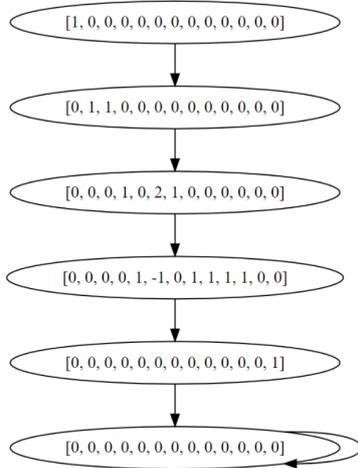


Fig. 6. Configuration tree of the SUB module presented in [14] generated using Algorithm 1

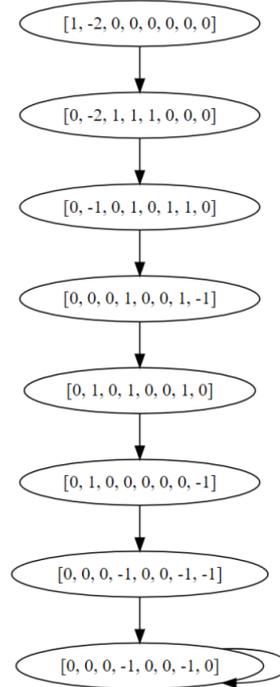


Fig. 7. Configuration tree of the FIN module presented in [14] generated using Algorithm 1

Figures 5 to 7 represent the configuration trees of modules presented in [14] used to show the correctness of the NSNP system model. A limitation of the simulator is that it can only generate computation trees from NSNP systems with actual values for the different variables whereas the computation trees presented in [14] was able to represent some computation trees with arbitrary inputs.

6 Non-deterministic Solution to the Subset Sum Problem

In order to further realize the potential of the constructions we have defined for simulating NSNP systems, an NSNP system was developed to solve the NP-Complete Subset Sum problem. The Subset sum problem has 2 inputs, a sum S and a multiset $V = \{v_0, v_1, \dots, v_n\}$ with $S, v_i \in N$ and $0 \leq i \leq n$. The subset sum problems asks if there exists a subset $V' \subseteq V$ such that $\sum_{v' \in V'} v' = S$.

SNP systems and its variants have been already used to solve this problem as shown in [9] and [5]. Among solutions presented it can be seen that there are two types namely uniform and non-uniform, a non-uniform solution has variable number of neurons dependent on the values in the multiset V while a uniform solution has a fixed amount of neurons regardless of the values in the multiset V . Non-uniformity of SNP systems presented in [9] and [5] come from the fact that they are required to use multiple neurons to represent the integer values because of the singleton alphabet limitation of such systems. From literature it would seem that searching for a non-uniform solution be the first step to solving this problem but considering the semantics of NSNP systems, an existence of a non-uniform solution would result in a trivial manipulation to convert such a non-uniform solution to a uniform solution.

Consider a family \mathbf{II} of nondeterministic and uniform NSNP systems to solve Subset sum. Each $\Pi(q) \in \mathbf{II}$, for $q \in I(V)$, has a certain structure solely dependent on the number of elements in V while initial values of the variables in each neurons are dependent on values in V and the sum S . This structure is represented in Figure 8.

Variables v_0 to v_{n-1} in neurons a_0 to a_{n-1} represent the values inside V . The two production functions for each of the a neurons simulate the nondeterministic choice of using or not a given $v_i \in V$ with this all possible configurations will be checked. Both of these rules can be activated during the first step, the second set of neurons (ones labeled with b) are in charge of only letting positive values pass through. Let B be the sum of the values entering s_2 from the b neurons. Neuron s_o stores the target sum $S + 1$ but since we are only considering NSNP systems without delays an auxiliary neuron s_1 was introduced to represent the delay so that values from the a neurons will arrive at s_2 at the same time step. In NSNP systems there is no natural way to test for equality with just using one neuron. Neurons s_2 and s_3 together simulate the test for equality of S and B . The computation in s_2 only proceeds when the sum of $-(S + 1)$ and B is greater than or equal to -1 , this is the case when $B \geq S$. The computation in s_3 proceeds when the value obtained from s_2 is greater than or equal to 1, this is the cause when $B \leq S$. Satisfying these 2 conditions means that B and S are

equal and that there exists a subset $V' \in V$ such that $\sum_{v' \in V'} v' = S$. Consequently not satisfying these conditions means such a set does not exist which results in the inactivity of neuron s_3 for the duration of the computation.

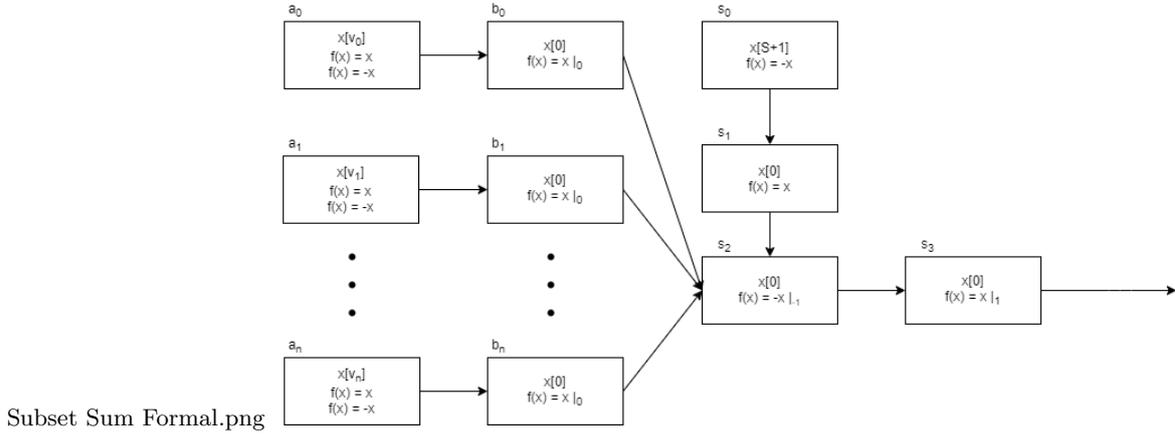


Fig. 8. SNSP system to solve the Subset Sum problem

We generate 2 instances of the subset sum problem to show results for an SNSP system with and without a solution. Figure 9 represent the case *A* where $V = \{1, 2\}$ and $S = 3$ and Figure 10 represent the case *B* where $V = \{1, 2\}$ and $S = 4$. Notice that for case *A* in Figure 9 there is a branch in the computation with an extra step, this indicates a subset $V' \subseteq V$ exists such that when the elements of V' are added together is equal to S which means a solution to the subset sum instance exists. For case *B* in Figure 10 wherein there is no branch with an extra step, no $V' \subseteq V$ exists such that the sum of the the elements of V' is equal to S meaning no solution to the subset sum instance exists.

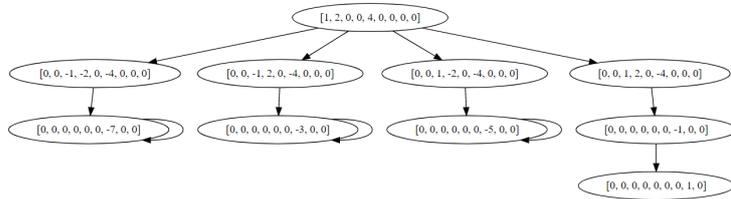


Fig. 9. Configuration tree for the instance with solution to the subset sum problem

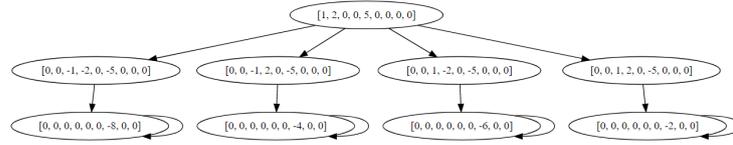


Fig. 10. Configuration tree for the instance without solution to the subset sum problem

7 Final Remarks

The presented implementation of the production matrix could be improved such that the variable matrix would be unnecessary. For the current implementation, the stopping criterion is dictated by the desired depth of the computation tree. An improvement could be made by stopping the computation of deeper states when a halting configuration is reached or a state previously computed is encountered. Example computations shown in Section 3 failed to show some cases in the activation of productions functions, making the example configuration be able to show all cases would provide less confusion for the reader. The implementation of the production functions did not include any constant values, only variables were included as was presented in [14]. The presented matrix representation also be extended such that it can accommodate for semantics of emerging works the general topic of NSNP systems like in [15]. While there is already work for demonstrating possible learning behavior for SNP systems like in [12] the learning capabilities of NSNP systems can also be investigated and show whether there is an advantage of using the production function and variable semantics for learning purposes. For future work, we are currently extending our simulation and solutions to include the following: implementation with constant values and a deterministic solution to the subset sum problem. Lastly we are also currently working on a parallel GPU simulator to speed up simulations even further.

Acknowledgements

K.J. Ballesteros and R.T.A. De La Cruz acknowledge support from ERDT scholarships of the DOST-SEI, Philippines. F.G.C. Cabarle acknowledges support from ERDT of DOST-SEI, and the Dean Ruben A. Garcia PCA from the University of the Philippines Diliman. H.N. Adorna is supported by the Semirara Mining Corporation PCA, also from UP Diliman.

References

1. Aboy, B.C., Bariring, E.J., Carandang, J.P.A., Cabarle, F.G.C., Dela Cruz, R.T., Adorna, H.N., Martinez-del Amor, M.A.: Optimizations in CuSNP simulator for Spiking Neural P Systems on CUDA (2019)

2. Adorna, H.N.: Matrix representation of spiking neural p systems: Revisited. In: Păun, G. (ed.) Proc 20th International Conf on membrane Computing, August 3-8, 2019. pp. 227–248. Editura BIBLIOSTAR (2019)
3. Martínez-del Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I., Cabarle, F.G.C., Adorna, H.N.: Simulation of spiking neural p systems with sparse matrix-vector operations. *Processes* **9**(4) (2021). <https://doi.org/10.3390/pr9040690>, <https://www.mdpi.com/2227-9717/9/4/690>
4. Cabarle, F.G.C., Adorna, H.N., Martínez-del Amor, M.A., Pérez-Jiménez, M.J.: Improving gpu simulations of spiking neural p systems. *Romanian Journal of Information Science and Technology* **15**, 5–20 (06/2012 2012), [http : //www.imt.ro/romjst/Volum15/Number151/cuprins151.htm](http://www.imt.ro/romjst/Volum15/Number151/cuprins151.htm)
5. Cabarle, F.G.C., Cruz, R.T.A.D.L., Cailipan, D.P.P., Zhang, D., Liu, X., Zeng, X.: On solutions and representations of spiking neural p systems with rules on synapses. *Information Sciences* **501**, 3049 (2019). <https://doi.org/10.1016/j.ins.2019.05.070>
6. Carandang, J.P.A., Villaflores, J.M.B., Cabarle, F.G.C., Adorna, H.N., Martínez-del Amor, M.A.: Cusnp: Spiking neural p systems simulators in cuda. *Romanian Journal of Information Science and Technology* **20**, 57–70 (2017), [http : //www.imt.ro/romjst/Volum20/Number201/cuprins201.htm](http://www.imt.ro/romjst/Volum20/Number201/cuprins201.htm)
7. Ionescu, M., Păun, G., Yokomori, T.: Spiking Neural P Systems. *Fundamenta Informaticae* **71**(2,3), 279–308 (Feb 2006)
8. Jimenez, Z.B., Cabarle, F.G.C., de la Cruz, R.T.A., Buño, K.C., Adorna, H., Hernandez, N.H.S., Zeng, X.: Matrix representation and simulation algorithm of spiking neural p systems with structural plasticity. *Journal of Membrane Computing* **1**, 145–160 (2019)
9. Leporati, A., Zandron, C., Ferretti, C., Mauri, G.: Solving numerical np-complete problems with spiking neural p systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing*. pp. 336–352. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
10. Păun, G.: *Membrane Computing: An Introduction*. Springer Berlin Heidelberg (2002)
11. Păun, G., Păun, R.: Membrane computing and economics: Numerical p systems. *Fundam. Inf.* **73**(1,2), 213227 (Apr 2006)
12. Song, T., Pan, L., Wu, T., Zheng, P., Wong, M.L.D., Rodriguez-Patn, A.: Spiking neural p systems with learning functions. *IEEE Transactions on NanoBioscience* **18**(2), 176–190 (2019)
13. Song, T., Pang, S., Hao, S., Rodriguez-Patn, A., Zheng, P.: A parallel image skeletonizing method using spiking neural p systems with weights. *Neural Processing Letters* **50**, 14851502 (2018). <https://doi.org/10.1007/s11063-018-9947-9>
14. Wu, T., Pan, L., Yu, Q., Tan, K.C.: Numerical spiking neural p systems. *IEEE Transactions on Neural Networks and Learning Systems* pp. 1–15 (2020)
15. Yin, X., Liu, X., Sun, M., Ren, Q.: Novel numerical spiking neural p systems with a variable consumption strategy. *Processes* **9**(3) (2021). <https://doi.org/10.3390/pr9030549>, <https://www.mdpi.com/2227-9717/9/3/549>
16. Zeng, X., Adorna, H., Martínez-del Amor, M.Á., Pan, L., Pérez-Jiménez, M.J.: Matrix representation of spiking neural p systems. In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing: 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010. Revised Selected Papers*. pp. 377–391. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18123-8_29, [https : //doi.org/10.1007/978-3-642-18123-8_29](https://doi.org/10.1007/978-3-642-18123-8_29)

Array P Systems and Pure 2D Context-free Grammars with Independent Mode of Rewriting

Somnath Bera¹[0000-0002-1886-5681], Rodica Ceterchi²[0000-0001-5354-8588],
Sastha Sriram³[0000-0003-0604-2159], and K.G.
Subramanian^{4,#}[0000-0001-8726-5850]

¹ School of Advanced Sciences-Mathematics, Vellore Institute of Technology,
Chennai, Tamil Nadu 600 127 India

² University of Bucharest, Faculty of Mathematics and Computer Science,
14 Academiei St, 010014 Bucharest, Romania

³ School of Arts, Science and Humanities, SASTRA Deemed University, Tanjore,
Tamil Nadu 613 401 India

⁴ School of Mathematics, Computer Science and Engineering, Liverpool Hope
University, Hope Park, Liverpool L16 9JD, UK

#Honorary Visiting Professor, Corresponding author
kgsmani1948@gmail.com

Abstract. Rewriting array P systems for generation of rectangular picture arrays have been considered with the rules in membranes and the application of the rules as in a pure 2D context-free grammar (*P2DCFG*) and its variants. Here we introduce in *P2DCFG*, a different mode of rewriting of an array, which we call as *independent* mode. We then consider rewriting array P systems involving *P2DCFG* type of rules but with the independent mode of rewriting. We show that the array generative power is increased in the framework of P systems. This framework also allows for the treatment of the so-called “extended” array grammars.

Keywords: Two dimensional languages · Pure context-free grammars · P systems.

1 Introduction

Based on pure context-free grammars [10] which have been extensively investigated for their language generating power, a simple but effective non-isometric 2D grammar model, called pure 2D context-free grammar (*P2DCFG*) was introduced in [23, 24] in the area of two-dimensional picture languages [8, 14, 15, 27], to generate rectangular picture array languages. In a *P2DCFG*, all symbols in any column or any row of the rectangular array are rewritten at a time by equal length strings, thus maintaining the array to be rectangular. Several properties [1, 2] and variants [9, 25] of *P2DCFG* have been studied.

On the other hand, in the area of membrane computing [12, 13], a computing model based on the membrane structure and the functioning of living cells,

was introduced by Păun in [11] and is now referred to as P system. This area of membrane computing has seen a vast growth both in terms of theoretical results [13] and application studies [28].

Formal language theory [16, 17], which is a classical area of theoretical computer science, has close connections with membrane computing. The two areas of membrane computing and two-dimensional picture array grammars were linked in [3] by developing an array P system for dealing with the problem of generation of two dimensional (2D) objects or picture arrays based on Chomsky type array grammars of the isometric variety. Several models of array P systems (see, for example, [4, 22, 26]) have been subsequently introduced and studied in the area of two-dimensional picture languages.

Here we consider pure 2D context-free grammars, which belong to the non-isometric variety of array grammars, with a variation in the mode of rewriting of a picture array, which we call as *independent mode*. The resulting class of picture languages is shown to be incomparable with the family of picture languages generated by *P2DCFG*. We then consider an array P system with objects in the regions of this P system as rectangular picture arrays and rules to generate picture arrays as *P2DCFG* kind of rules but with an independent mode of rewriting. We show that the use of two membranes gives more picture array generative power. We also provide an application of this array P system in generating certain floor designs, referred to as “kolam” patterns.

2 Preliminaries

Let T be a finite alphabet. A word or a string $w = w_1w_2 \cdots w_n$, ($n \geq 1$) over T is a finite sequence of symbols from T . We denote by $|w|$, the length of the word w . The set of all words over T , including the empty word λ with no symbols, is denoted by T^* . We call words of T^* also as *row words*. For any word $w = a_1a_2 \cdots a_n$, we denote by w^t the word w written vertically as follows and call w^t as a *column word*:

$$\begin{array}{c} a_1 \\ a_2 \\ \vdots \\ a_n \end{array}$$

Since the transpose of a row is a column and viceversa, we have $(w^t)^t = w$. A rectangular $m \times n$ array M over T , called picture array, is of the form

$$M = \begin{array}{ccc} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{m1} & \cdots & p_{mn} \end{array}$$

where each $p_{ij} \in T$, $1 \leq i \leq m$, $1 \leq j \leq n$. The set of all picture arrays over T is denoted by T^{**} , which includes the empty array λ . We write $T^{+++} = T^{**} - \{\lambda\}$.

We refer to [16, 17] for concepts related to formal languages and to [14, 15] for array grammars. For notions related to P Systems we refer to [11, 12]. We now recall a pure 2D context-free grammar introduced in [23, 24].

Definition 1. A pure 2D context-free grammar (*P2DCFG*) is a 4-tuple $G = (T, P_1, P_2, I)$ where

- i) T is a finite set of symbols;
- ii) P_1 is a finite set of column tables c , where c is a finite set of context-free rules of the form $a \rightarrow \alpha, a \in T, \alpha \in T^*$ satisfying the property that for any two rules $a \rightarrow \alpha, b \rightarrow \beta$ in c , we have $|\alpha| = |\beta|$ i.e. the words α and β have equal length;
- iii) P_2 is a finite set of row tables r , where r is a finite set of rules of the form $d \rightarrow \gamma^t, d \in T, \gamma \in T^*$ such that for any two rules $d \rightarrow \gamma^t, e \rightarrow \delta^t$ in r , we have $|\gamma| = |\delta|$;
- iv) $I \subseteq T^{**} - \{\lambda\}$ is a finite set of initial (axiom) arrays.

A derivation in a *P2DCFG* G is defined as follows: For $p, q \in T^{**}$, q is derived in G from a picture p , written $p \Rightarrow q$, either (i) by rewriting in parallel all the symbols in a column of p , rewriting each symbol by a rule in some column table or (ii) by rewriting in parallel all the symbols in a row of p , rewriting each symbol by a rule in some row table. All the rules used to rewrite a column (or a row) of symbols should belong to the same table. The reflexive, transitive closure of \Rightarrow is denoted by \Rightarrow^* .

The picture language generated by G is the set of picture arrays $L(G) = \{M \in T^{**} \mid M_0 \Rightarrow^* M \text{ for some } M_0 \in I\}$. The family of picture languages generated by *P2DCFGs* is denoted by *P2DCFL*.

Example 1. Consider the *P2DCFG* $G_1 = (T, P_1, P_2, \{M_0\})$ where $T = \{a, b, d, e\}$, $P_1 = \{c_1, c_2\}$, $P_2 = \{r\}$, where $c_1 = \{a \rightarrow ab, d \rightarrow da\}$, $c_2 = \{a \rightarrow a, d \rightarrow e\}$,
 $r = \left\{ d \rightarrow \begin{matrix} a & b \\ d & a \end{matrix}, a \rightarrow \begin{matrix} b \\ a \end{matrix} \right\}$, and $M_0 = \begin{matrix} a & b & b \\ a & b & b \\ d & a & a \end{matrix}$.

G_1 generates a picture language L_1 consisting of picture arrays p of size (m, n) , $m, n \geq 3$ with $p(i, 1) = p(m, j) = a$, for $1 \leq i \leq m - 1, 2 \leq j \leq n$; $p(m, 1) = d$ or $p(m, 1) = e$; $p(i, j) = b$, otherwise. We note that a derivation in G_1 , starting from the axiom array M_0 , generates picture arrays of the forms

$$\begin{matrix} a & b & \cdots & b & & a & b & \cdots & b \\ \vdots & \vdots & \ddots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ a & b & \cdots & b & & a & b & \cdots & b \\ d & a & \cdots & a & & e & a & \cdots & a \end{matrix}$$

since the column table c_1 is applicable to only the leftmost column $(a \cdots ad)^t$, rewriting in parallel all the symbols a and d in that column, thereby adding the symbol b to the immediate right of each a while adding the symbol a to

the immediate right of d . Likewise, the row table r is applicable to only the bottommost row and adds a row of the form $ab \cdots b$ just above it. Likewise the column table c_2 is applicable to only the leftmost column $(a \cdots ad)^t$, rewriting in parallel all the symbols a as a itself but changing d in that column as e , and after this no table of rules is applicable.

3 Pure 2D context-free grammar in independent mode

We now introduce a different mode of rewriting in a pure 2D context-free grammar, which we call as independent mode, based on a corresponding notion in the study of two-dimensional insertion systems considered in [7].

Definition 2. *A pure 2D context-free grammar in independent mode (IP2DCFG) $G_I = (T, P_1, P_2, I)$ has its components T, P_1, P_2, I as in the P2DCFG in Definition 1, with a difference in the mode of rewriting of a picture array, which we call as independent mode, and which is done as given below:*

A direct derivation of a picture array M_2 from an $m \times n$ picture array M_1 , written as $M_1 \Rightarrow_i M_2$, is done in the following manner: In applying the rules of a row (respy. column) table r (respy. c) to the $m \times n$ picture array M_1 , only one symbol in each of the m rows (respy. n columns) is rewritten at a time and it can be any symbol in that row (respy. column). All the symbols (chosen for rewriting) should have rules in the row table r (respy. column table c). Otherwise, the table of rules is not applicable. If a picture array Y is obtained from a picture array X using a IP2DCFG, through a sequence of direct derivation steps, we write $X \Rightarrow_i^ Y$. Note that the lengths of the right sides of all the rules in a column table or a row table, are the same and so the derived array is also a rectangular array.*

*The picture language generated by a IP2DCFG G_I is the set of picture arrays $L(G) = \{M \in T^{**} \mid M_0 \Rightarrow_i^* M \text{ for some } M_0 \in I\}$. The family of picture languages generated by IP2DCFGs is denoted by IP2DCFL.*

We illustrate with an example.

Example 2. Consider the IP2DCFG $G_2 = (T, P_1, P_2, \{M_0\})$ where $T = \{a, b, d, e, x\}$, $P_1 = \{c_1, c_2\}$, $P_2 = \{r\}$, where $c_1 = \{a \rightarrow ab, x \rightarrow xx, d \rightarrow bd\}$, $c_2 = \{a \rightarrow a, x \rightarrow x, d \rightarrow e\}$, $r = \left\{ a \rightarrow \begin{matrix} a \\ b \end{matrix}, x \rightarrow \begin{matrix} x \\ x \end{matrix}, d \rightarrow \begin{matrix} b \\ d \end{matrix} \right\}$, and

$$M_0 = \begin{matrix} a & b & b \\ b & x & b \\ b & b & d \end{matrix}$$

G_2 generates a picture language L_2 consisting of picture arrays p of size $m \times n$, $m, n \geq 3$ with $p(1, 1) = a$, $p(1, j) = p(i, 1) = b$, for $2 \leq j \leq n$ and $2 \leq i \leq m$, $p(m, j) = p(i, n) = b$, for $2 \leq j \leq n - 1$ and $2 \leq i \leq m - 1$, $p(m, n) = d$ or e ,

and

$$\begin{array}{c} b a a \\ b a a . \\ d b b \end{array}$$

But it can be seen that this picture language L cannot be generated by any $IP2DCFG$. In fact arrays of the form p_1 can be generated in the independent mode only by a column table of rules $c = \{b \rightarrow bb, d \rightarrow da\}$, since rules for rewriting a can not be included in such a table as it will result in picture arrays not in the language. But then the column table c could be applied to a picture array of the form p_2 in the independent mode, again resulting in picture arrays not in the language. Similar reasoning can be done for picture arrays p_2 .

On the other hand, consider the picture language L' consisting of $2 \times n, n \geq 3$ picture arrays q such that $q(1, 1) = q(2, 2) = a, q(i, j) = b$, otherwise. L' is in $IP2DCFL$ generated by a $IP2DCFG$ with an axiom array $\begin{array}{c} a b \\ b a \end{array}$ and a column table $c = \{a \rightarrow ab\}$. It cannot be generated by any $P2DCFG$ as we need to include a rule for the symbol b in forming a column table of rules due to the requirement that in a $P2DCFG$, all symbols in a single column should be rewritten at a time. But inclusion of a rule for b will allow rewriting of the first b in the second row (irrespective of which symbol is rewritten in the first row), thus generating picture arrays not in the language.

4 Array P system based on $IP2DCFG$

We consider now an array P system model with the membranes of the P system containing picture array objects and column or row tables of rules as in $IP2DCFG$ in the sense that the rewriting is in independent mode.

Definition 3. An array P system (of degree $m \geq 1$) with $IP2DCFG$ kind of rules is a construct

$$\Pi = (T, \mu, A_1, \dots, A_m, R_1, \dots, R_m, i_o),$$

where T is the alphabet consisting of terminal symbols, μ is a membrane structure with m membranes labelled in a one-to-one manner with $1, 2, \dots, m$; A_1, \dots, A_m are finite sets (can be empty) of rectangular picture arrays over T with A_i in the membrane or region labelled i for $1 \leq i \leq m$; R_1, \dots, R_m are finite sets of column tables or row tables of context-free rules over T (as in a $IP2DCFG$) with R_i in the membrane or region labelled i for $1 \leq i \leq m$. A region can contain both column tables of rules and row tables of rules. The application of a column or row table is as done in a $IP2DCFG$. The tables have attached targets here, out, in, in_j (in general, here is omitted) and i_o is the label of the output membrane which is an elementary membrane of μ .

A computation in Π is done as in an array-rewriting P system [3] with the

successful computations being the halting ones; each rectangular picture array in each region of the system, which can be rewritten by a column table of rules or a row table of rules, associated with that region, should be rewritten. This means that a region can contain column table of rules and/or row table of rules but one table (column or row) of rules is applied at a time to a picture array and the rewriting is done as in a *IP2DCFG*. The picture array obtained by rewriting is retained in the same region if the target associated with the table used is here or sent to an immediate outer region (respy. directly inner region, nondeterministically chosen), if the target is out (respy. in). If the target is in_j then the array is immediately sent to a directly inner membrane with label j. If no internal membrane exists, then a table with the target indication in cannot be used. A computation is successful only if it stops, that is, a configuration is reached where no table of rules can be applied to the existing arrays in the regions.

The result of a halting computation consists of rectangular picture arrays over T collected in the output membrane with label i_o in the halting configuration. Note that all the picture arrays that stay at the output membrane in the halting configuration will belong to the picture language since there are only one kind of symbols, namely terminal symbols.

The set of all picture arrays generated by such a system Π is denoted by $IAL(\Pi)$. The family of all array languages $IAL(\Pi)$ generated by such systems Π as above, with at most m membranes, is denoted by $IAP_m(IP2DCFG)$.

Example 3. Consider the picture language L_{sq} consisting of square sized $n \times n, n \geq 3$, picture arrays p with $p(1,1) = a, p(1,j) = p(i,1) = b$, for $2 \leq j \leq n$ and $2 \leq i \leq n, p(n,j) = p(i,n) = b$, for $2 \leq j \leq n-1$ and $2 \leq i \leq n-1, p(n,n) = e, p(i,j) = x$, otherwise. We construct an array P system Π_{sq} with only one membrane and with the only region containing tables having *IP2DCFG* kind of rules applied in independent mode. Π_{sq} is given by

$$\Pi = (T, \mu, A_1, R_1, 1),$$

where

- i) $T = \{a, b, d_1, d_2, e, x\}$
- ii) $\mu = [1]_1$
 $a \ b \ b$
- iii) $A_1 = \{ \begin{matrix} b \ x \ b \\ b \ b \ d_1 \end{matrix} \}$,
- iv) R_1 consists of a row table r and two column tables c_1 and c_2 , each with target *here*. The tables of rules are given as follows: $c_1 = \{a \rightarrow ab, d_1 \rightarrow bd_2, x \rightarrow xx\}$, $c_2 = \{a \rightarrow a, d_1 \rightarrow e, x \rightarrow x\}$, $r = \{a \rightarrow \begin{matrix} a \\ b \end{matrix}, x \rightarrow \begin{matrix} x \\ x \end{matrix}, d_2 \rightarrow \begin{matrix} b \\ d_1 \end{matrix}\}$.

In the array P system Π_{sq} , the only membrane or region labelled 1 which is also the output membrane, initially, contains the array

$$\begin{array}{c} a \ b \ b \\ b \ x \ b \ . \\ b \ b \ d_1 \end{array}$$

If the column table c_2 is applied, then the array generated is

$$\begin{array}{c} a \ b \ b \\ b \ x \ b \\ b \ b \ e \end{array}$$

which is collected in the language as membrane 1 is the output membrane and no table of rules is applicable in the region at this moment with the computation coming to a halt. If the column table c_1 is applied to the axiom array in region 1, then the array generated is

$$\begin{array}{c} a \ b \ b \ b \\ b \ x \ x \ b \ . \\ b \ b \ b \ d_2 \end{array}$$

The row table r can be applied now which generates the array

$$\begin{array}{c} a \ b \ b \ b \\ b \ x \ x \ b \\ b \ x \ x \ b \\ b \ b \ b \ d_1 \end{array}$$

and the process can be repeated. If the column table c_2 is applied during the process instead of c_1 in region 1, the array

$$\begin{array}{c} a \ b \ b \ b \\ b \ x \ x \ b \\ b \ x \ x \ b \\ b \ b \ b \ e \end{array}$$

is generated changing the symbol d_1 to e in the array. The computation comes to a halt and the array is collected in the language generated by Π_{sq} . Note that the generated array at the halting configuration will have an equal number of rows and columns and will be an element of L_{sq} . Thus the system Π_{sq} generates the language L_{sq} .

We now examine the generative power of the array-rewriting P system with *IP2DCFG* kind of rules in independent mode of derivation.

Theorem 2. $IP2DCFL \subset IAP_1(IP2DCFG) \subset IAP_2(IP2DCFG)$.

Proof. The inclusion $IP2DCFL \subseteq IAP_1(IP2DCFG)$ can be seen as follows: Let $L \in IP2DCFL$ and $G_I = (T, P_1, P_2, I)$ be an *IP2DCFG* generating

L . We construct an array P system of degree 1 with $IP2DCFG$ kind of rules, $\Pi = (T \cup \bar{T}, [1]_1, A, R, 1)$ where $\bar{T} = \{\bar{a} \mid a \in T\}$. In other words the alphabet of Π contains all the symbols of the alphabet T of G_I and in addition contains the “barred” version of every symbol of T ; A contains all the picture arrays of I but every symbol in each picture array of I is replaced by its barred version. Like wise R contains all the column tables of P_1 and all the row tables of P_2 but each symbol in the right and left sides of every rule in the tables, is replaced by the corresponding barred symbol. In addition R contains a new column table $c = \{\bar{a} \rightarrow a \mid a \in T\}$. We denote by \bar{M} the array obtained from the array M by replacing each symbol of M by the corresponding barred symbol. It can be seen that for every direct derivation $M_1 \Rightarrow M_2$ in G_I , there is a computation in Π with the array \bar{M}_1 generating \bar{M}_2 which then generates M_2 by the application of the rules of the column table c and the computation halts. Hence every picture array in L generated in G_I from an axiom array in I is also computed by Π . Thus $L \in IAP_1(IP2DCFG)$.

The proper inclusion follows from the picture array language L_{sq} in Example 3 which shows that $L_{sq} \in IAP_1(IP2DCFG)$. On the other hand this picture language cannot be generated by any $IP2DCFG$ since we need to have some control on the application of column table of rules and row table of rules to maintain square shape of the picture arrays generated by a $IP2DCFG$. We can use some “intermediate” symbols in order to alternate the application of the column and row tables of rules but then this will result in picture arrays not in the language.

The inclusion $IAP_1(IP2DCFG) \subseteq IAP_2(IP2DCFG)$ in statement (i) follows from the definition of the family $IAP_m(IP2DCFG)$. For the proper inclusion we consider the language L_{ab} consisting of picture arrays p of size $m \times (4n + 2)$, $m \geq 2$, $n \geq 1$, where p is such that every row of p is of the form $xa^n b^n a^n b^n y$ ($n \geq 1$) over the terminal symbols $\{x, y, a, b\}$. The language L_{ab} belongs to $IAP_2(IP2DCFG)$ generated by the P system with two membranes having the membrane structure $[1 [2]_2]_1$ and the only axiom array

$$\begin{array}{c} x d e y \\ x d e y \end{array}$$

in membrane 1 initially. Membrane 1 has a row table r with target *here* and two column tables c_1, c_2 with target *in*. Membrane 2, which is the output membrane, has a column table c_3 with target *out* and another column table c_4 with target *here*.

The tables of rules are given below:

$$r = \left\{ x \rightarrow \begin{array}{c} x \\ x \end{array}, a \rightarrow \begin{array}{c} a \\ a \end{array}, b \rightarrow \begin{array}{c} b \\ b \end{array}, y \rightarrow \begin{array}{c} y \\ y \end{array} \right\}, c_1 = \{d \rightarrow adb\}, c_2 = \{e \rightarrow ab\}, \\ c_3 = \{e \rightarrow aeb\}, c_4 = \{d \rightarrow ab\}.$$

Application of the rules of the row table r will add the row $xa^n b^n a^n b^n y$ to the array which will remain in membrane 1. This row table can be applied any number of times. A computation starts with an application of the rules of the column table c_1 or c_2 in membrane 1 to the axiom array. If the rules of the column

table c_1 in membrane 1 are applied to the axiom array , then the generated array

$$\begin{array}{c} x a d b e y \\ x a d b e y \end{array}$$

will be sent to membrane 2 and if the rules of the column table c_3 are applied, then the generated array is

$$\begin{array}{c} x a d b a e b y \\ x a d b a e b y \end{array}$$

which is sent back to membrane 1. This process can repeat. If the rules of the column table c_2 are applied to the axiom array in membrane 1 (instead of c_1), the generated array

$$\begin{array}{c} x d a b y \\ x d a b y \end{array}$$

is sent to membrane 2. Here only the column table c_4 is applicable generating the array

$$\begin{array}{c} x a b a b y \\ x a b a b y \end{array}$$

which is collected in the language as the computation halts. On the other hand, if in membrane 1, the rules of the column table c_1 are applied, then the generated array is sent to the inner membrane 2 and if the rules of the column table c_4 are applied which replaces the symbol d by ab in the rows, then the array remains there but the computation does not halt as the rules of the column table c_3 are applicable generating and sending the resulting array to membrane 1. Here the rules of the column table c_2 are only applicable with the resulting array sent to the region 2 and the computation halts. Here again it is collected in the language. Note that no other sequence of application of the rules of the tables will be a correct sequence. Thus the picture language L_{ab} is generated and hence belongs to $IAP_2(IP2DCFG)$. This picture language cannot belong to $IAP_1(IP2DCFG)$ as there is only one membrane and so the technique of alternately generating the first “block” of $a^n db^n$ and the second “block” of $a^n eb^n$ using “intermediate” symbols, cannot be managed for ever as the number of rows can keep increasing.

Remark : Analogous to the statement in Theorem 2, in the case of $P2DCFL$ [23, 24], we have $P2DCFL \subset AP_1(P2DCFG) \subset AP_2(P2DCFG)$, which corrects the erroneous statements in [26, Theorem 1, Page 1905]. The proof of this statement is similar to the proof of Theorem 2.

5 Array P systems and Extended 2D Grammars

An earliest two-dimensional picture array generating model introduced by the Siromoney group [19], originally called matrix grammar [20] and subsequently referred to as $2DCFG$ in [16], involves two phases of rewriting. The first phase generates strings over “intermediate” symbols using context-free string grammar rules and in the second phase, these strings are rewritten by groups of normal

form regular grammar rules such as a group of nonterminal rules of the form $A \rightarrow aB$ or a group of terminal rules of the form $A \rightarrow a$ in the vertical direction to produce the columns of a rectangular picture array over a set of terminal symbols. The “intermediate” symbols of the first phase will be the start non-terminal symbols in the second phase. This model has been extensively used in 2D grammar studies. An extension of this 2D grammar was introduced in [21]. We do not give here the formal definition of this extended 2D grammars but informally mention the details in the extended model. The first phase is similar to the Siromoney matrix grammar [20] but in the second phase different sets of regular nonterminal rules or sets of regular terminal rules, called tables of rules are allowed and applied in the rewriting. We will call the extended 2D grammar model as $T2DCFG$ or $T2DCSG$ depending on whether the first phase involves a CFG or a CSG . The corresponding families of picture languages are denoted by $T2DCF$ and $T2DCSL$, following the notation used in [16]. We now compare the array P system considered here with these extended 2D grammars.

Theorem 3. (i) $IAP_2(IP2DCFG) \cap T2DCSL \neq \emptyset$.
(ii) $IAP_2(IP2DCFG) \setminus T2DCF \neq \emptyset$.

Proof We consider the picture language L consisting of $m \times (4n + 2)$, ($m \geq 4, n \geq 1$) picture arrays p such that the first row of p is of the form $xa^n b^n a^n b^n y$ ($n \geq 1$) and the next few rows are of the form $xz^{4n}y$ ($n \geq 1$) followed by a row of the form $qz^{4n}r$ ($n \geq 1$) and the remaining rows are of the form $sz^{4n}t$ ($n \geq 1$) over the terminal symbols $\{x, y, q, r, s, t, z, a, b\}$. This picture language is in $T2DCSL$ but is not in $T2DCF$ since the first row is a strictly context-sensitive language. In fact the corresponding $T2DCSG$ will generate in the first phase the $CSL \{XA^n B^n A^n B^n Y \mid n \geq 1\}$ where X, Y, A, B are “intermediate” symbols which will serve as the start symbols for the second phase. The tables of rules in the second phase are $t_1 = \{X \rightarrow xX, A \rightarrow aZ, B \rightarrow bZ, Y \rightarrow yY\}$, $t_2 = \{X \rightarrow xX, Z \rightarrow zZ, Y \rightarrow yY\}$, $t_3 = \{X \rightarrow qU, Z \rightarrow zZ, Y \rightarrow rV\}$, $t_4 = \{U \rightarrow sU, Z \rightarrow zZ, V \rightarrow tV\}$, and $t_5 = \{U \rightarrow s, Z \rightarrow z, V \rightarrow t\}$. Application of the rules of the table t_1 will generate the first row $xa^n b^n a^n b^n y$ of the array p . This can be followed by the application of the rules of the table t_2 yielding the rows of the form $xz^{4n}y$ till the rules of the table t_3 are applied. This will yield the row $qz^{4n}r$. Likewise using the table t_4 the rows of the form $sz^{4n}t$ are generated and the derivation ends when the table t_5 is used generating the required array.

The language L belongs to $IAP_2(IP2DCFG)$ generated by the P system with two membranes having the membrane structure $[_1[_2]_2]_1$ and the only axiom array

$$\begin{array}{c} x d e y \\ x z z y \\ q z z r \\ s z z t \end{array}$$

in membrane 1 initially. Membrane 1 has two row tables r_1, r_2 with target *here* and two column tables c_1, c_3 with target *in*. Membrane 2 has a column table

c_2 with target *out* and another column table c_4 with target *here*. The output membrane is 2.

The tables of rules are given below:

$r_1 = \{x \rightarrow \begin{smallmatrix} x \\ x \end{smallmatrix}, z \rightarrow \begin{smallmatrix} z \\ z \end{smallmatrix}, y \rightarrow \begin{smallmatrix} y \\ y \end{smallmatrix}\}, r_2 = \{s \rightarrow \begin{smallmatrix} s \\ s \end{smallmatrix}, z \rightarrow \begin{smallmatrix} z \\ z \end{smallmatrix}, t \rightarrow \begin{smallmatrix} t \\ t \end{smallmatrix}\}, c_1 = \{d \rightarrow adb, z \rightarrow zzz\}, c_2 = \{e \rightarrow aeb, z \rightarrow zzz\}, c_3 = \{e \rightarrow ab, z \rightarrow zz\}, c_4 = \{d \rightarrow ab, z \rightarrow zz\}$. Application of the rules of the row tables r_1, r_2 will add the rows $xz^{4n}y$ and $sz^{4n}t$ independently any number of times above and below the row $qz^{4n}r$ with the array remaining in membrane 1. A computation starts with an application of the rules of the column table c_1 in membrane 1 to the axiom array, sending the generated array

$$\begin{array}{c} x a d b e y \\ x z z z z y \\ q z z z z r \\ s z z z z t \end{array}$$

to membrane 2. If the rules of the column table c_2 in membrane 2, are applied, then the generated array

$$\begin{array}{c} x a d b a e b y \\ x z z z z z z y \\ q z z z z z z r \\ s z z z z z z t \end{array}$$

will be sent back to membrane 1. This process can repeat. If in membrane 1, the rules of the column table c_3 (instead of c_1) are applied, then the generated array is sent to the inner membrane 2 where the rules of the column table c_4 are applied which replaces the symbol d by ab in the first row and replaces one symbol z in each of the remaining rows by zz . The resulting array is in the form of the picture array p , which remains in the output region 2 and the computation halts. Here it is collected in the language. Thus the picture language L is generated. Note that no other correct sequence of applications of the tables of rules is possible.

6 Generation of floor designs

As an application of the array P System model considered in Section 4, we use a well-known technique [18] developed to generate certain interesting classes of floor designs, called “kolam patterns” [18, 20]. The idea is that with each symbol

$$\begin{array}{c} a q q q y \\ s d d d t \\ s d d d t \\ s d d d t \\ x r r r e \end{array}$$

Fig. 2. A picture array of L_{kolam}

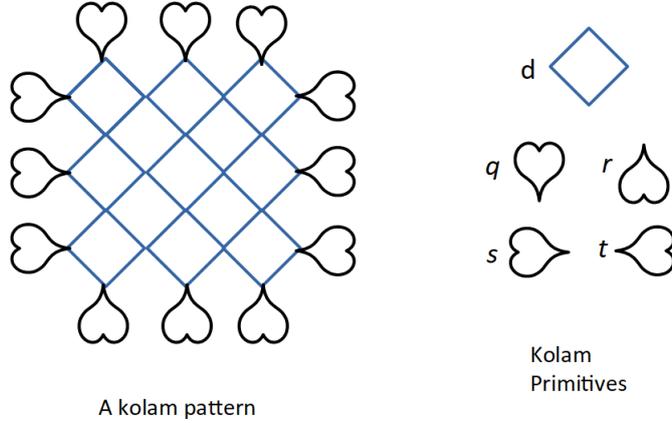


Fig. 3. A kolam pattern

a of a rectangular picture array which is considered to occupy a unit square in the rectangular grid, a primitive picture pattern $i(a)$ of the “kolam pattern” is associated. The picture array is then interpreted as a “kolam pattern” replacing the symbols in the picture array generated by the array P system, by the corresponding primitive picture patterns, to yield the required “kolam pattern”.

We illustrate this by considering the picture language L_{kolam} consisting of $n \times n$, $n \geq 3$ picture arrays p such that $p(1, 1) = a, p(1, n) = y, p(n, 1) = x, p(n, n) = e$, $p(1, j) = q, p(n, j) = r$, for $2 \leq j \leq n - 1$, $p(i, 1) = s, p(i, n) = t$, for $2 \leq i \leq n - 1$, $p(i, j) = d$, otherwise. A picture array of L_{kolam} is shown in Fig. 2 and the corresponding “kolam pattern” with the “kolam” primitives used, is shown in Fig. 3. L_{kolam} is generated by an array P system as in Definition 2 with membrane structure $[1 [2]_2]_1$, two column tables c_1, c_2 in region 1 and a row table r in region 2 given by $c_1 = \{a \rightarrow aq, d \rightarrow dd, b \rightarrow rb\}$, $c_2 = \{a \rightarrow a, d \rightarrow d, b \rightarrow e\}$, $r = \left\{ a \rightarrow \begin{matrix} a \\ s \end{matrix}, d \rightarrow \begin{matrix} d \\ d \end{matrix}, b \rightarrow \begin{matrix} t \\ b \end{matrix} \right\}$ with c_1, c_2 having target *in* and r having target *out*. The only initial axiom array in membrane 1, is

$$M_0 = \begin{matrix} a & q & y \\ s & d & t \\ x & r & b \end{matrix}.$$

The primitive picture patterns associated with q, r, s, t, d are shown in Fig. 3 and $i(a) = i(e) = i(x) = i(y) = blank$.

7 Conclusions and Open problems

We have introduced here a new rewriting mode in pure 2D context-free grammars, called independent mode, for the generation of certain picture array languages, inspired from two-dimensional insertion systems with independent mode considered in [7]. We have shown the incomparability of the two classes of array languages, namely *P2DCFL* [23, 24] and *IP2DCFL*, introduced here. Using P systems as a control mechanism for array rewriting, we have generated square pictures of a certain type (Example 3) using only one membrane and target agreement for rules. We also sketch an application of this formalism of *P2DCFG* with *independent* mode to the generation of certain simple floor designs, known as “kolam” patterns. It will be of interest to compare the array models considered here with the Chomsky type of array grammars [3, 5, 6].

Acknowledgements

The authors thank the referees for their useful review comments which helped to improve the presentation of the paper.

References

1. Bersani, M. M., Frigeri, A., Cherubini, A.: On some classes of 2D languages and their relations. In: Aggarwal, J.K., et al. (eds.) *Combinatorial Image Analysis. Lecture Notes Comput. Sci.*, vol. **4958** 222–234 (2011).
2. Bersani, M. M., Frigeri, A., Cherubini, A.: Expressiveness and complexity of regular pure two-dimensional context-free languages. *Int. J. Comput. Math.* **90** 1708–1733 (2013).
3. Ceterchi, R., Mutyam, M., Păun, Gh., Subramanian, K.G. : Array-rewriting P systems. *Natural Computing*, **2** 229–249 (2003).
4. Ceterchi R., Subramanian K.G., Venkat I. : P Systems with Parallel Rewriting for Chain Code Picture Languages. In: Beckmann A., Mitrana V., Soskova M. (eds) *Evolving Computability. CiE 2015. Lecture Notes in Computer Science*, vol **9136**. Springer, Cham. 145–155 (2015).
5. Freund, R. : Control mechanisms on #-context-free array grammars, In: *Mathematical Aspects of Natural and Formal Languages* (Gh. Păun, ed.), World Scientific, Singapore, 97137 (1994).
6. Freund, R. : *Array Grammars*, Technical Rep. 15/00, Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, 164 pages (2000).
7. Fujioka, K. : A two-dimensional extension of insertion systems. A.-H. Dediu et al. (Eds.): *TPNC 2014, Lecture Notes Comp. Sci.* **8890** 181-192 (2014).
8. Giammarresi, D., Restivo, A.: Two-dimensional languages, In: *Handbook of Formal Languages. Vol.3*, Eds. G. Rozenberg and A. Salomaa, Springer Verlag, 215–267 (1997).
9. Krivka, Z., Martín-Vide, C., Meduna, A., Subramanian, K.G.: A variant of pure two-dimensional context-free grammars generating picture languages. In: Barneva, R.P., Brimkov, V.E., Slapal, J. (eds.): *Combinatorial Image Analysis. LNCS*, vol. **8466**, Springer, Heidelberg (2014) 123–133.

10. Maurer, H.A., Salomaa, A., Wood, D.: Pure Grammars. *Inform. Control*, **44** 47–72 (1980).
11. Păun, Gh. : Computing with membranes. *J. Comp. System Sci.* **61** 108–143 (2000).
12. Păun, Gh.: *Membrane Computing: An Introduction*. Springer-Verlag Berlin, Heidelberg. (2000).
13. Păun, Gh., Rozenberg, G., Salomaa, A.: *The Oxford Handbook of Membrane Computing*. Oxford University Press. New York, USA, (2010).
14. Rosenfeld, A.: *Picture Languages - Formal Models for Picture Recognition*. Academic Press, New York, (1979).
15. Rosenfeld, A. and Siromoney, R.: Picture languages - a survey. *Languages of design*, **1**, 229–245 (1993).
16. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages*. Vol. 1 – 3, Springer, Berlin, (1997).
17. Salomaa, A.: *Formal languages*. Academic Press, London, (1973).
18. Siromoney, G., Siromoney, R., Krithivasan, K. : Array Grammars and kolam. *Comput. Graphics Image Processing* **3(1)** 63–82 (1974).
19. Siromoney, R. : Contributions of Professor Gift Siromoney in the Area of Pattern Recognition. *IETE Journal of Research*, **37(5–6)** (1991).
20. Siromoney, G., Siromoney, R. , Krithivasan, K.: Abstract families of matrices and picture languages. *Comput. Graphics Image Processing*, **1**, 234–307 (1972).
21. Siromoney, R., Subramanian, K.G. , Rangarajan, K.: Parallel/Sequential rectangular arrays with tables. *Int. J. Computer Mathematics*, 143–158 (1977).
22. Subramanian, K.G. : P systems and picture languages. *Lecture Notes in Comp. Sci.* **4664** 99–109 (2007).
23. Subramanian, K.G., Ali, R.M., Geethalakshmi, M., Nagar, A.K.: Pure 2D picture grammars and languages. *Discrete Appl. Math.* **157(16)** 3401–3411 (2009).
24. Subramanian, K.G., Nagar, A.K., Geethalakshmi, M.: Pure 2D picture grammars (P2DPG) and P2DPG with regular control. In: Brimkov, et al. (eds.), *Combinatorial Image Analysis*. LNCS, vol. **4958** 330–341 (2008).
25. Subramanian, K.G., Geethalakshmi, M., David, N.G.: Nagar, A.K. : Picture array generation using pure 2D context-free grammar rules. *Lecture Notes Comput. Sci.* **9448** (2015) 187–201.
26. Subramanian, K.G., Pan, L., Lee, S.K., Nagar, A.K. : A P system model with pure context-free rules for picture array generation. *Math. Comp. Modelling*, **52** 1901–1909 (2010).
27. Wang, P.S.P.: *Array grammars, Patterns and recognizers*. World Scientific, (1989).
28. Zhang, G., Pérez-Jiménez, M.J., Păun, Gh.: Real-life Applications with Membrane Computing. In: *Emergence, Complexity and Computation Series*. (2017).

Double Fusion of multiple graphs for Multi-view Clustering based on tissue-like P System

Huijian Chen¹ and Xiyu Liu² *

¹ Business School, Shandong Normal University, Jinan, China

² Business School, Shandong Normal University, Jinan, China

Abstract. Multi-view clustering has recently been studied by many scholars because it is generally better than single-view clustering in clustering performance. Multi-view subspace clustering and multi-view graph clustering show good clustering performance, but there is a lack of research on the integration of the two types of clustering methods. In order to more effectively reduce the analysis of noise and outliers and improve the clustering performance, we propose a new idea, that is, to perform multi-view clustering in a double fusion manner. Therefore, this paper proposes a novel multi-view clustering method, called Double Fusion of multiple graphs for Multi-view Clustering based on tissue-like P System (DFGMC-P). In the first fusion process, DFGMC-P treats each data point as a linear combination of other data, and constructs the affinity matrixes in a self-representation way. In the second fusion process, the optimized affinity matrixes obtained in the first fusion process is fused for the second time using a graph-based method. In addition, in this paper, we combine the model of this paper with the Tissue-like P system, so that the computational parallelism of the tissue-like P system can improve the computational efficiency of the algorithm. We conducted a comparative experiment on the single-view clustering method and the multi-view clustering method on three data sets, and the clustering performance of DFGMC-P algorithm is relatively good.

Keywords: Multi-view clustering, multi-view subspace clustering, multi-view graph clustering, Tissue-like P system.

1 Introduction

Membrane computing is also called P system[1], which is a branch of natural computing. It was originally proposed by PG[2], a member of the European Academy of Sciences and a member of the Romanian Academy of Sciences, in 1998. Membrane computing is established by the inspiration of the structure and function of biological cells, and has been widely concerned and researched. At present, there are roughly three types of membrane computing models: cell-like membrane system, tissue-like membrane system and neuro-like membrane system. The membrane system mainly includes membrane structure, objects and evolutionary rules. In the calculation process, the evolution rules are selected indeterministically and synchronously to move the objects[3].

* Xiyu Liu.

Membrane computing has the advantage of parallel computing. Therefore, the membrane computing can be combined with other algorithms to improve the efficiency of other algorithms. For example, the algorithm SCBK-CP[4], which combines membrane computing and spectral clustering by Liu et al., effectively improves the efficiency of the original algorithm and reduces the complexity of the algorithm.

Clustering is a basic technique of machine learning and data mining. Many clustering algorithms have been proposed and studied so far, such as k-means algorithm[5], spectral clustering[6-8], graph clustering[9], subspace clustering[10], etc. These are single-view clustering algorithms. With the development of science and technology, there is more and more multi-view data. More and more scholars have joined the trend of studying multi-view clustering[11, 12]. Due to the good clustering performance, multi-view subspace clustering[13, 14] and multi-view graph clustering[15, 16] have been extensively studied. Multi-view subspace clustering shows good clustering performance in processing high-dimensional data. It can map high-dimensional data to low-dimensional subspace and reconstruct data in the subspace to achieve better clustering performance. Graph-based multi-view clustering method is also one of the most popular multi-view clustering methods. The graph-based multi-view clustering method first constructs the affinity matrix of each view, then merges each view into a unified matrix, and finally applies additional clustering algorithms or other methods to the unified matrix to obtain the clustering results[12]. Whether it is multi-view subspace clustering or multi-view graph clustering, only one fusion operation for data. In this way, the presence of noisy data will lead to poor clustering performance. At the same time, for multi-view subspace clustering, the clustering results need to be obtained separately. Therefore, in this paper, we propose Double Fusion of multiple graphs for Multi-view Clustering based on tissue-like P System (DFGMC-P). DFGMC-P integrates the multi-view subspace clustering method and the multi-view graph clustering method to achieve two fusion operations. And the model of this paper is combined with the tissue-like P system to effectively improve the calculation efficiency. Figure 1 shows the membrane structure of DFGMC-P. The contributions of our work are listed as follows

- In order to assign weights to each view more reasonably, we propose the Double Fusion of multiple graphs for Multi-view Clustering based on tissue-like P System (DFGMC-P) method. Two self-weighting fusion processes are carried out for each view. Each process assigns appropriate weights to each view. After the two fusion processes, the weight distribution of each view will be more reasonable.
- The parallel computing feature of the tissue-like P system can significantly improve the computational efficiency of the algorithm, so we combine the multi-view clustering method proposed in this paper with the tissue-like P system to improve the computational efficiency of the DFGMC-P algorithm.

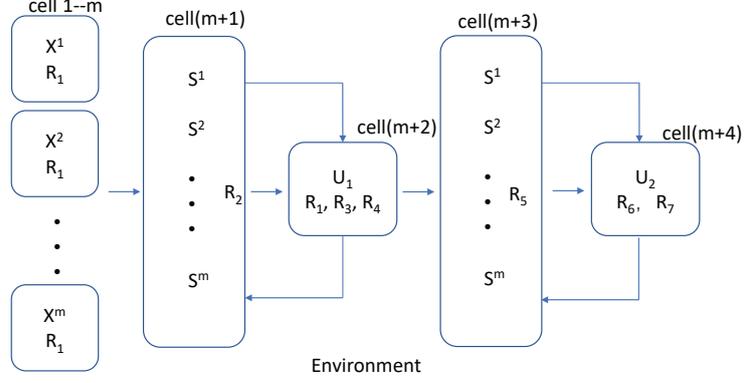


Fig. 1. The specific process of DFGMC-P

2 Related work

2.1 Multi-view subspace clustering

Multi-view subspace clustering is a multi-view clustering method for processing high-dimensional data. The self-representation method is often used for subspace clustering, which treats data points as a linear combination of the data itself. Liu et al.[17] applied the self-representation method to multi-view subspace clustering and showed good clustering performance. Liu et al.[18] proposed the GFSC method, whose objective function is

$$\min_{\mathbf{S}^v, \mathbf{U}_1, \mathbf{F}_{\mathbf{U}_1}} \sum_{v=1}^m \|\mathbf{X}^v - \mathbf{X}^v \mathbf{S}^v\|_F^2 + \alpha \|\mathbf{S}^v\|_F^2 + \beta w_v \|\mathbf{S}^v - \mathbf{U}_1\|_F^2 + \gamma \text{Tr}(\mathbf{F}_{\mathbf{U}_1}^T \mathbf{L}_{\mathbf{U}_1} \mathbf{F}_{\mathbf{U}_1}) \quad (1)$$

$$\text{s.t. } \mathbf{S}^v \geq 0, \mathbf{F}_{\mathbf{U}_1}^T \mathbf{F}_{\mathbf{U}_1} = \mathbf{I}$$

where

$$w_v = \frac{1}{2 \|\mathbf{S}^v - \mathbf{U}_1\|_F} \quad (2)$$

where \mathbf{X}^v is the original data matrix of the v -th view. \mathbf{S}^v is the affinity matrix of the v -th view. \mathbf{U}_1 is the unified matrix after fusion. $\mathbf{L}_{\mathbf{U}_1}$ is the Laplacian matrix of \mathbf{U}_1 . $\mathbf{F}_{\mathbf{U}_1}$ is the spectral embedding matrix, which is composed of eigenvectors corresponding to the first k smallest eigenvalues of $\mathbf{L}_{\mathbf{U}_1}$. α, β, γ are regularization parameters. w_v is the weight of the v th view.

2.2 Multi-view graph clustering

The graph-based multi-view clustering method first constructs the affinity matrix of each view, then merges each view into a unified matrix, and finally applies additional

clustering algorithms or other methods to the unified matrix to obtain the clustering results. The SwMC method proposed by Nie et al.[19] removes hyperparameters and has good clustering performance. The objective function is

$$\min_{\mathbf{U}_2, \mathbf{F}_{\mathbf{U}_2}} \sum_{v=1}^m \alpha^v \|\mathbf{U}_2 - \mathbf{S}^{(v)}\|_F^2 + 2\lambda \text{Tr}(\mathbf{F}_{\mathbf{U}_2}^T \mathbf{L}_{\mathbf{U}_2} \mathbf{F}_{\mathbf{U}_2}) \quad (3)$$

$$\text{s.t. } u_{2ij} \geq 0, u_i \mathbf{1}_n = 1, \mathbf{F}_{\mathbf{U}_2} \in R^{n \times c}, \mathbf{F}_{\mathbf{U}_2}^T \mathbf{F}_{\mathbf{U}_2} = \mathbf{I}$$

where

$$\alpha^v = \frac{1}{2 \|\mathbf{U}_2 - \mathbf{S}^{(v)}\|_F} \quad (4)$$

where α^v is the weight of the v -th view. \mathbf{U}_2 is the unified matrix after fusion. $\mathbf{L}_{\mathbf{U}_2}$ is the Laplacian matrix of the unified matrix \mathbf{U}_2 . $\mathbf{F}_{\mathbf{U}_2}$ is composed of eigenvectors corresponding to the first k eigenvalues of the Laplacian matrix of the unified matrix \mathbf{U}_2 . It is obvious that the model removes the hyperparameters that need to be set in advance.

2.3 The tissue-like P system

The tissue-like P system is related to the graph structure. The nodes of the graph correspond to the cells and the environment in the tissue membrane system, and the edges of the graph represent the communication channels between cells. The tissue-like P system performs calculations in each cell, and transmits objects between cells and between cells and the environment. We define the tissue-like P system of this paper as follows

$$\Pi = (O, K, \omega_1, \dots, \omega_m, E, ch, (s_{(i,j)})_{(i,j) \in ch}, (R_{(i,j)})_{(i,j) \in ch}, i_0)$$

O represents a finite set of objects ; K represents the states of the alphabet; $\omega_i, 1 \leq i \leq m$ represents the finite multiset of objects in the initial state of cells ; $E \subseteq O$ represents a copy of any number of symbolic objects in the environment; $ch \subseteq \{(i, j) | i, j \in \{0, 1, \dots, m\}, i \neq j\}$ represents the communication channel between cells and cells and between cells and the environment; $s_{(i,j)}$ is the initial state of the channel (i, j) ; $R_{(i,j)}$ is a finite co/inverse transportation rule of the form $(s, x/y, s')$, where $s, s' \in K$, $x, y \in O^*$; $i_0 \in \{1, \dots, m\}$ is the output cell.

3 DFGMC-P

Our DFGMC-P method integrates multi-view subspace clustering and multi-view subspace clustering, and performs two fusion operations. We will solve problem (1) and problem (2) to obtain the optimized \mathbf{S}^v , and then use the optimized \mathbf{S}^v as the input to problem (3). First, we optimize the problem (1) and problem (3).

3.1 Optimization of problem (1)

Reference[18] shows the detailed optimization process. Updating \mathbf{S}^v when $\mathbf{F}_{\mathbf{U}_1}$ and \mathbf{U}_1 are fixed. After removing the extraneous variables, we derive the derivative of \mathbf{S}^v in equation (1), and set the derivative to zero, then the expression for updating \mathbf{S}^v is

$$\mathbf{S}^v = ((\mathbf{X}^v)^T \mathbf{X}^v + \alpha \mathbf{I} + \beta w_v \mathbf{I})^{-1} (\beta w_v \mathbf{U}_1 + (\mathbf{X}^v)^T \mathbf{X}^v) \quad (5)$$

Where \mathbf{I} is the identity matrix. Updating \mathbf{U}_1 when $\mathbf{F}_{\mathbf{U}_1}$ and \mathbf{S}^v are fixed. After deriving, we can obtain

$$\mathbf{U}_1(:, i) = \frac{\sum_v w_v \mathbf{S}^v(:, i) - \frac{\gamma \mathbf{q}_i}{4\beta}}{\sum_v w_v} \quad (6)$$

where $\mathbf{q}_i \in R^{n \times 1}$ with the j -th entry $q_{ij} = \|\mathbf{F}_{\mathbf{U}_1 i, :} - \mathbf{F}_{\mathbf{U}_1 j, :}\|^2$. Updating $\mathbf{F}_{\mathbf{U}_1}$ when \mathbf{S}^v and \mathbf{U}_1 are fixed. The objective function becomes

$$\min_{\mathbf{F}_{\mathbf{U}_1}} Tr(\mathbf{F}_{\mathbf{U}_1}^T \mathbf{L}_{\mathbf{U}_1} \mathbf{F}_{\mathbf{U}_1}) \quad (7)$$

$$\text{s.t. } \mathbf{F}_{\mathbf{U}_1}^T \mathbf{F}_{\mathbf{U}_1} = \mathbf{I}$$

This is an iterative optimization operation. We stop when the number of iterations is greater than 200 or the relative change of \mathbf{U}_1 is less than 0.001.

3.2 Optimization of problem (3)

So far we can get the optimized \mathbf{S}^v of the first fusion. In the next step, we use the optimized \mathbf{S}^v as the input of equation (3). Updating $\mathbf{F}_{\mathbf{U}_2}$ when \mathbf{U}_2 , $\alpha^{(v)}$ is fixed. The objective function becomes

$$\min_{\mathbf{F}_{\mathbf{U}_2} \in R^{n \times c}, \mathbf{F}_{\mathbf{U}_2}^T \mathbf{F}_{\mathbf{U}_2} = \mathbf{I}} Tr(\mathbf{F}_{\mathbf{U}_2}^T \mathbf{L}_{\mathbf{U}_2} \mathbf{F}_{\mathbf{U}_2}) \quad (8)$$

Updating \mathbf{U}_2 when $\mathbf{F}_{\mathbf{U}_2}$, $\alpha^{(v)}$ is fixed. equation (3) becomes

$$\min_{u_{2ij} \geq 0, u_{2i} \mathbf{1}_n = 1} \sum_{v=1}^m \alpha^v \sum_{i,j=1}^n (u_{2ij} - s_{ij}^{(v)})^2 + \lambda \sum_{i,j=1}^n \|\mathbf{f}_{\mathbf{U}_2 i} - \mathbf{f}_{\mathbf{U}_2 j}\|_2^2 u_{2ij} \quad (9)$$

It is obvious that equation (9) is independent for each i , so we consider each i separately

$$\min_{u_{2ij} \geq 0, u_{2i} \mathbf{1}_n = 1} \sum_{j=1}^n \sum_{v=1}^m \alpha^v (u_{2ij} - s_{ij}^{(v)})^2 + \lambda \sum_{j=1}^n \|\mathbf{f}_{\mathbf{U}_2 i} - \mathbf{f}_{\mathbf{U}_2 j}\|_2^2 u_{2ij} \quad (10)$$

We denote $r_{ij} = \|\mathbf{f}_{\mathbf{U}_2 i} - \mathbf{f}_{\mathbf{U}_2 j}\|_2^2$ for the purpose of avoiding equation (10) that are too complicated. So equation (10) can be written in vector form as

$$\min_{u_{2ij} \geq 0, u_{2i} \mathbf{1}_n = 1} \|\mathbf{u}_{2i} - (\sum_{v=1}^m \alpha^v \mathbf{s}_i^{(v)} - \frac{\lambda}{2} \mathbf{r}_i) / \sum_{v=1}^m \alpha^v\|_2^2 \quad (11)$$

The problem can be solved by an iterative algorithm, which shows the process of the second fusion.

3.3 The initial configuration of the tissue-like P system

Before we perform calculations, we set up the initial configuration of the tissue-like P system proposed in this paper. The initial configuration of each cell and environment is as follows

cell $i, 1 \leq i \leq m$: $\omega_i = \mathbf{X}^i, w_i, \mathbf{F}_{U_1}, \mathbf{U}_1$; $R_1 = \text{Eq. (5)}$, $R_{10} = \alpha, \beta, \mathbf{I} \mid]_{1-m} \rightarrow [\alpha, \beta, \mathbf{I}]_{1-m}$.

cell (m+1): $\omega_{m+1} = w_1, \dots, w_m, \mathbf{F}_{U_1}$; $R_2 = \text{Eq. (6)}$, $R_{20} = \beta \mid]_{m+1} \rightarrow [\beta]_{m+1}$.

cell (m+2): $\omega_{m+2} = \mathbf{X}^1, \dots, \mathbf{X}^m, w_1, \dots, w_m$; $R_1 = \text{Eq. (5)}$, $R_{10}, R_3 = \text{Eq. (2)}$, $R_4 = \text{Eq. (7)}$, $R_{40} = [\mathbf{S}^1, \dots, \mathbf{S}^m]_{m+2} \rightarrow [\mathbf{S}^1, \dots, \mathbf{S}^m]_{m+3}$, when the relative change of \mathbf{U}_1 is less than 0.001.

cell (m+3): $\omega_{m+3} = \alpha^1, \dots, \alpha^m, \mathbf{F}_{U_2}$; $R_5 = \text{Eq. (11)}$.

cell (m+4): $R_6 = \text{Eq. (8)}$, $R_7 = \text{Eq. (4)}$, $R_{70} = [\mathbf{U}_2]_{m+4} \rightarrow []_{m+4} \mathbf{U}_2$ when the relative change of Eq. (3) does not exceed 10^{-8} .

Environment $E = \alpha, \beta, \gamma, \mathbf{I}$.

Figure 2 shows the initial configuration of the tissue-like P system.

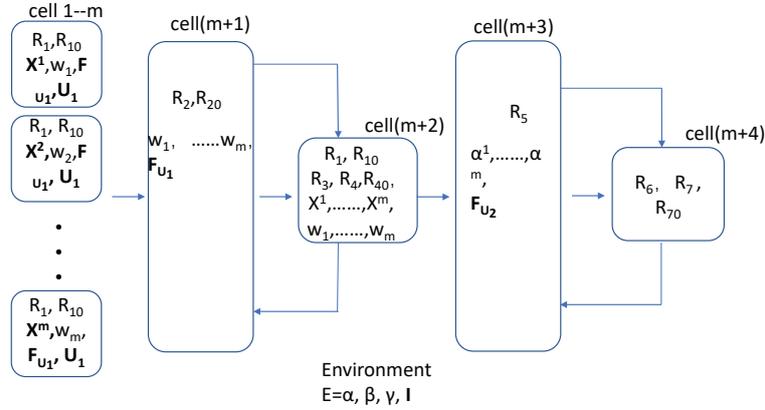


Fig. 2. The initial configuration of the tissue-like P system

3.4 Calculation process

In cells 1 to m, we first simultaneously apply the rule R_{10} to transfer the copies of $\alpha, \beta, \mathbf{I}$ in the environment to the membrane, and then apply the rule R_1 to generate $\mathbf{S}^v, 1 \leq v \leq m$ and send it to the cell (m+1). Then in cell (m+1), the rule R_{20} is

applied to transfer the copy of β in the environment to the membrane, and then the rule R_2 is applied to generate \mathbf{U}_1 and send it to cell (m+2). In the next step, first apply the rule R_3 to generate the updated weight w_1, \dots, w_m , and then the rule R_4 is applied to generate the updated $\mathbf{F}_{\mathbf{U}_1}$. Next the rule R_{10} is applied to transfer the copies of $\alpha, \beta, \mathbf{I}$ in the environment to the cell (m+2), and then apply the rule R_1 to produce the updated $\mathbf{S}^v, 1 \leq v \leq m$ and send it to the cell (m+1). Then apply the rules in the cell (m+1) again. This is a cyclic operation. When the conditions of triggering rule R_{40} are met, rule R_{40} is triggered, and the updated $\mathbf{S}^v, 1 \leq v \leq m$ is generated and sent to the cell (m+3).

In the cell (m+3), after receiving the updated $\mathbf{S}^v, 1 \leq v \leq m$ from the cell (m+2), the rule R_5 is applied to generate \mathbf{U}_1 and send it to the cell (m+4). When the cell (m+4) receives the \mathbf{U}_1 sent from the cell (m+3), the rules R_6 and R_7 are applied to generate updated $\mathbf{F}_{\mathbf{U}_2}$ and $\alpha^1, \dots, \alpha^m$, which are sent back to the cell (m+3). Then apply rule R_5 in the cell (m+3) again. This is another cyclic operation. When the trigger condition of rule R_{70} is met, rule R_{70} is triggered, the calculation is terminated, and the updated \mathbf{U}_2 is output. The specific process is shown by Algorithm 1.

Algorithm 1 DFGMC-P

Input: Data matrices: $\mathbf{X}^1, \dots, \mathbf{X}^m$, parameters $\alpha > 0, \beta > 0, \gamma > 0, \mathbf{I}$.
Output: Unified matrix $\mathbf{U}_1, \mathbf{F}_{\mathbf{U}_1}, \mathbf{U}_2 \in R^{n \times n}$ with c connected components, $\mathbf{F}_{\mathbf{U}_2}$.
Initialize: Random matrices \mathbf{U}_1 and \mathbf{F} , $w_v = 1/m$, random $\alpha^v, 1 \leq v \leq m$.
1: cell 1:m: $R_{10}, R_1 : \mathbf{S}^v, 1 \leq v \leq m \rightarrow$ cell (m+1);
repeat
2: cell (m+1): $R_{20}, R_2 : \mathbf{U}_1 \rightarrow$ cell (m+2);
3: cell (m+2): $R_3 : w_1, \dots, w_m \rightarrow$ cell (m+1); $R_4 : \mathbf{F}_{\mathbf{U}_1} \rightarrow$ cell (m+1); $R_{10}, R_1 : \mathbf{S}^v, 1 \leq v \leq m \rightarrow$ cell (m+1).
until Rule R_{40} is triggered, $\mathbf{S}^v, 1 \leq v \leq m \rightarrow$ cell (m+3).
repeat
4: cell (m+3): $R_5 : \mathbf{U}_1 \rightarrow$ cell (m+4);
5: cell (m+4): $R_6 : \mathbf{F}_{\mathbf{U}_2} \rightarrow$ cell (m+3); $R_7 : \alpha^1, \dots, \alpha^m \rightarrow$ cell (m+3);
until Rule R_{70} is triggered. Output \mathbf{U}_1 .

4 Experiment

4.1 datasets

In order to verify the effectiveness of our proposed algorithm, we conducted comparative experiments on three UCI data sets, which are ORL, MSRC, HW. The specific information of the data set is shown in Table 1, where $di, 1 \leq i \leq m$ is the dimension of the i -th view, n is the number of sample points, and c is the number of clusters.

Table 1. The specific details of the three data sets ORL, MSRC, HW

	d1	d2	d3	d4	d5	d6	n	c
ORL	512	59	864	254	-	-	400	40
MSRC	1302	48	512	100	256	210	210	7
HW	216	76	64	6	240	47	2000	10

4.2 Experimental evaluation and results

The algorithms for comparison in this paper are Auto-weighted Multiple Graph Learning (AMGL)[20], Multi-view Concept Clustering (MVCC)[21], Deep Matrix Factorization based Solution (DMClusts)[22], Multi-graph Fusion for Multi-view Spectral Clustering (GFSC)[18]. In this paper, the indicators used to evaluate clustering performance are accuracy (Acc), normalized mutual information (NMI) and purity. The specific experimental results are shown in Table 2.

- It is obvious that our proposed DFGMC-P is superior to other multi-view clustering algorithms in terms of clustering performance. Among the compared algorithms, MVCC showed good clustering performance second only to DFGMC-P algorithm, which shows that concept clustering exhibits a good clustering effect in multi-view clustering.
- AMGL is a graph learning algorithm, and GFSC is a multi-view spectral clustering algorithm based on a self-representation method. It can be seen that the clustering performance of DFGMC-P algorithm is better than single multi-view graph clustering (AMGL) and multi-view subspace clustering algorithm (GFSC). This shows that the double fusion method is effective for multi-view clustering.
- Figure 3 shows the convergence rate of the second fusion of the DFGMC-P algorithm. It can be seen that the convergence rate of the algorithm is very fast in general. The algorithm has converged after only one iteration on the ORL and HW data sets, and it has been converged after about 17 iterations on the MSRC data set.

5 Conclusion

In order to more effectively reduce the analysis of noise and outliers and improve the clustering performance and computational efficiency, we propose Double Fusion of multiple graphs for Multi-view Clustering based on tissue-like P System (DFGMC-P). The DFGMC-P algorithm integrates multi-view subspace clustering and multi-view graph clustering, and performs two fusion operations. In the first fusion process, DFGMC-P treats each data point as a linear combination of other data, and constructs the affinity matrixes in a self-representation way. In the second fusion process, the optimized affinity matrixes obtained in the first fusion process is fused for the second time using a graph-based method. In addition, in order to improve the computational efficiency of

Table 2. Comparison of algorithms on three data sets for Acc, NMI, purity

	Acc			NMI			purity		
	ORL	MSRC	HW	ORL	MSRC	HW	ORL	MSRC	HW
AMGL	54.00	72.86	71.13	75.91	73.20	78.69	62.50	78.57	74.88
MVCC	67.75	75.71	84.11	84.35	65.25	78.81	72.25	75.71	85.00
DMClusts	41.75	46.71	50.93	63.66	32.81	46.20	47.20	50.71	55.58
GFSC	63.68	56.43	70.48	81.83	51.05	74.19	78.38	74.71	82.27
DFGMC-P	80.50	83.33	82.75	89.07	84.25	87.62	85.00	83.81	86.90

the DFGMC-P algorithm, we combine the multi-view clustering model with the tissue-like P System, and use the computational parallelism of the tissue-like P System to improve the computational efficiency of the DFGMC-P algorithm. Comparative experiments on the three data sets verify the superiority of the DFGMC-P algorithm over other algorithms.

6 Acknowledgment

This research project is supported by the National Natural Science Foundation of China (61876101, 61802234, 61806114), Social Science Fund Project of Shandong Province, China (16BGLJ06, 11CGLJ22), Natural Science Fund Project of Shandong Province, China (ZR2019QF007), Postdoctoral Project, China (2017M612339, 2018M642695), Humanities and Social Sciences Youth Fund of the Ministry of Education, China (19YJCZH244), Postdoctoral Special Funding Project, China (2019T120607).

References

1. Díaz-Pernil, D and Prez-Jimnez, MJ and Romero-Jimnez, Ivaro.: Efficient simulation of tissue-like P systems by transition cell-like P systems. *Natural Computing* **8**(4), 797-806 (2009)
2. Paun, G.:Computing with membranes.*Journal of Computer and System Sciences* **61**(1), 108-143 (2000)[Online]. Available: ;Go to ISI;://WOS:000089085800004.
3. Pan, L. Q. and Perez-Jimenez, M. J.:Computational complexity of tissue-like P systems.*Journal of Complexity* **26**(3), 296-315 (2010)[Online]. Available: ;Go to ISI;://WOS:000279996800005.
4. Zhang, Z. and Liu, X.:An Improved Spectral Clustering Algorithm Based on Cell-Like P System.*Human Centered Computing*. (2019)
5. Macqueen, J. B.:Some Methods for Classification and Analysis of Multivariate Observations. (1966)

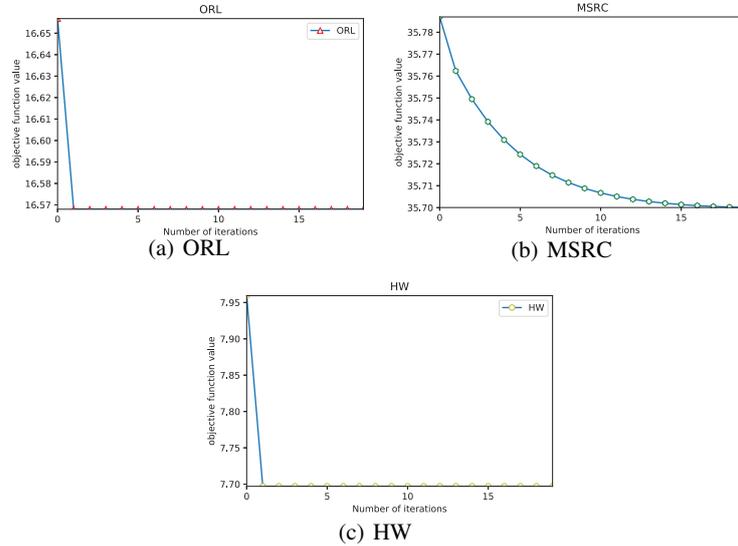


Fig. 3. The change of the objective function value of the second fusion process of the DFGMC-P algorithm on the three data sets.

6. Wang, Q. and Qin, Z. and Nie, F. and Li, X.:Spectral Embedded Adaptive Neighbors Clustering.Neural Networks and Learning Systems, IEEE Transactions on **30**(4), 1265-1271 (2019)
7. Wei, Z. and Nie, F. and Li, X.:Fast Spectral Clustering with efficient large graph construction.IEEE International Conference on Acoustics (2017)
8. Huang, D. and Wang, C. D. and Wu, Jiansheng and Lai, JH and Kwoh, C. K.:Ultra-Scalable Spectral Clustering and Ensemble Clustering.IEEE Transactions on Knowledge and Data Engineering. 1-1, (2019)
9. Nie, F. and Wang, X. and Jordan, M. I. and Huang, H.:The Constrained Laplacian Rank Algorithm for Graph-Based Clustering.AAAI Press (2016)
10. Luo, J. and Jiao, L. and Lozano, J. A.:A Sparse Spectral Clustering Framework via Multiobjective Evolutionary Algorithm.IEEE Transactions on Evolutionary Computation **20**(3), 418-433 (2016)
11. Liu, B. Y. and Huang, L. and Wang, C. D. and Lai, J. H. and Yu, P.:Multi-view Consensus Proximity Learning for Clustering.IEEE Transactions on Knowledge and Data Engineering **pp**(99), 1-1 (2020)
12. Wang, H. and Yang, Y. and Liu, B.:GMC: Graph-based Multi-view Clustering.IEEE Transactions on Knowledge and Data Engineering. 1-1 (2019)
13. Liu, X. L. and Pan, G. and Xie, M. Y.:Multi-view subspace clustering with adaptive locally consistent graph regularization.Neural Computing and Applications. (2021). [Online]. Available: ;Go to ISI;://WOS:000658070100003.
14. Lv, J. C. and Kang, Z. and Wang, B. Y. and Ji, L. P. and Xu, Z. L.:Multi-view subspace clustering via partition fusion.Information Sciences **560**, 410-423 (2021)[Online]. Available: ;Go to ISI;://WOS:000641000800005.
15. Jiang, G. Q. and Wang, H. B. and Peng, J. J. and Chen, D. Y. and Fu, X. P.:Graph-based Multi-view Binary Learning for image clustering.Neurocomputing **427**, 225-237 (2021)[Online]. Available: ;Go to ISI;://WOS:000611067800020.

16. Xiao, H. and Chen, Y. D. and Shi, X. D.: Knowledge Graph Embedding Based on Multi-View Clustering Framework. *Ieee Transactions on Knowledge and Data Engineering* **33**(2), 585-596 (2021)[Online]. Available: ;Go to ISI;://WOS:000607806200019.
17. Guo, J. P. and Yin, W. B. and Sun, Y. F. and Hu, Y. L.: Multi-View Subspace Clustering With Block Diagonal Representation. *Ieee Access* **7**, 84829-84838 (2019)[Online]. Available: ;Go to ISI;://WOS:000475803000001.
18. Kang, Z. and Shi, G. and Huang, S. and Chen, W. and Pu, X. and Zhou, J. T. and Xu, Z.: Multi-graph Fusion for Multi-view Spectral Clustering. *Knowledge-Based Systems*. (2019)
19. Nie, F and Jing, L. and Li, X.: Self-weighted Multiview Clustering with Multiple Graphs. *Twenty-Sixth International Joint Conference on Artificial Intelligence*. (2017)
20. Nie, F and Jing, L. and Li, X.: Parameter-free auto-weighted multiple graph learning: A framework for multiview clustering and semi-supervised classification. *AAAI Press*. (2016)
21. Hao, W. and Yan, Y. and Li, T.: Multi-view Clustering via Concept Factorization with Local Manifold Regularization. *IEEE International Conference on Data Mining (ICDM2016)*. (2017)
22. Wei, S. and Wang, J. and Yu, G. and Domeniconi, C. and Zhang, X.: Multi-View Multiple Clusterings Using Deep Matrix Factorization. *Proceedings of the AAAI Conference on Artificial Intelligence* **34**(4), 6348-6355 (2020)

Spiking Neural P Systems and Their Semantics in Haskell

Gabriel Ciobanu and Enea Nicolae Todoran

¹ Gabriel Ciobanu, A.I.Cuza University and Romanian Academy, Iași, Romania
`gabriel@info.uaic.ro`

² Enea Nicolae Todoran, Technical University, Cluj-Napoca, Romania
`enea.todoran@cs.utcluj.ro`

Abstract. We express spiking neural P systems and their semantics by using the functional programming language Haskell. This functional approach is realized by a semantic interpreter for a simple concurrent language able to describe the structure and behaviour of spiking neural P systems. The semantic interpreter is designed by using the method of denotational semantics. Haskell is a functional language providing excellent support for various techniques of the denotational semantics; we use fixed point semantics and continuations for concurrency. The semantic interpreter captures accurately the nondeterministic behaviour, the time delays between firings and spikings, and the synchronized functioning specific to spiking neural P systems.

1 Introduction

We address the problem of implementing spiking neural P systems [9] by using the functional programming language Haskell (<http://haskell.org/>). For designing such an implementation we use the tradition of programming languages semantics, and the discipline of denotational semantics [17, 19]. Haskell is a lazy purely functional language providing excellent support for implementing the techniques that are specific to denotational semantics. In this work we use fixed point semantics and continuations for concurrency [5, 21]. We design and implement a semantic interpreter for a (concurrent) language named \mathcal{L}_{snp} providing constructs that can be used to describe the structure and behaviour of Spiking Neural P (SN P) systems.

The language \mathcal{L}_{snp} used in this paper is similar to that presented in [7], where we presented a denotational semantics designed with metric spaces [1] for the behaviour of SN P systems. The language \mathcal{L}_{snp} provides constructs to describe neurons, synapses and rules with time delays that are specific to SN P systems. Here we emphasize on issues related to implementation, simulation and verification, describing a semantic interpreter which captures accurately the nondeterministic behaviour, the time delays between firings and spikings, and the synchronized functioning of the SN P systems.

In \mathcal{L}_{snp} a spike is represented as an elementary 'statement', and statements can be combined by using an operator for parallel composition. Intuitively, each

spike is 'executed' in the context of a neuron. In general, a statement describes a multiset of spikes which are executed concurrently. According to [20], "it is not necessary for the semantics to determine an implementation, but it should provide criteria for showing that an implementation is correct". In order to provide support for automated verification, the nondeterministic systems are modelled in our implementation taking into account all possible interactions in the behaviour of an SN P system. In denotational semantics, such a nondeterministic behaviour is described by using powerdomains [14]; since a specific simulation of an SN P system represents an execution trace, we work with a collections of execution traces implementing elements of a linear time powerdomain [1].

Implementation The semantic interpreter of language \mathcal{L}_{snp} described in this paper is implemented in Haskell, and it is available online at [22] in two variants contained in files `semSNP.hs` and `semSNP-fin.hs`. The file `semSNP.hs` implements a semantic interpreter producing all possible execution traces regardless of the length or number of execution traces; in the presence of nondeterminism, this interpreter can only be used to verify toy SN P systems. On the other hand, the semantic interpreter implemented in the file `semSNP-fin.hs` is designed to preserve only a finite prefix for each execution trace. Intuitively, each execution trace is stopped after a finite number of steps (no matter whether the system subsequently halts or not). Thus, the interpreter `semSNP-fin.hs` can be used to simulate and verify (bounded) executions of nondeterministic SN P systems.

2 SN P Systems and Their Semantics in Haskell

We develop software components able to provide automatic support for simulating and verifying SN P systems. For this purpose, we design a simple concurrent language called \mathcal{L}_{snp} capturing the basic notions specific to an SN P system: neurons, synapses and rules with time delays. We present the syntax of \mathcal{L}_{snp} , and explain informally the behaviour of its constructs. Both the syntax and the semantics of language \mathcal{L}_{snp} are implemented in the purely functional programming language Haskell providing excellent support for the techniques specific to denotational semantics.

In general, denotational semantics provide a meaning function $\mathcal{M} : \mathcal{L} \rightarrow \mathbf{D}$, where \mathcal{L} is a language and \mathbf{D} is a mathematical domain used to assign meanings to the language constructs. The main characteristic of such a semantics is its *compositionality*: the semantics of a composed construct is defined solely based on the semantics of its components. Compositionality is a nice property to have because it provides scalability and modularity.

In recent work we developed denotational semantic models for various kinds of P systems [6, 7]. This paper is mainly related to [7], where we have presented a denotational semantics designed with metric spaces for a language inspired by SN P systems similar to the language \mathcal{L}_{snp} investigated in this work. As in [7], we use a terminology that is specific to programming languages, similar to the terminology employed in the monograph [1]. The elements of language \mathcal{L}_{snp} are

syntactic constructions called *statements* and *programs*, and the terms *execution* and *evaluation* are used to describe their behaviour. Here, the method of denotational semantics is considered from an engineering viewpoint [17], emphasizing on implementation, simulation and verification. Rather than using the theory of domains [8, 18] or the theory of metric spaces [1], we use the functional programming language Haskell as an implementation tool and as a metalanguage for our denotational semantics.

Comparing the semantics presented in this paper and in [7], there exist the following differences. In both papers, the semantic models are designed with *continuations*; however, in [7] the continuations contain all information regarding declarations and concurrent control aspects, while in this paper we aimed to obtain a modular design based on the concepts of *semantic environment* and *continuation*. The information regarding declarations and the execution of rules specific to the SN P systems is contained in semantic environments. Also, the concurrent control aspects are described using the concepts of *synchronous continuation* and *asynchronous continuation*, fact which is not articulated in [7]. A simulator for SN P systems without delays implemented by using GPUs (Graphics Processing Units) is presented in [2]. On the other side, here we present SN P systems allowing delays between firings and spikings, proposing the denotational semantics as a general design and verification method for SN P systems.

Remark 1 *We use specific mathematical notations to describe the formal syntax of language \mathcal{L}_{snp} (Definition 2) and to explain Example 4 and Example 5. We present shortly some mathematical preliminaries (for sets, multisets and regular languages) as in our previous work [7]. We use the notation $(a, b \in)S$ to introduce the set S with typical elements a, b ranging over S . $\mathcal{P}_{fin}(S)$ denotes the powerset of all finite subsets of set S . We denote by $L(E)$ the language associated with a regular expression E . Also, we represent a multiset by enumerating its elements between brackets '[' and ']'. For example, [] is the empty multiset. A multiset is an unordered collection, e.g., $[a, a, b] = [a, b, a] = [b, a, a]$ is the multiset containing two occurrences of element a and one occurrence of element b . A multiset u is also written in the form $u = [a_1^{m_1}, \dots, a_n^{m_n}]$, where m_i is the multiplicity (number of occurrences) of element a_i in multiset u , for $i = 1, \dots, n$. For example, $[a, a, b] = [a^2, b^1]$. For more information on formal languages, see [16].*

In Definition 2 we present the formal syntax for language \mathcal{L}_{snp} by using BNF. The basic components are an alphabet $(a \in)O$ of *spikes* or *objects*, and a set $(N \in)Nname$ of *neuron names*. We use the set $(u \in)U$ of all finite multisets over O , and the set $(\xi \in)\Xi = \mathcal{P}_{fin}(Nname)$ whose elements ξ are finite sets of neuron names. To simplify Definition 2, we use the (semantic) notions of a set $\xi \in \Xi$ and a multiset $u \in U$ to represent some (syntactic) components for which the order of elements is irrelevant.³ Anyway, in this work both sets and multisets are implemented as Haskell lists.

Definition 2 (*Syntax of \mathcal{L}_{snp}*)

³ These components could be defined in BNF as pure syntactic entities, e.g., as lists.

- (a) (Statements) $s(\in S) ::= a \mid \text{init } \xi \mid \text{send } \xi a \mid s \parallel s$
 (b) (Rules) $rs(\in Rs) ::= r_\epsilon \mid r, rs$
 where $r(\in R) ::= E/u \rightarrow s; \vartheta \mid u \rightarrow \lambda$
 with E a regular expression over O , and $\vartheta \geq 0, \vartheta \in \mathbb{N}$
 (c) (Neuron declarations)
 $D(\in Decl) ::= d \mid d, D$
 where $d(\in NDecl) ::= \text{neuron } N \{ rs \mid \xi \}$
 (d) (Programs) $\rho(\in \mathcal{L}_{snp}) ::= D, s$

An element $rs \in Rs$ is a list of rules. A construction $E/u \rightarrow x; \vartheta$ is called a firing rule. A construction $u' \rightarrow \lambda$ is called a forgetting rule. A list of rules $rs \in Rs$ is valid only if for any pair of (forgetting and firing) rules $u' \rightarrow \lambda$ and $E/u \rightarrow s; \vartheta$ contained in rs the following condition is satisfied: $\neg(u' \in L(E))$. Here $L(E)$ is the language associated with the regular expression E , and we use the notation $u' \in L(E)$ to express that there is a permutation of multiset u' which is an element of the language $L(E)$ (\neg is the logical negation operator). Note that the multiset u occurring on the left-hand side of a firing or a forgetting must be nonempty, i.e., $u \neq []$. Usually we omit a terminating r_ϵ , and write a non-empty list of rules $rs \in Rs$, $rs = r_1, \dots, r_i, r_\epsilon$, as $rs = r_1, \dots, r_i$. Also, for a list of declarations $D = \text{neuron } N_0 \{ rs_0 \mid \xi_0 \}, \dots, \text{neuron } N_n \{ rs_n \mid \xi_n \}$ to be valid the neuron names N_0, \dots, N_n must be pairwise distinct and the name of the first neuron in the list must be N_0 (the name N_0 is special, it is reserved).

Apart from minor notational differences, the definitions for the set of programs $\rho \in \mathcal{L}_{snp}$, the set of declarations $D \in Decl$ and the set of rules $rs \in Rs$ remain as in [7]. A neuron declaration is a construction $\text{neuron } N \{ rs \mid \xi \}$, where $N \in Nname$ is a neuron name, $rs \in Rs$ is a list of rules and $\xi \in \Xi$ is a finite set containing the names of the neurons that are connected by synapses (are adjacent or neighbouring) to neuron with name N . Intuitively, each statement $s \in S$ is executed by (or in the context of) a neuron.

A program $\rho \in \mathcal{L}_{snp}$ is a pair $\rho = (D, s)$, where $D = \text{neuron } N_0 \{ rs_0 \mid \xi_0 \}, \dots, \text{neuron } N_n \{ rs_n \mid \xi_n \}$ is a declaration (a list of neuron declarations), and $s \in S$ is a statement. The execution of program $\rho = (D, s)$ starts by executing statement s in the context of neuron with name N_0 (i.e., statement s is executed in the context of the neuron whose declaration occurs on the first position in the list of declarations D). In all other cases, a statement $s \in S$ is executed in the context of a neuron $\text{neuron } N \{ rs \mid \xi \}$ if s occurs in the right-hand side of a rule occurring in the list rs .

As in the original SN P systems model [9], a neuron may be in one of the following two states: *open* or *closed*. A neuron only accepts spikes in the open status. Essentially, *firing* and *forgetting* rules are handled as in [9]. An \mathcal{L}_{snp} firing rule is a construct $E/u \rightarrow s; \vartheta$, where E is a regular expression, $u \in U$ is a multiset, $s \in S$ is a statement, and $\vartheta \in \mathbb{N}$ is a natural number denoting a time interval. A statement s denotes a multiset of spikes that are executed concurrently. Hence, a firing rule $E/u \rightarrow s; \vartheta$ is similar to an extended SN P firing rule allowing for multiple types of spikes [10, 12].

At any moment, each neuron contains a (possibly empty) multiset of spikes. Let $\text{neuron } N \{rs_N \mid \xi_N\}$ be a neuron that contains in its list of rules rs_N a firing rule $r = E/u \rightarrow s; \vartheta$ and currently stores a multiset of spikes u_N , such that $u_N \in L(E)$ and $u \subseteq u_N$ (the notation $u_N \in L(E)$ is described Definition 2). In this case, neuron N can *fire* (execute) the rule r meaning the following: the multiset u is consumed (in neuron N remains the multiset of spikes $(u_N \setminus u)$) and the execution of statement s is triggered after exactly ϑ time units (s is suspended for the next ϑ time units). As in the original SN P systems, the neuron is closed (meaning that it cannot receive new spikes) in the time interval between firing and spiking; after this time interval elapses, the neuron becomes open (hence able to receive spikes) again. A forgetting rule $u \rightarrow \lambda$ executed by a neuron N removes the multiset $u \in U$ if it is executed in a state where the neuron N contains exactly the multiset u . Rules and the open/closed status are handled as in the original proposal [9], assuming a global clock for measuring time, each neuron operating in a nondeterministic sequential manner, with at most one rule applied in each step.

A statement $s \in S$ may be a spike $a \in O$, an initialization statement $\text{init } \xi$ (with $\xi \in \Xi$), a send statement $\text{send } \xi a$ (with $\xi \in \Xi$ and $a \in O$), or a parallel composition $s_1 \parallel s_2$ of two statements s_1 and s_2 .⁴ As in [7], when a spike $a \in O$ is executed by a neuron $\text{neuron } N \{rs_N \mid \xi_N\}$ the spike a is transmitted to all open neighbouring neurons with names in the set ξ_N . A statement $s \in S$ essentially denotes a multiset of spikes that are executed concurrently.

The constructs $\text{init } \xi$ and $\text{send } \xi a$ have an initialization effect with no counterpart in the original SN P systems. The execution of a program $\rho = (D, s)$, where $D = \text{neuron } N_0 \{rs_0 \mid \xi_0\}, \dots, \text{neuron } N_n \{rs_n \mid \xi_n\}$, with $D \in \text{Decl}$ and $s \in S$, starts by executing statement s in the context of neuron with name N_0 (by convention, the name N_0 is reserved to be used as the name of the neuron whose declaration occurs on the first position in any list of declarations $D \in \text{Decl}$). In the initial state only the neuron with name N_0 is *active*. The neuron with name N_0 is initialized automatically upon system start up as an *open* neuron containing an empty multiset of spikes. In the initial state all other neurons are *idle*, meaning that they cannot participate in interactions (they cannot send and cannot receive spikes), and in order to participate in interactions they must be initialized explicitly by using statements $\text{init } \xi$ and $\text{send } \xi a$.

When a statement $\text{init } \xi$ is executed by a neuron $\text{neuron } N \{rs_N \mid \xi_N\}$, each neighbouring neuron with name in the set $\xi \cap \xi_N$ (that have not been initialized previously) is initialized as an open neuron containing an empty multiset of spikes; here, $\xi \cap \xi_N$ is the set theoretic intersection of sets ξ and ξ_N . The statement $\text{init } \xi$ has no effect upon neurons that have been initialized beforehand. If the statement $\text{init } \xi$ is executed by a neuron $\text{neuron } N \{rs_N \mid \xi_N\}$ in a state where all neurons with names in the set $\xi \cap \xi_N$ have been initialized beforehand, then the statement $\text{init } \xi$ is inoperative.

⁴ The set of statements $(s \in S)$ is similar to the set of statements employed in [7], but there are also differences. The construct $\text{init } \xi$ is lacking from [7]. A version of the construct $\text{send } \xi a$ is provided in [7], but that version has no initialization effect.

The statement `send ξ a` has a similar initialization effect. When a statement `send ξ a` is executed by a neuron `neuron $N \{rs_N \mid \xi_N\}$` , each neighbouring neuron with name in the set $\xi \cap \xi_N$ that have not been initialized previously is initialized as an open neuron containing the multiset of spikes $[a]$ (multiset $[a]$ contains exactly one occurrence of spike a , and nothing else). If the statement `send ξ a` is executed by a neuron `neuron $N \{rs_N \mid \xi_N\}$` in a state where all neurons with names in the set $\xi \cap \xi_N$ have been initialized beforehand, then the statement `send ξ a` has no initialization effect, but it transmits the spike a to all open neurons with names in the set $\xi \cap \xi_N$. `send ξ a` is a send operation with target indication given by ξ [11]. Note that, spikes $a' \in O$ occurring outside the scope of any `send ξ a` statement are transmitted to the open neighbouring neurons, but have no initialization effect.

The execution of an \mathcal{L}_{snp} program involves an initialization phase with no counterpart in the original SN P systems presented in [9]. In the SN P model presented in [9], the initial multiset of objects contained in each neuron is part of the system specification. By contrast, an \mathcal{L}_{snp} program $\rho = (D, s)$ starts its execution with only one *active* neuron, namely the neuron with name N_0 , which is automatically initialized upon system start up as an open neuron containing an empty multiset of spikes (by convention, N_0 is the name of the first neuron occurring in any list of declarations $D \in Decl$). All other neurons are *idle* in the initial state, and must be initialized explicitly (by using statements `init ξ` and `send ξ a`) in order to participate in interactions.

When an \mathcal{L}_{snp} program $\rho = (D, s)$ starts its execution, the system initialization operation can be accomplished with the aid of statement $s \in S$, which is executed by the neuron with name N_0 . In case the neuron with name N_0 is connected (by synapses) to all other neurons, this initialization phase can be accomplished in a single step (one time unit). Apart from this initialization phase, an \mathcal{L}_{snp} program can describe accurately how an SN P system behaves.

In the rest of this section we describe the main components of a semantic interpreter for the language \mathcal{L}_{snp} . The semantic interpreter is available online at [22], in the file `semSNP.hs`.

```

type Spike = String
type U      = [Spike]
type Nname = String
type Xi    = [Nname]
    
```

The types `Spike` and `Nname` implement the sets O of spikes and $Nname$ of neuron names, respectively. The types `U` and `Xi` implement the sets U of multisets of spikes and Ξ of (finite) sets of neuron names, respectively. Sets and multisets are implemented as Haskell lists. The syntax of language \mathcal{L}_{snp} presented in Definition 2 can be implemented in Haskell as follows:

```

data S      = Spike Spike | Init Xi | Send Xi Spike | Par S S

data Rule   = FireR (RegExp Spike) U S Int | ForgetR U
    
```

```

type Rules = [Rule]
type NDecl = (Nname, Rules, Xi)
type Decl  = [NDecl]
type Prog  = (Decl, S)
    
```

The types `S`, `Rule`, `Rules`, `NDecl`, `Decl` and `Prog` implement the classes S of statements, R of rules, Rs of lists of rules, $NDecl$ of neuron declarations, $Decl$ of declarations and \mathcal{L}_{snp} of programs, respectively, introduced in Definition 2.

We omit here the definition of the type `RegExp` which we use to handle regular expressions defined over the alphabet of spikes `Spike`. For manipulating regular expressions we use a predicate (`elemRegExp u re`) which accepts as arguments a multiset `u` of spikes and a regular expression `re` and verifies whether any permutation of multiset `u` is an element of the language associated with regular expression `re`.

The final result of the semantic interpreter of language \mathcal{L}_{snp} is an element of the type `F`. The elements of type `F` represent *program answers*. In this section `F` is a synonym for type `P`, which implements a linear time *powerdomain* presented in [1]. Powerdomains are employed in denotational semantics to describe concurrent and nondeterministic behaviour. Nondeterministic systems are modelled in our Haskell implementation taking into account all possible interactions describing the behaviour of an SN P system.

```

type F    = P
type P    = [Q]
data Q    = Qe | Q Obs Q
data Obs  = Obs [(Nname, U)]
    
```

An element of the type `P` is a set (implemented as a list) of sequences of type `Q`. An element of type `Q` is a sequence of observables of type `Obs`. `Qe` is the empty sequence. An *observable* of type `Obs` is a set (implemented as a list) of pairs of the type `(Nname, U)`. We recall that `Nname` implements the set of neuron names, and `U` implements the set of all finite multisets of spikes. We use the type `Obs` to represent the current state of a spiking neural P system.

```

pref :: Obs -> P -> P
pref obs p = [ Q obs q | q <- p ]

ned :: P -> P -> P
ned p1 p2 =
  [ q | q <- p1 'setUnion' p2, q /= Qe ] 'setUnion'
  [ Qe | Qe <- p1 'setIntersect' p2 ]

bigned :: [P] -> P
bigned []      = [Qe]
bigned (p:ps) = p 'ned' (bigned ps)
    
```

The mapping `pref` implements the prefixing of an observable to a program answer. The mapping `ned` describes nondeterminism as a union of behaviours. Note

that `(ned p1 p2)` terminates only if both `p1` and `p2` terminate. The mapping `bigned` implements a nondeterministic choice between several alternatives.

```

fe :: F
fe = [Qe]

preff :: Obs -> F -> F
preff = pref

bignedf :: [F] -> F
bignedf = bigned
    
```

We use `fe = [Qe]` to represent termination or the reach of a halting configuration as a final program answer of type `fe :: F`. In this section `F` is a synonym for type `P`, and mappings `bignedf` and `preff` behave the same as mappings `bigned` and `pref`, respectively. In Section 3 the definition of type `F` is modified in order to obtain a different representation of nondeterministic behaviour.

We present now a (denotational) semantic function `den` which describes the behaviour of \mathcal{L}_{snp} statements in a compositional manner. In the definition of mapping `den` we use a set of *actions* implemented in Haskell by the type `Act`.

```

data Act = Act Spike Xi | ActInit Xi | ActSend Spike Xi
type MsetAct = [Act]
    
```

The elements of the type `Act` are used to define the semantics of the elementary statements `a`, `init ξ` and `send ξa` . The type `MsetAct` implements a set of *multisets of actions*. The type of the semantic function `den` is

```

den :: S -> E -> Xi -> D
    
```

The type `S` implements the class of \mathcal{L}_{snp} statements, `E` is a type of *semantic environments* and `Xi` is the type of sets (implemented as lists) of neuron names specified above. An element of type `D` is called a *computation*. The types `E` and `D` are defined below.

```

type E = Act -> D
    
```

A semantic environment of the type `E` is a function which maps an action of type `Act` to a computation of type `D`. In the definition of semantic function `den` we use a semantic environment defined as fixed point of a higher order mapping.

```

type D      = Cont -> MsetAct -> F

type Cont   = (C,K)
data C      = Ce | C D
type K      = [(Nname,NState)]
data NState = Open U | Closed U Int U D
    
```

The type of computations is designed by using the *continuation semantics for concurrency (CSC)* technique [5, 21]. A computation of type `D` is a function which receives as arguments a *continuation* of type `Cont` and a multiset of actions of type `MsetAct` and yields a final value of the type `F`. In the CSC approach a continuation is a structured configuration of computations which can be evaluated concurrently. In the implementation presented in this paper a continuation of type `Cont` is a pair (c, k) , where `c` is an element of type `C` and `k` is an element of type `K`. An element of type `C` is called a *synchronous continuation*. A synchronous continuation is a computation which executes the spikes that are transmitted between neurons in the current step.⁵ An element of the type `K` is called an *asynchronous continuation*. An asynchronous continuation is a list of pairs of type $[(Nname, NState)]$. An element of type `Nname` is a neuron name. An element of type `NState` describes the current *state* of a neuron. A neuron can be in one of the following two states: open or closed. The open and closed states are implemented by using the data constructors `Open` and `Closed`, respectively.

The techniques specific to denotational semantics can be conveniently implemented in Haskell. Since Haskell supports lazy evaluation, the fixed point combinator can be defined as follows:

```
fix :: (a -> a) -> a
fix f = f (fix f)
```

The definition of the semantic mapping `den` uses a semantic operator for parallel composition defined (in the style of denotational semantics) as the fixed point of a higher order mapping `hopar`.

```
type Op = D -> D -> D
hopar :: Op -> Op
hopar op d1 d2 = \ (c,k) m ->
    bignedf [d1 (lift op (C d2) c,k) m, d2 (lift op (C d1) c,k) m]

par = fix hopar
```

The operator `lift` is used to lift an operator on computations of type `D` to an operator on synchronous continuations of type `C`. The semantic interpreters available at [22] use the following definition:

```
lift :: Op -> C -> C -> C
lift op Ce Ce = Ce
lift op Ce f = f
lift op f Ce = f
lift op (C d1) (C d2) = C (op d1 d2)
```

⁵ The empty synchronous continuation `Ce` is used to handle steps where no spikes are emitted, e.g., because all neurons are closed and no neuron can move to the open status in the current step.

The denotational semantics `den` is a compositional function defined using the semantic operator for parallel composition `par` as follows:

```
den :: S -> E -> Xi -> D
den (Spike a)     env xi = env (Act a xi)
den (Send xi1 a) env xi = env (ActSend a (xi1 'setIntersect' xi))
den (Init xi1)   env xi = env (ActInit (xi1 'setIntersect' xi))
den (Par s1 s2)  env xi = (den s1 env xi) 'par' (den s2 env xi)
```

where `setIntersect` is the Haskell implementation of the set theoretic intersection operation.

Finally, the semantic interpreter is a mapping `denp` which receives as argument a program of the type `Prog` and yields a (final) value of the type `F`.

```
denp :: Prog -> F
denp (decl,s) = den s env0 xi0 (Ce,k0) []
  where env0           = fix (hoenv decl)
        ((n0,rs0,xi0):_) = decl
        k0              = [(n0,Open [])]
```

The semantic interpreter function `denp` uses a semantic environment `env0 :: E` which stores information about the neuron declarations and applies in a non-deterministic sequential manner the rules contained in declarations. The semantic environment `env0` is defined as the fixed point of a higher order mapping `hoenv`. The Haskell implementation of the mapping `hoenv` is available online.

We illustrate in Example 4 how to use the semantic interpreter described in this section. Further experiments are provided in Section 3, Example 5.

Remark 3 *In language \mathcal{L}_{snp} , neuron N_0 can be used in to receive the spikes emitted by the output neuron.⁶ This technique is also used in the design of \mathcal{L}_{snp} programs ρ_1 and ρ_3 , presented in Example 4 and Example 5, respectively, where the neuron with name N_0 receives the spikes emitted by the output neuron N_3 .*

As in [9], we consider that the result of a computation is the number of steps elapsed between the first two consecutive spikes produced by the output neuron (assuming the output neuron spikes at least twice, ignoring whether or not the calculation subsequently halts). This convention is used in the both examples (Example 4 and Example 5) presented in this paper.

Example 4 *We consider an \mathcal{L}_{snp} program ρ_1 implementing the spiking neural P system Π_1 given in [9] (Section 5, Figure 2). Before presenting the \mathcal{L}_{snp} program ρ_1 , we briefly describe the system Π_1 using the notation and terminology employed in [9]. As in [9], we use the following convention: when the language associated with regular expression E is $L(E) = \{a^i\}$, we write a firing rule $E/a^i \rightarrow a;t$ in the abbreviated form $a^i \rightarrow a;t$.*

⁶ In the original SN P systems this role is played by the *environment*; this notion should not be confused with the notion of a *semantic environment* (employed in this paper), which is specific to denotational semantics; see, e.g., [1].

Π_1 is a tuple $\Pi_1 = (\{a\}, \sigma_1, \sigma_2, \sigma_3, \{(1, 2), (2, 3)\}, 3)$, where $\{a\}$ is a singleton set (a is called a spike), $\sigma_1, \sigma_2, \sigma_3$ are neurons, $\text{syns} = \{(1, 2), (2, 3)\}$ is the set of synapses, and 3 is the index of the output neuron (σ_3 in this example). Neuron σ_1 is a pair $\sigma_1 = (2n - 1, \{a^+ / a \rightarrow a; 2\})$, where component $2n - 1$ is the initial number of spikes contained in σ_1 , and $\{a^+ / a \rightarrow a; 2\}$ is the set of rules describing the behaviour of neuron σ_1 . Neurons σ_2 and σ_3 behave as follows: $\sigma_2 = (0, \{a^n \rightarrow a; 1\})$ and $\sigma_3 = (1, \{a \rightarrow a; 0\})$. In the initial state, neurons σ_2 and σ_3 contain 0 and 1 spike, respectively. Each pair $(i, j) \in \text{syns}$ describes a synapse providing support for transmitting spikes from neuron σ_i to neuron σ_j . Two neurons (namely σ_1 and σ_3) fire in the first time unit. The output neuron σ_3 produces its first spike in step 1. Neuron σ_1 remains closed for two time units, and in the next step it produces a spike which is transmitted to neuron σ_2 . In $3n$ time units, neuron σ_2 receives n spikes from neuron σ_1 . Next, neuron σ_2 will fire in step $(3n + 1)$ and, after a delay of one time unit, it will send a spike to neuron σ_3 in step $(3n + 2)$. The output neuron σ_3 will release its second spike in step $(3n + 3)$. Hence, (using the convention presented in Remark 3) the number computed by system Π_1 is $3n + 2 = (3n + 3) - 1 = 3(n + 1) - 1$.

Following [9], we write a firing rule of the form $E/[a^i] \rightarrow s; \vartheta$ with $L(E) = \{a^i\}$ in the abbreviated form $[a^i] \rightarrow s; \vartheta$. Such a rule consumes all i spikes, when the neuron executing it contains exactly the multiset $[a^i]$. We recall that $[a^i]$ is the multiset containing i occurrences of the spike a . For example, $[a^3] = [a, a, a]$; see Remark 1. Also, we use the notation s^i , to represent i copies of statement s executed in parallel: $s^1 = s$ and $s^{i+1} = s \parallel s^i$, for $s \in S$ and $i \in \mathbb{N}$, $i > 0$.

Let $n \in \mathbb{N}, n > 0$. The program ρ_1 is given by $\rho_1 = (D_1, s_1)$, where the statement $s_1 \in S$ is $s_1 = (\text{send } \{N_1\} a)^{2n-1} \parallel \text{init } \{N_2\} \parallel \text{send } \{N_3\} a$ and the declaration $D_1 \in \text{Decl}$ is given by

$$\begin{aligned}
 D_1 = & \text{neuron } N_0 \{ r_\epsilon \mid \{N_1, N_2, N_3\} \}, \\
 & \text{neuron } N_1 \{ a^+ / [a] \rightarrow a; 2 \mid \{N_2\} \}, \\
 & \text{neuron } N_2 \{ [a^n] \rightarrow a; 1 \mid \{N_3\} \}, \\
 & \text{neuron } N_3 \{ [a] \rightarrow a; 0 \mid \{N_0\} \}.
 \end{aligned}$$

After the initialization step, neuron N_1 contains (a multiset with) $2n - 1$ spikes, neuron N_3 contains 1 spike, neuron N_2 contains 0 spikes, and neuron N_0 contains 0 spikes. N_3 is the output neuron. In our \mathcal{L}_{snp} program ρ_1 , neurons N_1, N_2 and N_3 implement the neurons σ_1, σ_2 and σ_3 , respectively, presented in the description of system Π_1 given in [9]. Apart from the initialization step (which is specific to \mathcal{L}_{snp}), the \mathcal{L}_{snp} program ρ_1 describes accurately the behaviour of system Π_1 presented in [9].

As explained in Remark 3, in our implementation neuron N_0 receives the spikes emitted by the output neuron N_3 , and the result is considered to be the number of steps elapsed between the first two consecutive spikes produced by the output neuron. The number computed by both system Π_1 presented in [9] and \mathcal{L}_{snp} program ρ_1 is $3n + 2$.

In the Haskell execution of the program ρ_1 we consider $n = 2$.⁷ In this case, the number of steps elapsed between the first and the second spike produced by output neuron N_3 is $8 = 3n + 2$, which coincides with the result obtained in [9] for the system Π_1 .

The semantic interpreter is available online at [22] in the file `semSNP.hs`; the \mathcal{L}_{snp} program ρ_1 is implemented and stored in variable `rho1 :: Prog`. Using the interpreter `semSNP.hs` to run this program with `(denp rho1)`, the following output is obtained:

```
[["N0", []], ("N3", ["a"]), ("N1", ["a", "a", "a"]), ("N2", [])] .
 [["N0", ["a"]], ("N3", []), ("N1", ["a", "a", "a"]), ("N2", [])] .
 [["N0", ["a"]], ("N3", []), ("N1", ["a", "a", "a"]), ("N2", [])] .
 [["N0", ["a"]], ("N3", []), ("N1", ["a", "a"]), ("N2", ["a"])] .
 [["N0", ["a"]], ("N3", []), ("N1", ["a", "a"]), ("N2", ["a"])] .
 [["N0", ["a"]], ("N3", []), ("N1", ["a", "a"]), ("N2", ["a"])] .
 [["N0", ["a"]], ("N3", []), ("N1", ["a"]), ("N2", ["a", "a"])] .
 [["N0", ["a"]], ("N3", []), ("N1", ["a"]), ("N2", ["a", "a"])] .
 [["N0", ["a"]], ("N3", ["a"]), ("N1", ["a"]), ("N2", [])] .
 [["N0", ["a", "a"]), ("N3", []), ("N1", []), ("N2", ["a"])]]
```

In the experiments presented in this paper (in Example 4 and in Example 5), the output is a set (implemented as a list) of type P , where each element is an execution trace of type Q , and each execution trace is a sequence of observables of type Obs . Each observable is displayed on a separate line, and the observables that make up an execution trace are displayed in chronological order. In this example, the output of `(denp rho1)` is a set (of type P) containing a single execution trace, where output neuron N_3 produces spikes (received by neuron N_0) in steps 1 and 9. Thus, the result of the computation is $8 = 9 - 1$.

3 Semantic Interpreter for Finite Execution Traces

The semantic interpreter presented in Section 2 produces all possible execution traces regardless of the length or number of execution traces. The final result of the semantic interpreter is an element of a linear time powerdomain [1]. In the presence of nondeterminism the implementation solution described in Section 2 only provides support to run and verify the execution of toy \mathcal{L}_{snp} programs. This is not surprising. An element of a powerdomain is exponential in the length of execution traces, hence a direct implementation can lead to an intractable solution.⁸ We could adapt our semantic interpreter to produce a single execution trace. For example, such an execution could be randomly selected from all possible executions, e.g., by using a (pseudo) random number generator [4]. Such

⁷ In the Haskell implementation available at [22] this program ρ_1 is stored in variable `rho1 :: Prog` (in the both files `semSNP.hs` and `semSNP-fin.hs`).

⁸ An element of a powerdomain is a tree-like structure, or a collection of "traces" essentially equivalent to an unfolding of such a tree.

an implementation would be tractable, but could only be used for simulation purposes and, in general, would provide only limited information regarding the behaviour of nondeterministic SN P systems.

In this section we explore an alternative implementation option. The semantic interpreter presented in this section is designed to prune the final yield of the semantic function preserving only a finite prefix for each execution trace. Intuitively, the interpreter stops each execution trace after a given number of steps (accepted as an argument by the semantic interpreter). This solution does not solve the tractability issue, but it can provide support for verifying bounded versions of nondeterministic SN P systems, as illustrated in Example 5.

The Haskell interpreter described in this section (available online at [22] in file `semSNP-fin.hs`) can be obtained from the semantic interpreter described in Section 2 with only few simple modifications, which are described below.

The Haskell type `F` implements the final yield of our semantic interpreter. The definition of type `F` changes now to:

```
type F = Int -> P
```

An element of the final domain `F` is a function of type `Int -> P`, which accepts as argument a number representing the length of the finite prefix that is produced for each execution trace contained in a set of type `P`. We recall that an element of type `P` is a set (implemented as a Haskell list) of execution traces of the type `Q` (the types `P` and `Q` are presented in Section 2).

The definition of the prefixing operator `preff` is adapted to prune the final yield of the semantic interpreter preserving only a finite prefix of a given length for each execution trace:

```
preff :: Obs -> F -> F
preff obs f =
  \l -> if l<=0 then [Qe] else [ Q obs q | q <- f (l-1) ]
```

The operators `bignedf` and `fe` are easily adapted to the new definition of type `F`.

```
bignedf :: [F] -> F
bignedf fs = \l -> bigned [ f l | f <- fs ]

fe :: F
fe = \l -> [Qe]
```

No other modifications are required. However, note that now the semantic interpreter mapping (`denp rho l`) accepts two arguments: a program `rho` of type `Prog` and an additional argument `l` of type `Int` representing the number of steps after which the execution is stopped for each trace.

Example 5 *We consider an \mathcal{L}_{snp} program ρ_3 which implements the spiking neural P system Π_3 given in [9] (section 5, Figure 4). We use the same notations and conventions as in Example 4. In [9], Π_3 is described by a tuple $\Pi_3 =$*

$(\{a\}, \sigma_1, \sigma_2, \sigma_3, \{(1, 2), (2, 1), (1, 3), (2, 3)\}, 3)$. The three neurons σ_1, σ_2 and σ_3 behave as follows: $\sigma_1 = (2, \{a^2/a \rightarrow a; 0, a \rightarrow \lambda\})$, $\sigma_2 = (1, \{a \rightarrow a; 0, a \rightarrow a; 1\})$ and $\sigma_3 = (3, \{a^3 \rightarrow a; 0, a \rightarrow a; 1, a^2 \rightarrow \lambda\})$. In the first time unit, all the neurons fire. In particular, the output neuron σ_3 emits the first spike in step 1. In a nondeterministic manner, neuron σ_2 chooses between rules $a \rightarrow a; 0$ and $a \rightarrow a; 1$. As long as neuron σ_2 chooses rule $a \rightarrow a; 0$, neurons σ_1 and σ_2 transmit each other one spike and (together) they transmit two spikes to neuron σ_3 , which (applying its forgetting rule $a^2 \rightarrow \lambda$) eliminates the two spikes in the next step. Alternatively, if neuron σ_2 chooses rule $a \rightarrow a; 1$, then it moves to the closed status for one time unit; it does not receive the spike emitted by neuron σ_1 , and so neuron σ_2 remains empty. In the next step, neuron σ_3 fires applying the rule $a \rightarrow a; 1$, hence it is closed for one time unit and cannot receive the spike issued by neuron σ_2 . The spike issued by neuron σ_2 is received by neuron σ_1 , but it is removed using the forgetting rule $a \rightarrow \lambda$. Thus, all the neurons remain empty. Finally, the output neuron σ_3 emits its second spike with a delay of 1 time unit since the moment it fired, by applying the rule $a \rightarrow a; 1$. Thus, there are at least 2 time units between the two spikes produced by the output neuron σ_3 ; in this way, system Π_3 generates in a nondeterministic manner all natural numbers greater than 1 (2, 3, 4, 5, ...), as in [9].

The \mathcal{L}_{snp} program ρ_3 presented below is designed to capture the behaviour of spiking neural P system Π_3 described in [9], namely to generate all natural numbers greater than 1 (2, 3, 4, 5, ...) in a nondeterministic manner. The program ρ_3 is given by $\rho_3 = (D_3, s_3)$, where the statement $s_3 \in S$ is

$$s_3 = (\text{send } \{N_1\} a)^2 \parallel \text{send } \{N_2\} a \parallel (\text{send } \{N_3\} a)^3$$

and the declaration $D_3 \in \text{Decl}$ is given by

$$\begin{aligned}
 D_3 = & \text{neuron } N_0 \{ r_\epsilon \mid \{N_1, N_2, N_3\} \}, \\
 & \text{neuron } N_1 \{ a^2/[a] \rightarrow a; 0, [a] \rightarrow \lambda \mid \{N_2, N_3\} \}, \\
 & \text{neuron } N_2 \{ [a] \rightarrow a; 0, [a] \rightarrow a; 1 \mid \{N_1, N_3\} \}, \\
 & \text{neuron } N_3 \{ [a^3] \rightarrow a; 0, [a] \rightarrow a; 1, [a^2] \rightarrow \lambda \mid \{N_0\} \}.
 \end{aligned}$$

After the initialization step (described by statement s_3) neuron N_1 contains 2 spikes (the multiset $[a, a]$), neuron N_2 contains 1 spike, neuron N_3 contains 3 spikes, and neuron N_0 contains 0 spikes. In our \mathcal{L}_{snp} program ρ_3 , neurons N_1, N_2 and N_3 implement the neurons σ_1, σ_2 and σ_3 , respectively, presented in the description of system Π_3 given in [9].

The semantic interpreter described in this section is available online at [22] in the file `semSNP-fin.hs`, where the \mathcal{L}_{snp} program ρ_3 is implemented and stored in variable `rho3 :: Prog`. In this case the semantic interpreter `denp` accepts as extra argument a natural number indicating a finite number of steps after which execution is stopped for each alternative trace. Using the interpreter `semSNP-fin.hs` to run this program with `(denp rho3 7)` the following output is obtained:

```
[[[("N0", []), ("N1", ["a", "a"]), ("N3", ["a", "a", "a"]), ("N2", ["a"])] .
```



```
[("N0", ["a"]), ("N1", ["a"]), ("N3", ["a"]), ("N2", ["a"])] .
[("N0", ["a"]), ("N1", ["a"]), ("N3", ["a"]), ("N2", [])] .
[("N0", ["a", "a"]), ("N1", []), ("N3", []), ("N2", [])]
```

The output presented above displays a set of execution traces separated by empty lines: there are 7 distinct execution traces. Each execution trace is a sequence containing at most 7 observables (the observables are displayed on separate lines in chronological order). Notice that the first spike is emitted by the output neuron N_3 (and received by neuron N_0) in step 2 – the first step is used for initialization. The next spike is emitted by the the output neuron N_3 (and received by neuron N_0) in steps 4, 5, 6 or 7. Thus, the distance between the first two spikes produced by the output neuron N_3 may be 2, 3, 4 or 5 (this can be observed in the last four execution traces of the experiment). The output neuron N_3 will continue to produce spikes in subsequent steps as well, but in our experiment execution is stopped (all execution traces are pruned) after the first 7 steps.⁹ Anyway, this example confirms experimentally that the system Π_3 presented in [9] (ρ_3 in our implementation) can generate the sequence of numbers 2, 3, 4, 5, ... (i.e., $\mathbb{N}^+ \setminus \{1\}$, where \mathbb{N}^+ is the set of natural numbers without 0).

The interpreter `semSNP-fin.hs` can also be used to test the \mathcal{L}_{snp} program ρ_1 presented in Example 4. In file `semSNP-fin.hs`, the program ρ_1 is implemented and stored in variable `rho1 :: Prog` for the case when $n = 2$ and the execution terminates after 10 steps. Hence, using the interpreter `semSNP-fin.hs` to run this program with `(denp rho1 10)` (or `(denp rho1 z)` with $z > 10$), it is obtained the same output as in Example 4.

4 Conclusion

We implemented the spiking neural P systems by using the purely functional programming language Haskell. The SN P systems implemented by Haskell programs might be seen as 'executable mathematics' [15]. We present the Haskell implementation of a semantic interpreter for a simple concurrent language providing constructs that can be used to describe the structure and behaviour of SN P systems. The semantic interpreter is designed following the discipline of denotational semantics [19]; it provides support for verifying the behaviour of SN P systems. Nondeterministic systems are modelled taking into account all possible interactions describing the behaviour of an SN P system. In order to obtain a tractable solution, only finite execution traces are verified.

There is a large variety of SN P systems [12, 13]. The implementation presented in this paper could be used for the simulation and verification in finite trace semantics of several classes of SN P systems.

⁹ Notice that executing `(den rho3 7)` of the nondeterministic program ρ_3 by using the interpreter `semSNP-fin.hs` requires around 120 seconds on a processor Intel(R) Core(TM) i5-7200U with CPU 2.50 GHz, while executing the deterministic program ρ_1 produces its output almost instantly.

References

1. J.W. de Bakker, E.P. de Vink. *Control Flow Semantics*. MIT Press (1996).
2. F.G. Cabarle, H.N. Adorna, M.A. Martínez-del-Amor, Simulating Spiking Neural P Systems Without Delays Using GPUs, *Int. J. Nat. Comput. Res.*, vol. 2(2), 19–31 (2011).
3. H. Chen, M. Ionescu, T.-O. Isidorj, A. Păun, Gh. Păun, M.J. Pérez-Jiménez. Spiking neural P systems with extended rules: universality and languages. *Natural Computing* 7(2), 147–166 (2008).
4. G. Ciobanu, E.N. Todoran, A methodology for concurrent languages development based on denotational semantics, *Proc. of 11th SYNASC 2009*, IEEE Computer Press, pp. 290-298 (2009).
5. G. Ciobanu, E.N. Todoran, Continuation semantics for asynchronous concurrency, *Fundamenta Informaticae* **131**(3-4), 373–388 (2014).
6. G. Ciobanu, E.N. Todoran. Denotational semantics of membrane systems by using complete metric spaces. *Theoretical Computer Science* **701**, 85–108 (2017).
7. G. Ciobanu, E.N. Todoran. A semantic investigation of spiking neural P systems. *Lecture Notes in Computer Science* **11399**, 108–130 (2019).
8. G. Gierz, K.H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D.S. Scott, *Continuous Lattices and Domains*, Cambridge University Press, 2003.
9. M. Ionescu, G. Păun, T. Yokomori, Spiking neural P systems, *Fundamenta Informaticae* **71**, 279–308 (2006).
10. M. Ionescu, G. Păun, M.J. Pérez-Jiménez and A. Rodríguez-Patón, Spiking neural P systems with several types of spikes, *Int. J. of Computers, Communications & Control*, vol. 6, pp. 647655 (2011).
11. Gh. Păun. *Membrane Computing. An Introduction*. Springer (2002).
12. Gh. Păun, G. Rozenberg, A. Salomaa (eds.). *Handbook of Membrane Computing*. Oxford University Press (2010).
13. L. Pan, Gh. Păun, G. Zhang, F. Neri, Spiking Neural P Systems with Communication on Request, *International Journal of Neural Systems*, vol. 27(8) (2017).
14. G. Plotkin, A powerdomain construction, *SIAM Journal of Computing*, vol.5, pp. 45287 (1976).
15. F. Rabhi, G. Lapalme, *Algorithms: a Functional Programming Approach*, Addison-Wesley, 1999.
16. G. Rozenberg, A. Salomaa (eds.). *Handbook of Formal Languages*. 3 volumes, Springer (1998).
17. D.A. Schmidt, *Denotational semantics: a methodology for language development*, Allyn & Bacon, 1986.
18. D.S. Scott, *Outline of a mathematical theory of computation*, Technical Monograph PRG-2, Oxford University Computing Laboratory, Oxford, England, November 1970.
19. D.S. Scott and C. Strachey, *Toward a mathematical semantics for computer languages*, Oxford Programming Research Group Technical Monograph. PRG-6. 1971.
20. D.S. Scott, *What is Denotational Semantics?* MIT Laboratory for Computer Science Distinguished Lecture Series, 1980.
21. E.N. Todoran. Metric semantics for synchronous and asynchronous communication: a continuation-based approach. *Electronic Notes in Theoretical Computer Science* **28**, 101–127 (2000).
22. Haskell implementation of the denotational semantics presented in this paper <http://ftp.utcluj.ro/pub/users/gc/eneia/cmc21> (2021).

A Modified Optimization Spiking Neural P System for Knapsack Problems

Jianping Dong¹, Gexiang Zhang^{2,3,4}*, Biao Luo², Qiang Yang³, Dequan Guo³, Haina Rong⁴, and Ming Zhu³

¹ School of Nuclear Technology and Automation Engineering,
Chengdu University of Technology, Chengdu 610059, China
master_djp@163.com

² Research Center for Artificial Intelligence,
Chengdu University of Technology, Chengdu 610059, China
zhgxdylan@126.com, 15775960380@163.com

³ School of Control Engineering,
Chengdu University of Information Technology, Chengdu, 610225, China
yuxia2008@126.com, newgdq@126.com, zhuming@cuit.edu.cn

⁴ School of Electrical Engineering,
Southwest Jiaotong University, Chengdu, 611756, China
ronghaina@126.com

Abstract. An optimization spiking neural P system (OSNPS) aims to obtain the approximate solutions of combinatorial optimization problems without the aid of evolutionary operators of evolutionary algorithms or swarm intelligence algorithms. To develop the promising and significant research direction, this paper proposes a distributed adaptive optimization spiking neural P system (DAOSNPS) with a distributed population structure and a new adaptive learning rate considering population diversity. Extensive experiments on knapsack problems show that DAOSNPS gains much better and more stable solutions than OSNPS, adaptive optimization spiking neural P system (AOSNPS) and other two optimization algorithms. Population diversity and convergence analysis indicate that DAOSNPS achieves a better balance between exploration and exploitation than OSNPS and AOSNPS.

Keywords: Membrane computing; spiking neural P system; optimization spiking neural P system; combinatorial optimization problems.

1 Introduction

As a branch of natural computing, membrane computing abstracts computing models, called P system, inspired from the structure and the functioning of the biological cells, organs and colonies of bacteria [1,2,3]. This area was initiated by Păun in 1998 and become one of the most promising and important research directions. Since then, membrane computing models have been studied and used in various applications such

* Corresponding author.

as biology and biomedicine [4], computer graphics [5], cryptography [6], robot control [7,8,9], image processing [10,11,12], distributed evolutionary computing [13,14], power system fault diagnosis [15,16,17] and other real-life complex problems [18,19].

Common models of membrane computing can be divided into three categories in terms of their membrane structures: cell-like P systems consist of a single cell's membrane structure, objects and evolution rules that are used to control objects evolution within a membrane and information communication between membranes [20,21]; tissue-like P systems are composed of a multicellular structure, objects, rules and communication rules where communication rules are used for offering channels between two different cells (similar to protein channels of biology membrane) [22,23,24]; neural-like P systems are inspired by the neurophysiological behavior of neurons sending electrical impulses (spikes) along axons to other neurons [25,26,27]. Unlike cell-like P systems and tissue-like P systems which use "symbol" to encode a piece of information, spiking neural P systems (SNPS), were introduced by Ionescu et al. in [28], deal with a unique object. Nowadays, much attentions is paid to SNPS from theory, applications to implementations [29,30,31,32,33,34].

In recent years, not only many new features have been abstracted to form new P systems, but also the features of existing P systems have been combined to form new P systems [35,36]. Besides, how to extend these P systems to solve combinatorial optimization problems is one of the most promising and important research directions [37,38,39]. In the fields of evolutionary membrane computing, related researchers have proposed two research topics: membrane-inspired evolutionary algorithms (MIEAs) [40,41,42] and automatic design of membrane computing models (ADMCM) [43,44,45]. After that, Zhang et al. proposed a novel way to design a P system for directly obtaining the approximate solutions of combinatorial optimization problems without the aid of evolutionary operators, optimization spiking neural P system (OSNPS) [46]. In OSNPS [46], an extended spiking neural P system (ESNPS) has been proposed by introducing the probabilistic selection of evolution rules, multi-neurons output and a family of ESNPS. In this work, a Guider algorithm is firstly introduced to SNPS for adjusting probabilities of spiking rules. The experimental results of OSNPS appear promising and competitive when compared with six other optimization algorithms by solving the same knapsack problem. As indicated in [46], one of the main limitations of OSNPS is that the learning rate is constant while the search may require a different learning rate in different moments of the search. For this limitation, AOSNPS [47] extends the work of OSNPS [46]. As we can see from [47], AOSNPS with adaptive learning rate and adaptive mutation has the faster convergence in the early stage and the better population diversity in the later stage. In [47], experiments show that AOSNPS has further improved the capability of OSNPS. However, as indicated in [47], one limitation is that the population diversity decreases significantly in the initial stage, which may jeopardize the performance of the algorithm.

This paper extends the work of OSNPS [46] and AOSNPS [47]. A novel modified spiking neural P system, a distributed adaptive optimization spiking neural P system, is proposed to solve combinatorial optimization problems. More specifically, a distributed population structure is introduced to maintain the better population diversity in the later

stage. In addition, a new adaptive learning rate considering population diversity is proposed to enhance the convergence speed in the early stage.

The remainder of this paper is organized as follows. Section 2 briefly reviews the optimization spiking neural P system. Section 3 presents the DAOSNPS consisting of a distributed population structure and a novel adaptive learning rate. Section 4 presents the experimental results with discussions.

2 Background

In this section, we briefly review and analyze the optimization spiking neural P system. ESNPS and OSNPS are introduced in Subsections 2.1 and 2.2, respectively.

2.1 Extended Spiking Neural P System

An ESNPS is a P system augmented with two additional neurons σ_{m+1} and σ_{m+2} . An ESNPS of degree $m \geq 1$, as shown in Fig. 1, is defined as the following structure [46]

$$\Pi = (O, \sigma_1, \dots, \sigma_{m+2}, syn, I_0), \quad (1)$$

where:

- (1) $O = \{a\}$ is the singleton alphabet (a is called *spike*);
- (2) $\sigma_1, \dots, \sigma_m$ are neurons of the form $\sigma_i = (1, R_i, P_i)$, $1 \leq i \leq m$.
 - (a) In each neuron σ_i , there is only one initial spike.
 - (b) $R_i = \{r_i^1, r_i^2\}$ is a set of rules of the firing or forgetting spike, where $r_i^1 = \{a \rightarrow a\}$ is firing rule and $r_i^2 = \{a \rightarrow \lambda\}$ is forgetting rule.
 - (c) $P_i = \{p_i^1, p_i^2\}$ is a finite set of probabilities, where p_i^1 and p_i^2 are the selection probabilities of rules r_i^1 and r_i^2 , respectively, and satisfy $p_i^1 + p_i^2 = 1$.
- (3) The two additional neurons, $\sigma_{m+1} = \sigma_{m+2} = (1, \{a \rightarrow a\})$, work as a step by step supplier of spikes to neurons $\sigma_1, \sigma_2, \dots, \sigma_m$.
- (4) $syn = \{(i, j) | (1 \leq i \leq m + 1 \wedge j = m + 2) \vee (i = m + 2 \wedge j = m + 1)\}$.
- (5) $I_0 = \{1, 2, \dots, m\}$ is a finite set of *output* neurons, i.e., the output is a spike train formed by concatenating the outputs of $\sigma_1, \sigma_2, \dots, \sigma_m$.

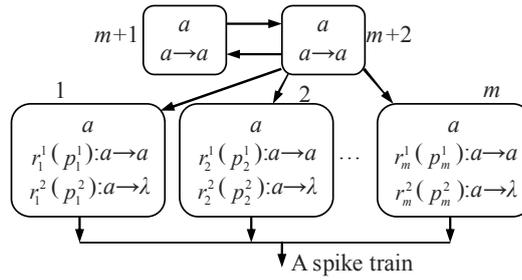


Fig. 1. Logical and functioning scheme of ESNPS

In [46], an ESNPS contains the subsystem consisting of neurons σ_{m+1} and σ_{m+2} , which are supplier of spikes to neurons $\sigma_1, \dots, \sigma_m$. σ_{m+1} and σ_{m+2} are the same neurons, each of which fires at each moment of time and sends a spike to each of neurons $\sigma_1, \dots, \sigma_m$. Each of neurons $\sigma_1, \dots, \sigma_m$ performs the firing rule r_i^1 by probability p_i^1 and the forgetting rule r_i^2 by probability p_i^2 , $i = 1, \dots, m$. Thus, this system outputs a spike train consisting of 0 and 1 at each time unit. The outputted spike train is controlled by adjusting the probabilities p_i^1, \dots, p_i^m . Hence, an adjustment strategy to adjust probabilities p_i^1, \dots, p_i^m by introducing a family of ESNPS is presented in the following subsection.

2.2 Optimization Spiking Neural P System

OSNPS is one variant of SNPS for solving combinatorial optimization problems [46]. Regarding its structure, OSNPS is composed of multiple ESNPS and a Guider algorithm. The structure of OSNPS is shown in Fig. 2.

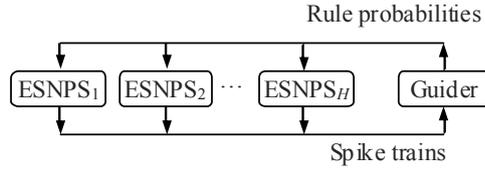


Fig. 2. Structure of OSNPS

In Fig. 2, each ESNPS_{*i*} has the same structure with $1 \leq i \leq H$ shown in Fig. 1. Here, H , the number of ESNPS, represents population size. The Guider algorithm consists of the rule probability matrix $\mathbf{P}_R = [p_{ij}^1]_{H \times m}$ (the apex 1 indicates the probability of the bit to be 1), the binary matrix $\mathbf{B}_R = [b_{ij}]_{H \times m}$ controlled by probability matrix $\mathbf{P}_R = [p_{ij}^1]_{H \times m}$ and evolutionary algorithm. $\mathbf{B}_R = [b_{ij}]_{H \times m}$, a set of H binary strings/candidate solutions of length m produced by each ESNPS_{*i*}, is the input of the Guider algorithm. $\mathbf{P}_R = [p_{ij}^1]_{H \times m}$, an adaptively adjusted matrix, is the output of the Guider algorithm.

$$\mathbf{P}_R = \begin{pmatrix} p_{11}^1 & p_{12}^1 & \cdots & p_{1m}^1 \\ p_{21}^1 & p_{22}^1 & \cdots & p_{2m}^1 \\ \vdots & \vdots & \ddots & \vdots \\ p_{H1}^1 & p_{H2}^1 & \cdots & p_{Hm}^1 \end{pmatrix} \quad \mathbf{B}_R = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{H1} & b_{H2} & \cdots & b_{Hm} \end{pmatrix}. \quad (2)$$

3 A Distributed Adaptive Optimization Spiking Neural P System

This section proposes DAOSNPS to improve the performance of OSNPS [46] and AOSNPS [47] by introducing two novel elements: a distributed population structure and an

adaptive learning rate considering population diversity. A distributed population structure and an adaptive learning rate considering population diversity are described in Subsections 3.1 and 3.2, respectively. The new Guider algorithm embedding two novel elements is outlined in Subsection 3.3.

3.1 Distributed Population Structure

In a distributed population structure, the population is divided into several subpopulations, and each subpopulation evolves independently and in parallel. At the same time, the information exchange is used to achieve the communication among the subpopulations at a specific time to promote the evolution of each subpopulation. Information exchange plays an important role in the process of the execution of Guider algorithm, including migration interval M_i (how many times does the population iterate and exchange information), migration number M_n (the number of individuals in the immigrating subpopulation $P^{g,im}$ is replaced by the number of individuals in the emigrating subpopulation $P^{g,em}$), migration direction (migration topological direction), the selection of migration individual and the selection of replacement individual. A distributed population structure is shown in Fig. 3.

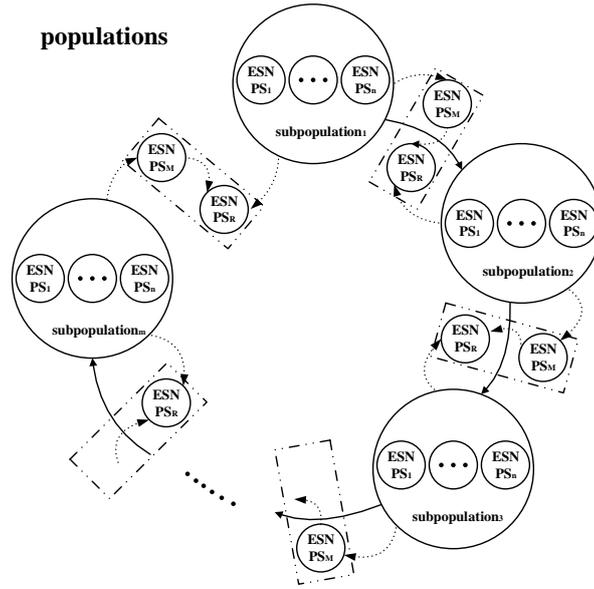


Fig. 3. The schematic diagram of distributed population structure in DAOSNPS

In Fig. 3, $subpopulation_i$ represents the i th subpopulation, $i = 1, \dots, m$. $ESN PS_i$ represents the i th individual in each subpopulation, $i = 1, \dots, n$. The two variables, m and n , represent the number of subpopulations and the number of individuals in each subpopulation, respectively. $ESN PS_M$ and $ESN PS_R$ are the migration individual in $P^{g,em}$ and the replacement individual in $P^{g,im}$.

The details of information exchange are as follows:

- (1) Initialization parameters: in most distributed evolutionary algorithms [48,49], information exchange is usually performed after certain number of iterations. The migration number is a single individual, and the migration topology is a ring topology. The selection of the migration individual is as follows:

- (a) A candidate set $S^{g,em}$ is constructed according to the average fitness value of individuals in $P^{g,em}$. The rule of construction is defined as:

$$S^{g,em} = \{ESNPS_i^{g,em} | ESNPS_i^{g,em} \in P^{g,em} \text{ and } f(ESNPS_i^{g,em}) \geq \overline{f^{g,em}}\}, \quad (3)$$

where $ESNPS_i^{g,em}$ is the i th individual of the g th generation in $P^{g,em}$, $i = 1, \dots, n$. $f(ESNPS_i^{g,em})$ is the fitness value of the i th individual of the g th generation in $P^{g,em}$. $\overline{f^{g,em}}$ is the average fitness value of the g th generation of $P^{g,em}$. If $f(ESNPS_i^{g,em})$ is greater than or equal to $\overline{f^{g,em}}$, $ESNPS_i^{g,em}$ is selected as one of elements in $S^{g,em}$, otherwise, $ESNPS_i^{g,em}$ is not selected as one of elements in $S^{g,em}$.

- (b) According to the diversity contribution of candidate individuals to the immigrating subpopulation, the migration individual is selected from $S^{g,em}$. The rule of selection is

$$ESNPS_M^{g,em} = \operatorname{argmax}_{ESNPS_{g,em} \in S^{g,em}} \|ESNPS^{g,em} - ESNPS_{ave}^{g,im}\|, \quad (4)$$

where $\|\cdot\|$ is the Euclidean distance. $ESNPS^{g,em}$ represents an individual from $S^{g,em}$. $ESNPS_{ave}^{g,im}$ is the average distance of $P^{g,im}$. If the Euclidean distance between $ESNPS^{g,em}$ and $ESNPS_{ave}^{g,im}$ is the greatest, $ESNPS^{g,em}$ is considered as $ESNPS_M^{g,em}$. Specially, when multiple individuals have the same maximum distance, one is randomly selected as $ESNPS_M^{g,em}$.

- (3) Replacement individual: the selection of replacement individual also meets the conditions of the fitness value and diversity. The details are as follows:

- (a) The Euclidean distance, diversity information, is calculated between pairs in $P^{g,im}$. The matrix of the Euclidean distance $\mathbf{D}^{g,im}$ is constructed as

$$\mathbf{D}^{g,im} = \begin{pmatrix} \|ESNPS_1^{g,im} - ESNPS_2^{g,im}\| & \dots & \|ESNPS_1^{g,im} - ESNPS_n^{g,im}\| \\ 0 & \dots & \|ESNPS_2^{g,im} - ESNPS_n^{g,im}\| \\ 0 & \dots & \|ESNPS_3^{g,im} - ESNPS_n^{g,im}\| \\ 0 & \dots & \|ESNPS_4^{g,im} - ESNPS_n^{g,im}\| \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix}, \quad (5)$$

where $ESNPS_i^{g,im}$ represents the i th individual in $P^{g,im}$, $i = 1, \dots, n$. Because the diversity matrix constructed is symmetric, the lower triangle is set to 0. Two individuals ($ESNPS_{r1}^{g,im}$ and $ESNPS_{r2}^{g,im}$) with minimum similarity (minimum Euclidean distance) are selected from $\mathbf{D}^{g,im}$ as candidate replacement individuals. In particular, if there are multiple pairs of individuals with minimum similarity, a pair of individuals is randomly selected as candidate replacement individuals.

- (b) The replacement individual is selected from candidate replacement individuals. The selected rule is

$$ESNPS_R^{g,im} = \begin{cases} ESNPS_{r_1}^{g,im}, & \text{if } f(ESNPS_{r_1}^{g,im}) \geq f(ESNPS_{r_2}^{g,im}) \\ ESNPS_{r_2}^{g,im}, & \text{if } f(ESNPS_{r_1}^{g,im}) < f(ESNPS_{r_2}^{g,im}) \end{cases}, \quad (6)$$

where $f(ESNPS_{r_1}^{g,im})$ and $f(ESNPS_{r_2}^{g,im})$ are the fitness values of $ESNPS_{r_1}^{g,im}$ and $ESNPS_{r_2}^{g,im}$, respectively. If $f(ESNPS_{r_1}^{g,im})$ is greater than or equal to $f(ESNPS_{r_2}^{g,im})$, $ESNPS_{r_1}^{g,im}$ is selected as $ESNPS_R^{g,im}$, otherwise, $ESNPS_{r_2}^{g,im}$ is selected as $ESNPS_R^{g,im}$.

- (4) Information exchange: from the first subpopulation to the m th subpopulation, information exchange is completed in the order of (1)-(3).

Compared with the centralized structure of OSNPS and AOSNPS, the distributed population structure of DAOSNPS has two differences as follows.

- (1) The centralized structure has only one best solution searched, and other individuals learn from the best individual searched, while the distributed population structure has multiple best individuals searched in multiple subpopulations, and other individuals in each subpopulation learn from the best individual of the subpopulation.
- (2) Each subpopulation evolves independently and multidirectionally without any influence among subpopulations. When a certain number of iterations is reached, it will perform information exchange. However, all individuals are learning at the same direction in OSNPS and AOSNPS.

3.2 Adaptive Learning Rate

The learning rate is the step size of probability adjustment for the elements p_{ij}^1 of the probability matrix \mathbf{P}_R . In OSNPS, the update rule of the current probability is defined as

$$p_{ij}^1 = p_{ij}^1 + \Delta \quad (7)$$

$$p_{ij}^1 = p_{ij}^1 - \Delta \quad (8)$$

with $1 \leq i \leq H$ and $1 \leq j \leq m$ in [46]. Δ is a random number between 0.005 and 0.02. The term p_{ij}^1 is the selection probability of rule r_{ij}^1 . If desired probability corresponding to the current binary bit in binary matrix \mathbf{B}_R is 1, the update rule of the current probability is (7), otherwise, the update rule of the current probability is (8). In AOSNPS, the update rule of the current probability is defined as

$$p_{ij}^1 = p_{ij}^1 + \Delta_{ij}^a = p_{ij}^1 + \frac{1 - p_{ij}^1}{2} = 0.5 + 0.5p_{ij}^1 \quad (9)$$

$$p_{ij}^1 = p_{ij}^1 + \Delta_{ij}^a = p_{ij}^1 + \frac{0 - p_{ij}^1}{2} = 0.5p_{ij}^1 \quad (10)$$

with $1 \leq i \leq H$ and $1 \leq i \leq m$ in [47]. The term p_{ij}^1 is the selection probability of rule r_{ij}^1 . Δ_{ij}^a is an adaptive probability adjustment step size. If desired probability corresponding to the current binary bit in binary matrix \mathbf{B}_R is 1, the update rule of the current probability is shown in (9), otherwise, the update rule of the current probability is shown in (10).

This paper designs a new adaptive probability adjustment step size for each element p_{ijk}^1 in \mathbf{P}_R in the i th subpopulation, $i = 1, \dots, m$, $j = 1, \dots, n$ and $k = 1, \dots, l$. At each time unit, the update rule of the current probability is

$$p_{ijk}^1 = p_{ijk}^1 + \Delta_{ijk}^d = p_{ijk}^1 + w_{ik}^0 \times (1 - p_{ijk}^1), \quad (11)$$

$$p_{ijk}^1 = p_{ijk}^1 + \Delta_{ijk}^d = p_{ijk}^1 + w_{ik}^1 \times (0 - p_{ijk}^1), \quad (12)$$

where Δ_{ijk}^d is an adaptive probability adjustment step size with $1 \leq i \leq m$, $1 \leq j \leq n$ and $1 \leq k \leq l$. w_{ik}^0 and w_{ik}^1 are the learning weights defined as

$$w_{ik}^0 = \frac{n_{ik}^0 + \frac{n}{2}}{2n}, \quad (13)$$

$$w_{ik}^1 = \frac{n_{ik}^1 + \frac{n}{2}}{2n}. \quad (14)$$

In (13) and (14), n_{ik}^0 and n_{ik}^1 refer to the number of 1 and 0 in the k th column of matrix \mathbf{B}_R in the i th subpopulation, respectively. m and n represent the number of subpopulations and the number of individuals in the independent subpopulation, respectively. If the desired probability corresponding to the current binary bit in binary matrix \mathbf{B}_R is 1 and the proportion of alleles with 0 (the learning weight w_{ik}^0) in the i th subpopulation, the update rule of the current probability is shown in (11). If the desired probability corresponding to the current binary bit in binary matrix \mathbf{B}_R is 0 and the proportion of alleles with 1 (the learning weight w_{ik}^1) in the i th subpopulation, the update rule of the current probability is shown in (12).

Compared with Δ of OSNPS in [46] and Δ_{ij}^a of AOSNPS in [47], the new adaptive probability adjustment step size Δ_{ijk}^d has the following features.

- (1) Δ_{ijk}^d depends not only on the distance between the current probability value and 0/1, but also on the number of alleles with 1/0. If the expected probability value is 1 and the proportion of alleles with 1 is small, the step that the current probability value p_{ijk}^1 should be close to 1 is relatively large. If the expected probability value is 1 and the proportion of alleles with 1 is high, the step that the current probability value p_{ijk}^1 should be close to 1 is relatively small.
- (2) DAOSNPS has a small learning step size than AOSNPS when it is close to the desired probability value. For example, if the desired probability value, the current probability value and the proportion of alleles with 0 are 1, 0.9 and 0.2, respectively, Δ_{ijk}^d and Δ_{ij}^a are 0.02 and 0.05 respectively.

3.3 The Guider Algorithm of DAOSNPS

Based on distributed population structure and adaptive learning rate, a novel Guider algorithm is shown in Algorithm 1. To clearly illustrate the Guider algorithm, we explain the details step by step as follows:

Step 1: Set initial parameters including spike train T_s , probability p_j^a , mutation probability P_j^m , the number of ESNPS H and the current best solution $\mathbf{x} = (x_1, x_2, \dots, x_m)$ of length l , the number of subpopulations m , the number of individuals in each subpopulation n , the length of problems l , migration interval M_i , migration number M_n . Rearrange T_s as matrix \mathbf{P}_R . \mathbf{B}_R is generated by probability matrix \mathbf{P}_R . Initialize gen , P_{m1} , M_i , M_n and then calculate the initialized fitness value $G_{bf}(0)$ and the initialized diversity value $DP_a(0)$.

Step 2: The initial population is divided into m subpopulations (\mathbf{P}_R and \mathbf{B}_R are divided into m submatrixs according to the subpopulations), each of which includes n individuals.

Step 3: If the parameter gen is greater than $maxgen$, i.e., $gen \geq maxgen$, the algorithm goes to **Step 21**.

Step 4: The number of iterations increases one, i.e., $gen = gen + 1$.

Step 5: Assign the subpopulation indicator the initial value $i = 1$.

Step 6: If the subpopulation indicator is greater than its maximum m , i.e., $i > m$, the algorithm goes to **Step 16**.

Step 7: Assign the row indicator the initial value $j = 1$.

Step 8: If the row indicator is greater than its maximum n , i.e., $j > n$, the algorithm goes to **Step 15**.

Step 9: Assign the column indicator the initial value $k = 1$.

Step 10: If the column indicator is greater than its maximum l , i.e., $k > l$, the algorithm goes to **Step 14**.

Step 11: If a random number $rand$ is less than the prescribed learning probability p_j^a , the algorithm performs the following two steps, otherwise, it goes to **Step 12**.

- (i) Choose two distinct chromosomes c_1 and c_2 that differ from the i th individual among the n chromosomes, i.e., $c_1 \neq c_2 \neq i$. If $f(x^{c_1}) > f(x^{c_2})$ ($f(\cdot)$ is an evaluation function to an optimization problem; x^{c_1} and x^{c_2} denote the c_1 th and c_2 th chromosomes, respectively), i.e., the c_1 th chromosome is better than the c_2 th one in terms of their fitness values (here we consider a maximization problem), the current individual learns from the c_1 th chromosome, i.e., $x_j = x_j^{c_1}$, otherwise, the current individual learns from the c_2 th chromosome, i.e., $x_j = x_j^{c_2}$, where x_j , $x_j^{c_1}$ and $x_j^{c_2}$ are an intermediate variable, the j th bits of the c_1 th and c_2 th chromosomes, respectively.
- (ii) If $x_j = 1$, we increase the current rule probability p_{ijk}^1 to $p_{ijk}^1 + w_{ik}^0 \times (1 - p_{ijk}^1)$, otherwise, we decrease p_{ijk}^1 to $p_{ijk}^1 + w_{ik}^1 \times (0 - p_{ijk}^1)$.

Step 12: If $b_j^{max} = 1$, the current rule probability p_{ijk}^1 is increased to $p_{ijk}^1 + w_{ik}^0 \times (1 - p_{ijk}^1)$, otherwise, p_{ijk}^1 is decreased to $p_{ijk}^1 + w_{ik}^1 \times (0 - p_{ijk}^1)$, where b_j^{max} is the j th bit of the best chromosome found.

Step 13: The column indicator k increases 1 and the algorithm goes to **Step 10**.

Algorithm 1 The new Guider algorithm of DAOSNPS

Input: $T_s, p_j^a, P_j^m, m, n, M_i, M_n$.

```

1: Rearrange  $T_s$  as matrix  $\mathbf{P}_R$ ,  $\mathbf{B}_R$  is generated by  $\mathbf{P}_R$ , initialize the parameters.
2: The initial population is divided into  $m$  subpopulations
3: while ( $gen \leq maxgen$ ) do
4:    $gen = gen + 1$ 
5:   for  $i = 1$  to  $m$  do
6:     for  $j = 1$  to  $n$  do
7:       for  $k = 1$  to  $l$  do
8:         if ( $rand() < p_j^a$ ) then
9:            $c_1, c_2 = ceil(rand * n), c_1 \neq c_2 \neq i$ 
10:          if ( $f(x^{c_1}) > f(x^{c_2})$ ) then
11:             $x_j = x_j^{c_1}$ 
12:          else
13:             $x_j = x_j^{c_2}$ 
14:          end if
15:          if ( $x_j == 1$ ) then
16:             $p_{ijk}^1 = p_{ijk}^1 + w_{ik}^0 \times (1 - p_{ijk}^1)$    {adaptive learning}
17:          else
18:             $p_{ijk}^1 = p_{ijk}^1 + w_{ik}^1 \times (0 - p_{ijk}^1)$    {adaptive learning}
19:          end if
20:          else
21:            if ( $x_j^{max} == 1$ ) then
22:               $p_{ijk}^1 = p_{ijk}^1 + w_{ik}^0 \times (1 - p_{ijk}^1)$    {adaptive learning}
23:            else
24:               $p_{ijk}^1 = p_{ijk}^1 + w_{ik}^1 \times (0 - p_{ijk}^1)$    {adaptive learning}
25:            end if
26:          end if
27:        end for
28:      end for
29:    end for
30:    {information exchange}
31:    if  $mod(gen, C_i) == 0$  then
32:      for  $i = 1$  to  $m$  do
33:        The candidate migrant individuals set is constructed by (3).
34:         $ESNPS_M^{g,em}$  is selected from  $S^{g,em}$  by (4).
35:        The matrix of the Euclidean distance  $\mathbf{D}^{g,im}$  is constructed by (5).
36:        Two individuals with minimum similarity are selected from  $\mathbf{D}^{g,im}$ .
37:         $ESNPS_R^{g,im}$  is selected from  $ESNPS_{r1}^{g,im}$  and  $ESNPS_{r2}^{g,im}$  by (6).
38:         $ESNPS_R^{g,im}$  is replaced by  $ESNPS_M^{g,em}$ .
39:      end for
40:    end if
41:    if Meet the condition of mutation in AOSNPS [47] then
42:      Adaptive mutation is the same as AOSNPS [47].
43:    end if
44:  end while

```

Output: Rule probability matrix \mathbf{P}_R

Step 14: The row indicator j increases 1 and the algorithm goes to **Step 8**.

Step 15: The subpopulation indicator i increases 1 and the algorithm goes to **Step 6**.

Step 16: If gen is an integral multiple of C_i , i.e., $mod(gen, C_i) \neq 0$, the algorithm goes to **Step 17**, otherwise, the algorithm goes to **Step 19**.

Step 17: Assign the subpopulation indicator the initial value $i = 1$.

Step 18: If the subpopulation indicator is greater than its maximum m , i.e., $i > m$, the algorithm performs the following several steps, otherwise, the algorithm goes to **Step 19**.

- (1) According to (3) in $P^{g,em}$, the candidate migration individuals set is constructed.
- (2) According to (4), the migration individual $ESNPS_M^{g,em}$ is selected from $S^{g,em}$.
- (3) According to (5) in $P^{g,r}$, the matrix of the Euclidean distance $D^{g,r}$ is constructed.
- (4) Two individuals ($ESNPS_{r_1}^{g,im}$ and $ESNPS_{r_2}^{g,im}$) with minimum similarity are selected from $D^{g,im}$.
- (5) Two individuals ($ESNPS_{r_1}^{g,im}$ and $ESNPS_{r_2}^{g,im}$) with minimum similarity are selected from $D^{g,im}$.
- (6) According to (6), the replacement individual $ESNPS_R^{g,im}$ is selected from $ESNPS_{r_1}^{g,im}$ and $ESNPS_{r_2}^{g,im}$. The replacement individual $ESNPS_R^{g,im}$ is replaced by migration individual $ESNPS_M^{g,em}$.

Step 19: The subpopulation indicator i increases 1 and the algorithm goes to **Step 18**.

Step 20: If algorithm meets mutation conditions in Subsection 3.2 in [47], performs the following several steps, otherwise, the algorithm goes to **Step 3**.

- (i) Assign the subpopulation indicator the initial value $i = 1$.
- (ii) If the subpopulation indicator is greater than its maximum m , i.e., $i > m$, the algorithm goes to **Step 3**.
- (iii) Assign the row indicator the initial value $j = 1$.
- (iv) If the row indicator is greater than its maximum n , i.e., $j > n$, the algorithm goes to (x) in **Step 20**.
- (v) Assign the column indicator the initial value $k = 1$.
- (vi) If the column indicator is greater than its maximum l , i.e., $k > l$, the algorithm goes to (ix) in **Step 20**.
- (vii) If a random number $rand$ is less than the prescribed mutation probabilities P_j^m , i.e., $rand_1() < P_j^m$, let $p_{ijk}^1 = rand_2()$, where, $rand_1()$ and $rand_2()$ are random numbers [47].
- (viii) The column indicator k increases 1 and the algorithm goes to (vi) in **Step 20**.
- (ix) The row indicator j increases 1 and the algorithm goes to (iv) in **Step 20**.
- (x) The subpopulation indicator i increases 1 and the algorithm goes to (ii) in **Step 20**.

Step 21: The algorithm outputs the modified rule probability matrix P_R to adjust each probability value of each evolution rule inside each of neurons $1, \dots, l$ in each ESNPS.

4 Experimental Results

To verify the effectiveness of distributed population structure and adaptive learning rate of the proposed P system, we test DAOSNPS against other evolutionary algorithms on the 0/1 knapsack problem. The 0/1 knapsack problem is described in Subsection 4.1. Statistical analysis and diversity analysis of experimental results are made in Subsection 4.2 and 4.3, respectively.

4.1 The 0/1 Knapsack Problem

The 0/1 knapsack problem, a well-known NP-complete combinatorial optimization problem, is used to investigate the performance of DAOSNPS. It can be described as: given a group of items, each item with its own weight and price, and a knapsack with limited capacity, the problem consists of selecting the items to make the total price of the knapsack as high as possible without violating its maximum capacity [50]. Moreover, the 0/1 knapsack problem is to select a subset from the given number of items so as to maximize the profit.

$$f(\mathbf{x}) = \sum_{j=1}^m p_j x_j, \quad (15)$$

subject to

$$\sum_{j=1}^m \omega_j x_j \leq C, \quad (16)$$

where $\mathbf{x} = (x_1, x_2, \dots, x_m)$ is item set (if the item is selected, $x_j = 1$. Otherwise, $x_j = 0$). p_j and ω_j are price and weight of the j^{th} item, respectively. C is the capacity of the knapsack.

This study uses strongly correlated sets of unsorted data [41,51,52], i.e., 1) the weights ω_i are sampled from the interval $[1, \Omega]$, where Ω is the upper bound of ω_i for $i = 1, \dots, K$; 2) $p_i = \omega_i + \frac{1}{2}\Omega$; 3) the average knapsack capacity C is applied:

$$C = \frac{1}{2} \sum_{i=1}^K \omega_i. \quad (17)$$

4.2 Statistical Analysis of Experiment results

In this subsection, a DAOSNPS consisting of $H = 50$ ESNPS, each of which has a certain number of neurons such as 1002 for the knapsack problem with 1000 items, is used to solve 10 knapsack problems with respective 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000 and 10000 items. All experiments are implemented on the platform MATLAB and on a work station with Intel i7 2.93 GHz processor, 32GB RAM and Windows 10.

In DAOSNPS, learning probability p_j^g is set randomly in the range of $([0.05, 0.2])$. Population size is set to 50 ($H = 50$), which is the same as OSNPS and AOSNPS.

The mutation probability p_j^m and the consecutive maximum generations of constance are 0.01 and 500, respectively. the number of subpopulations m and the number of individuals n in each subpopulation are set to 5 and 10, respectively. Migration interval M_i and migration number M_n are set to 100 and 1, respectively.

In order to compare with the performance of the Genetic Quantum algorithm (GQA) [53], Novel Quantum Evolutionary algorithm (NQEA) [54], OSNPS [46] and AOSNPS [47], each algorithm in this paper has been run for 30 independent runs.

Table 1. Mean values and standard deviations of knapsack problems gained by five algorithms and Wilcoxon rank sum test results

item	GQA			NQEA			OSNPS			AOSNPS			DAOSNPS	
	μ	σ	w	μ	σ	w	μ	σ	w	μ	σ	w	μ	σ
1000	26340.617	162.941	+ 29273.757	131.036	+ 28089.664	311.094	+ 29225.319	186.584	+ 29901.429	48.703				
2000	52908.434	208.761	+ 58515.548	293.090	+ 56150.321	535.740	+ 58561.778	385.807	+ 59728.213	98.580				
3000	78059.658	276.459	+ 85886.637	502.859	+ 82745.029	692.869	+ 86738.048	586.712	+ 89547.684	221.011				
4000	103801.413	422.955	+ 113690.579	666.608	+ 109458.886	724.479	+ 114706.678	831.661	+ 119021.651	173.797				
5000	131224.181	342.227	+ 142077.755	713.820	+ 137927.802	835.482	+ 143601.783	1328.339	+ 148068.149	256.375				
6000	157119.187	415.339	+ 169516.775	577.702	+ 164674.207	730.289	+ 171543.765	1515.563	+ 177784.311	489.606				
7000	182669.798	440.894	+ 196377.110	873.184	+ 191659.447	849.224	+ 200107.477	1218.512	+ 205698.891	754.716				
8000	208561.466	322.213	+ 223674.462	953.050	+ 217577.959	1128.401	+ 227193.619	1531.739	+ 234831.002	691.028				
9000	233945.639	570.136	+ 249931.187	916.934	+ 244397.767	1129.060	+ 254551.819	1162.772	+ 263942.931	775.361				
10000	259881.277	541.028	+ 276903.178	1159.924	+ 270663.831	769.233	+ 282101.492	1774.430	+ 292116.032	906.898				

In Table 1, the performance of the algorithms is mainly represented by two calculation indicators (the mean value μ and the standard deviation σ of the knapsack costs $f(\mathbf{x})$) of 30 independent runs. The larger the μ is, the stronger the ability to find the optimal fitness value is. The standard deviation σ indicates the stability of the algorithm to find the optimal value (The smaller the standard deviation is, the more stable the algorithm is). In addition, the statistical analysis of experimental results has been done by the Wilcoxon rank sum test [55], where "+", "-" and "=" represent that DAOSNPS achieves better performance, worse performance and no difference than other algorithms, respectively.

Experiment results in Table 1 show that DAOSNPS can get better fitness values than other algorithms on ten instances. Furthermore, DAOSNPS has better stability than other algorithms (DAOSNPS is closer to the actual optimal fitness value than other algorithms). As shown in Figs. 4 and 5, to further explain the changes of average fitness values and standard deviations, we compare DAOSNPS with GQA, NQEA, OSNPS and AOSNPS, respectively, and calculate the improvement percentage in terms of average fitness values and standard deviations. In Fig. 4, the average fitness values of DAOSNPS in different numbers of items is better than other algorithms, especially for GQA, which shows that DAOSNPS has stronger ability to search the optimal value than GQA, NQEA, OSNPS and AOSNPS. Except for GQA, DAOSNPS has less standard deviations than the other three algorithms shown in Fig. 5, which indicates that DAOSNPS has better stability than NEQA, OSNPS and AOSNPS.

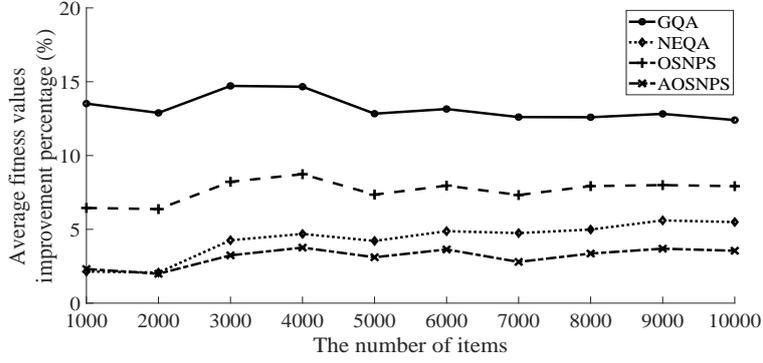


Fig. 4. Average fitness values improvement percentage of DAOSNPS comparing with GQA, NEQA, OSNPS and AOSNPS.

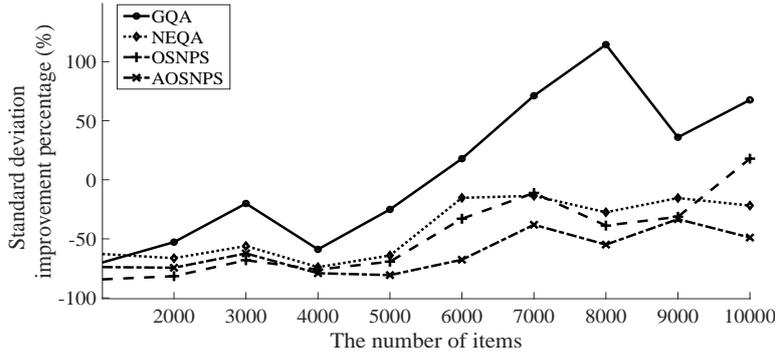


Fig. 5. Standard deviation improvement percentage of DAOSNPS comparing with GQA, NEQA, OSNPS and AOSNPS.

To strengthen the statistical analysis of experimental results, we perform the Holm-Bonferroni [56] for the five algorithms ($N_A = 5$) and ten problem instances ($N_p = 10$). The rank R_k ($k = 1, \dots, N_A$) assigned by each problem instance has been calculated (the score of the best algorithm is $N_A = 5$, the score of the second best algorithm is $N_A - 1, \dots$, the worst algorithm is $N_A - 4$). The rank R_k of each algorithm is the average value of all the problem instances. R_0 indicates the ranking of DAOSNPS. For the remaining $N_A - 1$ algorithms, the score z_k is calculated as follows:

$$z_k = \frac{R_k - R_0}{\sqrt{\frac{N_A(N_A+1)}{6N_p}}}. \quad (18)$$

By means of the z_k values, the corresponding cumulative normal distribution values p_k is

$$p_k = \frac{2}{\sqrt{\pi}} \int_{\frac{-z_k}{\sqrt{2}}}^{\infty} e^{-t^2} dt. \quad (19)$$

In this case, the level of confidence δ is set to 0.05. δ/k is used to compared with p_k . If p_k is greater than δ/k , the null-hypothesis is "Rejected" (the algorithms have statistically different performance), otherwise, the null-hypothesis is "Accepted". The Holm-Bonferroni procedure in Table 2 displays DAOSNPS has the best ranking over the other algorithms. In conclusion, DAOSNPS significantly outperforms NQEA, OSNPS and GQA.

Table 2. Holm-Bonferroni Procedure with DAOSNPS as reference ($R_0 = 5.0e + 00$)

	R_k	z_k	p_k	$\frac{\delta}{k}$	Test
AOSNPS	3.9e+00	-1.5556e+00	1.0617e-01	5.00e-02	Accepted
NQEA	3.1e+00	-2.6870e+00	6.3893e-03	2.50e-02	Rejected
OSNPS	2.0e+00	-4.2426e+00	1.9577e-05	1.67e-02	Rejected
GQA	1.0e+00	-5.6569e+00	1.3663e-08	1.25e-02	Rejected

4.3 Diversity Analysis of DAOSNPS, AOSNPS and OSNPS

The performance of the algorithm depends not only on the optimization results, but also on the diversity trend. Keeping good population diversity is helpful to enhance the exploration ability of the algorithm. In this subsection, we run OSNPS, AOSNPS and DAOSNPS to solve the knapsack problem with 5000 items. OSNPS, AOSNPS and DAOSNPS consist of $H = 50$ ESNPS, each of which has 5002 neurons (the number of subpopulations m , the number of neurons in each individual l and the number of individuals n in each subpopulation are set to 5, 5000 and 10, respectively).

In OSNPS, the learning probability p_j^a and the learning probability Δ are set randomly in the range of [0.05, 0.2] and [0.005, 0.02] as suggested in [46]. In AOSNPS, the learning probability p_j^a is the same as OSNPS, the mutation probability p_j^m and the consecutive maximum generations of consistence are 0.01 and 500 as suggested in [47], respectively. In DAOSNPS, the learning probability p_j^a , the mutation probability p_j^m and the consecutive maximum generations of consistence are the same value in [47], and migration interval M_i and migration number M_n are set 100 and 1, respectively. Hence, in this subsection, the performance of DAOSNPS compared with AOSNPS and OSNPS is analyzed from diversity (D_{qbw} , D_{qa} , D_{hbw} and D_{hm}) and convergence (G_{bf}) based on the above parameters.

The analysis involves the following five indicators from [57].

(1) G_{bf} : global optimal fitness convergence trend. A larger value of G_{bf} gives a better solution regard to a maximization optimization problem.

$$G_{bf} = \max \{ f_i(x), i = 1, \dots, n \}, \quad (20)$$

where $f_i(x)$ represents i th fitness function, n is the number of fitness function.

(2) D_{qbw} : binary distance between the best and worst binary individuals in a population. A larger value of D_{qbw} gives a hint of larger distance between the best and worst binary individuals.

$$D_{qbw} = \frac{1}{m} \sum_{j=1}^m \left| |b_{bj}|^2 - |b_{wj}|^2 \right|, \quad (21)$$

where $|b_{bj}|^2$ is binary value of the j th bit in the best binary individual; $|b_{wj}|^2$ is binary value of the j th bit in the worst binary individual; m is the number of bits in a binary individual.

(3) D_{qa} : average binary distance of all binary individuals in a population. A larger value of D_{qa} suggests a larger distance between each pair of binary individuals in a population.

$$D_{qa} = \frac{2}{n(n-1)} \sum_{i=1}^n \sum_{j=i+1}^n \left\{ \frac{1}{m} \sum_{k=1}^m (b_{ik}|^2 - |b_{jk}|^2) \right\}, \quad (22)$$

where $|b_{ik}|^2$ and $|b_{jk}|^2$ are binary values of the k th bit in the i th and j th binary individuals, respectively; m is the number of bits in a binary individual; n is the number of individuals in a population.

(4) D_{hbw} : Hamming distance between the best and worst binary individuals in a population. A larger value of D_{hbw} indicates more varieties between the best and worst binary individuals.

$$D_{hbw} = \frac{1}{m} \sum_{i=1}^m (b_{bi} \oplus b_{wi}), \quad (23)$$

where b_{bi} and b_{wi} are the i th bits in the best and worst binary solutions, respectively; m is the number of bits in a binary solution; the symbol \oplus represents the OR operator.

(5) D_{hm} : average Hamming distance of all binary individuals in a population. A larger value of D_{hm} indicates more varieties between each pair of binary individuals in a population.

$$D_{hm} = \frac{2}{n(n-1)} \sum_{i=1}^n \sum_{j=i+1}^n \frac{1}{m} \sum_{k=1}^m (b_{ik} \oplus b_{jk}), \quad (24)$$

where b_{ik} and b_{jk} are the k th bits in the i th and j th binary solutions, respectively; m is the number of bits in a binary solution; n is the number of individuals in a population; the symbol \oplus represents OR operator.

The trends of these five metrics in Figs. 6 - 10 are described for $m = 5000$ items for OSNPS, AOSNPS and DAOSNPS, respectively. OSNPS, AOSNPS and DAOSNPS have been run 12500 generations with the same early conditions, respectively. Fig. 6 displays that DAOSNPS have the better convergence ability than OSNPS and AOSNPS. The convergence speed of DAOSNPS is faster than OSNPS and AOSNPS in terms of the same number of generations. DAOSNPS not only has faster convergence speed, but also maintains better diversity shown in Figs. 7 - 10.

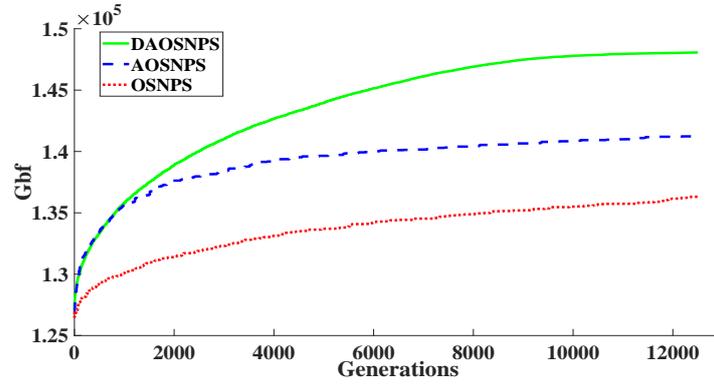


Fig. 6. Global best fitness convergence trend G_{bf} of OSNPS, AOSNPS and DAOSNPS

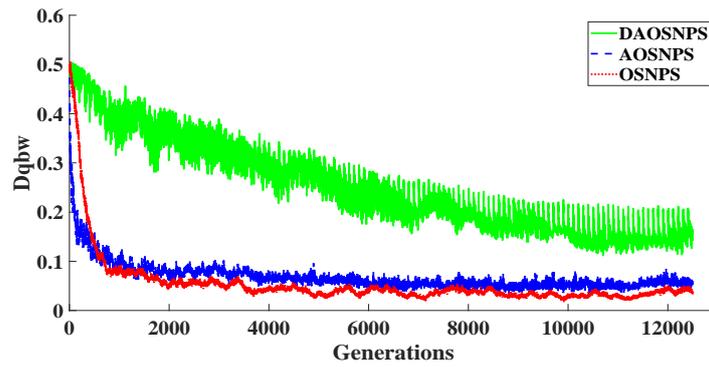


Fig. 7. Binary distance between the best and worst binary individuals in a population D_{qbw} : OSNPS and AOSNPS vs DAOSNPS, respectively

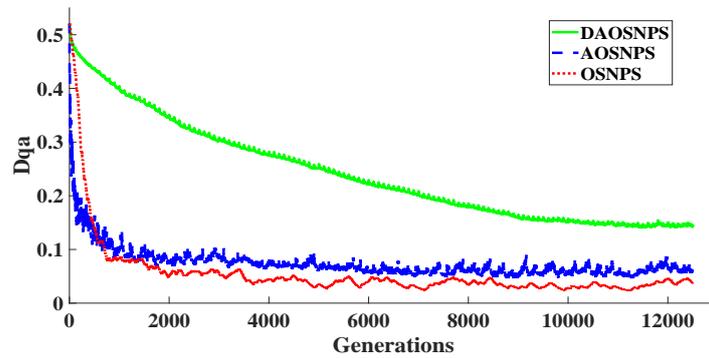


Fig. 8. Average binary distance among all binary individuals in a population D_{qa} : OSNPS and AOSNPS vs DAOSNPS, respectively

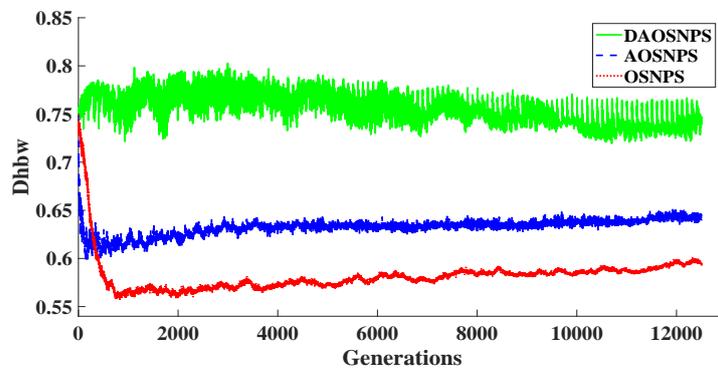


Fig. 9. Hamming distance between the best and worst binary individuals D_{hbw} : OSNPS and AOSNPS vs DAOSNPS, respectively

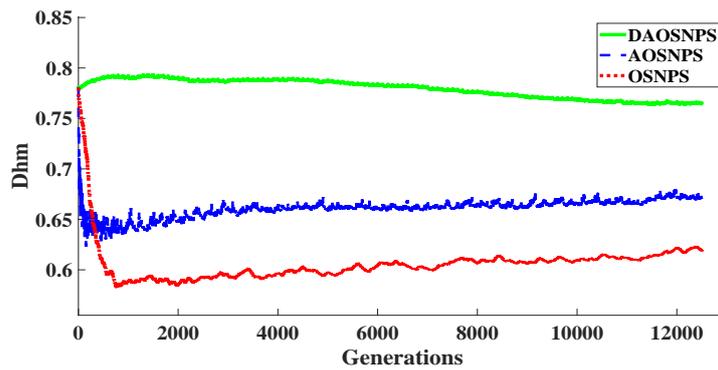


Fig. 10. Average Hamming distance D_{hm} : OSNPS and AOSNPS vs DAOSNPS, respectively

In summary, Figs. 6 - 10 show the following conclusions:

- (1) DAOSNPS has better convergence speed than OSNPS and AOSNPS, because the new learning rate Δ_{ijk}^d of DAOSNPS not only considers the best solutions searched, but also considers the population diversity. For example, in Fig. 6, when the number of generations is 12500, the best solutions of DAOSNPS, OSNPS and AOSNPS are 148068.149, 136309.146 and 141295.450, respectively.
- (2) In Figs. 7 - 10, when the number of generations is less than or equal to 2000, the population diversity of OSNPS and AOSNPS decreases significantly (When the number of generations is 2000, D_{qbw} and D_{qa} in OSNPS reduce to 0.051 and 0.020, respectively, and D_{qbw} and D_{qa} in AOSNPS reduce to 0.079 and 0.053, respectively), while that of DAOSNPS decreases slowly (When the number of generations is 2000, D_{qbw} and D_{qa} in DAOSNPS reduce to 0.430 and 0.371, respectively), which indicates that the distributed population structure is helpful to maintain the population diversity.
- (3) According to Figs. 6, 9 and 10, comparing with OSNPS and AOSNPS, DAOSNPS achieves a better balance between exploration and exploitation, which shows that DAOSNPS combining the distributed structure and the learning rate considering population diversity is competitive to the other two optimization approaches. For instance, in Fig. 6, when the number of generations is 12500, the values of D_{hbw} in DAOSNPS, OSNPS and AOSNPS are 0.742, 0.594 and 0.643, respectively, and the values of D_{hm} in DAOSNPS, OSNPS and AOSNPS are 0.765, 0.618 and 0.670, respectively.

5 Conclusions

This paper proposes a distributed adaptive spiking neural P system, called DAOSNPS, for solving the combinatorial optimization problems. A Guider algorithm including a distributed population structure and a adaptive learning rate considering population diversity in DAOSNPS is designed to improve the search.

Experimental results demonstrate that DAOSNPS is a feasible optimization approach for obtaining the approximate solutions of combinatorial optimization problems. Experimental data analysis shows that the introduction of an adaptive learning rate and a distributed population structure is helpful to improve the convergence and population diversity. The further improvement of DAOSNPS against OSNPS and AOSNPS implies that the design of the Guider, learning rate and/or evolution process is a proper way to enhance the performance.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61972324, 61672437, 61702428), the Sichuan Science and Technology Program (2021YFS0313, 2021YFG0133, 2020YJ0433, 2021YFN0104), Beijing Advanced Innovation Center for Intelligent Robots and Systems (2019IRS14) and Artificial Intelligence Key Laboratory of Sichuan Province (2019RYJ06).

References

1. Păun, Gh.: Computing with membranes. *Journal of Computer and System Sciences*, 61(1), 108-143(2000).
2. Pan, L., Păun, Gh., Zhang, G.: Foreword: Starting JMC. *Journal of Membrane Computing*, 1(1), 1-2(2019).
3. Zhang, G.: Membrane computing. *International Journal of Parallel, Emergent and Distributed Systems*, 36(1), 1-2(2021).
4. Manca, V., Bianco, L.: Biological networks in metabolic P systems. *Bio Systems*, 91(3), 489-498(2008).
5. Frisco, P., Gheorghe, M., Pérez-Jiménez, M.: *Applications of Membrane Computing in Systems and Synthetic Biology*. Communications of the Springer, (2014).
6. Ciobanu, G., Pérez-Jiménez, M., Păun, Gheorghe: *Applications of Membrane Computing*. Communications of the Springer, (2006).
7. Buiu, C., Vasile, C., Arsene, O.: Development of membrane controllers for mobile robots. *Information Sciences*, 187(1), 33-51(2012).
8. Wang, X., Zhang, G., Neri, F., Jiang, T., Zhao, J., Gheorghe, M., Ipate, F., Lefticaru, R.: Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots. *Integrated Computer Aided Engineering*, 23(11), 15-30(2016).
9. Wang, X., Zhang, G., Gou, X., Paul, P., Neri, F., Rong, H., Yang, Q., Zhang, H.: Multi-behaviors coordination controller design with enzymatic numerical P systems for robots. *Integrated Computer Aided Engineering*, 28(2), 119-140(2021).
10. Li, B., Peng, H., Luo, X., Wang, J., Song, X., Pérez-Jiménez, M., Riscos-Núñez, A.: Medical Image Fusion Method Based on Coupled Neural P Systems in Nonsampled Shearlet Transform Domain. *International Journal of Neural Systems*, 31(1), 1-17(2021).
11. Li, B., Peng, H., Wang, J.: A novel fusion method based on dynamic threshold neural P systems and nonsampled contourlet transform for multi-modality medical images. *Signal Process*, 178, 107793(2021).
12. Xue, J., Wang, Z., Kong, D., Wang, Y., Liu, X., Fan, W., Yuan, S., Li, D.: Deep ensemble neural-like P systems for segmentation of central serous chorioretinopathy lesion. *Information Fusion*, 65, 84-94(2021).
13. Zhang, G., Gheorghe, M., Wu, C.: A Quantum-Inspired elutionary Algorithm Based on P systems for Knapsack Problem. *Fundamenta Informaticae*, 87(1), 93-116(2008).
14. Zhu, M., Zhang, G., Yang, Q., Rong, H., Yuan, W., Pérez-Jiménez, Mario J.: P Systems Based Computing Polynomials With Integer Coefficients Design and Formal Verification. *IEEE Transactions on Nanobioscience*, 17(3), 272-280(2018).
15. Peng, H., Wang, J., Pérez-Jiménez, M., Wang, H., Shao, J., Wang, T.: Fuzzy reasoning spiking neural P system for fault diagnosis. *Information Sciences*, 235(1), 106-116(2013).
16. Wang, T., Zhang, G., Zhao, J., He, Z., Wang, J., Pérez-Jiménez, M.: Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. *IEEE Transactions on Power Systems*, 30(3), 1182-1194(2015).
17. Rong, H., Yi, K., Zhang, G., Dong, J., Paul, P., Huang, Z.: Automatic Implementation of Fuzzy Reasoning Spiking Neural P Systems for Diagnosing Faults in Complex Power Systems. *Complexity*, 2019, 1-16(2019).
18. Zhang, G., Pérez-Jiménez, M., Gheorghe, M.: *Real-Life Applications with Membrane Computing*. Emergence, Complexity and Computation (Springer,2017), (2017).
19. Zhang, G., Gheorghe, M., Pan, L., Pérez-Jiménez, M.: *Evolutionary Membrane Computing: A Comprehensive Survey and New Results*. *Information Sciences*, 279, 528-551(2014).
20. Orellana-Martín, D., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.: Minimal cooperation as a way to achieve the efficiency in cell-like membrane systems. *Journal of Membrane Computing*, 1(1), 1-2(2019).

21. Pan, L., Orellana-Martín, D., Song, B., Pérez-Jiménez, M.: Cell-like P systems with polarizations and minimal rules. *Theoretical Computer Science*, 816, 1-18(2020).
22. Kari, L., Rozenberg, G.: Tissue P systems with channel states. *Theoretical Computer Science*, 330(1), 101-116(2005).
23. Kari, L., Rozenberg, G.: Tissue-like P systems with evolutionary symport/antiport rules. *Information Science*, 378, 177-193(2017).
24. Song, B., Pan, L., Pérez-Jiménez, M.: Tissue P Systems with Protein on Cells. *Fundamenta Informaticae*, 144(1), 77-107(2016).
25. Ren, T., Cabarle, F., Adorna, H.: Generating context-free languages using spiking neural P systems with structural plasticity. *Journal of Membrane Computing*, 1(8), 161-177(2019).
26. Jiang, Y., Su, Y., Luo, F.: An improved universal spiking neural P system with generalized use of rules. *Journal of Membrane Computing*, 1(8), 270-278(2019).
27. Dong, J., Rong, H., Neri F., Yang, Q., Zhu, M., Zhang, G.: An Adaptive Memetic P System to Solve the 0/1 Knapsack Problem. *Communications of the IEEE*, 1-8(2020).
28. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking Neural P Systems. *Fundamenta Informaticae*, 71(2), 279-308(2006).
29. Xue, J., Wang, Y., Kong, D., Wu, F., Yin, A., Qu, J., Liu, X.: Deep hybrid neural-like P systems for multiorgan segmentation in head and neck CT/MR images. *Expert Systems with Applications*, 168, 114446(2021).
30. Liu, W., Wang, T., Zang, T., Huang, Z., Wang, J., Huang, T., Wei, X., Li, C.: A Fault Diagnosis Method for Power Transmission Networks Based on Spiking Neural P Systems with Self-Updating Rules considering Biological Apoptosis Mechanism. *Complexity*, 2020, 1-18(2020).
31. Wang, T., Zhang, G., Pérez-Jiménez, M.: Fuzzy Membrane Computing: Theory and Applications. *International Journal of Computers Communications and Control*, 10(6), 144-175(2015).
32. Sosík, P.: P systems attacking hard problems beyond NP: a survey. *Journal of Membrane Computing*, 1(3), 198-208(2019).
33. Zhang, G., Shang, Z., Verlan, S., Martínez-Del-Amor, M., Yuan, C., Valencia-Cabrera, L., Pérez-Jiménez, M.: An overview of hardware implementation of membrane computing models, 53(4), 1-38(2014).
34. Zhang, G., Pérez-Jiménez, M., Riscos-Núñez, A., Verlan, S., Hinze, T., Gheorghe, M.: *Membrane Computing Models: Implementations*, Springer, (2021).
35. Gheorghe, M., Neri, F., Zhang, G.: Introduction, *International Journal of Neural Systems*, 31(1), 1-2(2021).
36. Nash, A., Kalvala, S.: A P system model of swarming and aggregation in a Myxobacterial colony, *Journal of Membrane Computing*, 1(2), 103-111(2019).
37. Păun, Gh., Rozenberg, G., Salomaa, A.: *The Oxford Handbook of Membrane Computing*, Oxford University Press, Inc., New York, NY, USA, 2010, (2010).
38. Zhang, G., Gheorghe, M., Pan, L., Pérez-Jiménez, M.: Evolutionary membrane computing: A comprehensive survey and new results, *Information Sciences*, 279, 528-551(2014).
39. Zhang, G., Rong, H., Paul, P., He, Y, Neri, F., Pérez-Jiménez, M.: A Complete Arithmetic Calculator Constructed from Spiking Neural P Systems and its Application to Information Fusion, *International Journal of Neural Systems*, 31(1), 1-17(2021).
40. Yao, Z., Liang, H.: A variant of P systems for optimization, *Neurocomputing*, 72(4-6), 1355-1360(2009).
41. Zhang, G., Gheorghe, M., Li, Y.: A membrane algorithm with quantum-inspired subalgorithms and its application to image processing, *Natural Computing*, 11(4), 701-717(2012).
42. Zhang, G., Cheng, J., Gheorghe, M., Meng, Q.: A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems, *Applied Soft Computing*, 13, 1528-1542(2013).

43. Cheng, Y., Zhang, G., Wang, T.: Automatic Design of P Systems for Five Basic Arithmetic Operations within One Framework, *Chinese Journal of Electronics*, 2(23), 89-91(2014).
44. Ou, Z., Zhang, G., Wang, T., and Huang, X.: Automatic Design of Cell-like P Systems through Tuning Membrane Structures, Initial Objects and Evolution Rules, *International Journal of Unconventional Computing*, 9(5-6), 425-443(2013).
45. Dong, J., Michael, S., Zhang, G., Matteo, C., and Rong, H., Paul, P.: Automatic Design of Spiking Neural P Systems Based on Genetic Algorithms, *International Journal of Unconventional Computing*, 16(2-3), 201-216(2021).
46. Zhang, G., Rong, H., Neri, F., Pérez-Jiménez, M.: An optimization spiking neural P system for approximately solving combinatorial optimization problems, *International Journal of Neural Systems*, 24(5), 1440006(2014).
47. Zhu, M., Yang, Q., Dong, J., Zhang, G., Neri, F.: An Adaptive Optimization Spiking Neural P System for Binary Problems, *International Journal of Neural Systems*, 31(1), 2050054(2020).
48. Yang, S., Tinós, R.: A hybrid immigrants scheme for genetic algorithms in dynamic environments, *International Journal of Automation and Computing*, 4(3), 243-254(2007).
49. Yu, X., Tang, K., Yao, X.: An immigrants scheme based on environmental information for genetic algorithms in changing environments, *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2008*, June 1-6, 2008, Hong Kong, China, 1141-1147(2008).
50. Han, H., Kim, J.: Quantum-inspired evolutionary algorithm for a class of combinatorial optimization, *IEEE Transactions on Evolutionary Computation*, 6(6), 580-593(2002).
51. Han, H.: Quantum-inspired evolutionary algorithms with a new termination criterion, Hesion gate, and two-phase scheme, *IEEE Transactions on Evolutionary Computation*, 8(2), 156-169(2004).
52. Zhang, G.: Quantum-inspired evolutionary algorithms: a survey and empirical study, *Journal of Heuristics*, 17(3), 303-351(2011).
53. Han, H., Kim, J.: Genetic quantum algorithm and its application to combinatorial optimization problem, *Proceedings of the 2000 Congress on Evolutionary Computation*, (2002).
54. Gao, H., Xu, G., Wang, Z.: A Novel Quantum Evolutionary Algorithm and Its Application, *World Congress on Intelligent Control and Automation*, (2006).
55. Wilcoxon, F.: Individual comparison by ranking methods, *Biometrics*, 1(6), 80-83(1945).
56. Holm, S.: A simple sequentially rejective multiple test procedure, *Scandinavian Journal of Statistics*, 6(2), 65-70(1979).
57. Zhang, G., Cheng, J., and Gheorghe, M.: Dynamic behavior analysis of membrane-inspired evolutionary algorithms, *International Journal of Computers, Communications and Control*, 9(2), 227-242(2014).

Multi-parameter Optimization of Reducer Lubrication Based on Optimization Spiking Neural P Systems

Jianping Dong¹, Xingqiao Deng¹, Shisong Wang¹, Biao Luo², Huiling Feng¹, and Gexiang Zhang^{2,3} *

¹ School of Nuclear Technology and Automation Engineering,
Chengdu University of Technology, Chengdu 610059, China
master_djp@163.com, dengxingqiao19@cdut.edu.cn,
wsscdp@outlook.com, , 18215526091@163.com

² Research Center for Artificial Intelligence,
Chengdu University of Technology, Chengdu 610059, China
zhgxdylan@126.com, 15775960380@163.com

³ School of Control Engineering,
Chengdu University of Information Technology, Chengdu, 610225, China

Abstract. It is very difficult to improve the efficiency and accuracy of reducer lubrication since the limitation of traditional simulation method. In this paper, an optimization model with multiple parameters is first established to reflect the relationship between the churning loss and the optimized parameters of the Zero-backlash High Precision Roller Enveloping Reducer (ZHPRE). Then, an Optimization spiking neural P system (OSNPS) is applied to solve the multiple parameters optimization model. Finally, a simulation analysis (the semi-implicit moving particle method, MPS) is used to verify the correctness of the optimization results. Experimental results show that the multiple parameters optimization model and OSNPS are proved to be effective and accurate for solving the multiple parameters optimization problem of ZHPRE by MPS.

Keywords: Lubrication performance; high precision reducer; multi-parameter optimization; Optimization Spiking neural P system.

1 Introduction

The optimization of the designed parameters of the reducer is very important in the design and manufacture process. According to different requirements, the corresponding analysis method is adopted to get the optimal design parameter combination so as to improve the transmission performance of the reducer. The analysis methods can be divided into two categories: algorithm optimization [1,2,3,4,5] and simulation analysis [6,7,8,9,10]. For algorithm optimization, it is mainly to use the connection between the design goal and the design parameter to establish the corresponding objective function and the constraint

* Corresponding author.

range according to the actual demand, select the appropriate optimization algorithm, carry out the corresponding optimization analysis, and finally determine the required optimal parameters. Using the above algorithm can optimize and analyze multiple interconnected parameters fast. However, the accuracy of establishing the objective function is very important, which directly affects the reliability of the optimized parameters. For simulation optimization, it is mainly to use relevant simulation software to establish a corresponding simulation model, continuously analyze the optimized variables, and determine the best parameters. Compared with algorithm optimization, simulation analysis does not need to establish an objective function and it only needs to set corresponding boundary conditions for the working conditions, which improves the accuracy of the calculation results. However, the calculation efficiency is slightly insufficient in multiple parameter optimization. Due to these features, the two methods separately have many applications in the reducer's structure optimization and lubrication optimization.

In structural optimization, taking the reducer of the tillage machine as the research object, the minimum center distance of the reducer as the target, and the contact fatigue strength, bending fatigue strength, and oil film thickness ratio of the gear as the constraint conditions, the optimized designed parameters were obtained by He HB [11] et al., which greatly optimized the size and weight of the reducer case. These optimization algorithms were also employed in mechanism optimization. For example, Particle Swarm Optimization (PSO) and simulated annealing algorithms (SA) were used to conduct corresponding optimization analysis with the minimum weight of the reducer as the optimization objective [12]. After comparing the optimization results with the existing design, the algorithm can solve better design parameters. A two-phase evolutionary algorithm was adopted by Tudose L [13] et al. to comprehensively optimize the reducer's service life and overall weight for the two-phase transmission reducer. Using the response surface methodology, which based on the tooth surface modification method, the contact stress and transmission error in the gear transmission process was optimized by Korta J A [14] et al.. It can be observed that the optimized gear transmission performance has been significantly improved than before. Genetic algorithms were applied by Daoudi K [15], who conducted the planetary gear reducer optimization analysis to obtain better-designed parameters with a lightweight, a small center distance, and high efficiency.

In lubrication optimization, a numerical analysis model of gear oil injection lubrication with different transmission modes was established by Dai Y [16,17] et al.. By analyzing the lubrication conditions under different nozzle arrangement modes with this model, Dai Y finally determined the optimal nozzle arrangement mode. The moving particle semi-implicit (MPS) method was used by Deng X Q [18] et al. to establish a computational fluid dynamics (CFD) model for a worm gear reducer, analyzing the influence of different operating parameters, environmental parameters, and design parameters on the lubrication performance of the reducer. Meanwhile, Taguchi algorithm is used to carry

out corresponding optimization analysis [19] and the corresponding optimization design parameters are obtained. Based on the finite volume method (FVM), a reducer lubrication model was applied by Chen L Q [20] et al. to analyze the lubrication performance of the reducer under different lubricant volumes and gear speeds and use the response surface method to optimize it accordingly. By establishing a numerical analysis model for analyzing the influence of injection angle, position, and distance on lubrication performance, Wang Y Z [21] et al. determined a set of good injection lubrication parameters.

It can be seen from the above analysis that there are a large number of cases in the structure optimization of the reducer, using algorithms or simulation analysis methods to optimize the optimization target. In contrast, in the lubrication optimization of the reducer, most of them are based on one or two optimization goals, using the corresponding CFD simulation software to establish a simulation model for a large number of simulation analyses and optimization. For the distribution state of the internal lubrication oil constantly changes during the reducer operation, we have not found any relevant literature on the optimization and analysis of lubrication parameters of the reducer using the above algorithm. As a result, it is difficult to use a specific function (objective function) to describe the change of this lubricant behavior. If a specific objective function cannot be established, it is impossible to perform algorithm optimization analysis on the lubrication problem of the reducer. However, during the manufacture and use of the reducer, its lubrication performance is simultaneously affected by multiple parameters. Traditional CFD simulation analysis methods are challenging to perform high-efficiency analysis for multi-parameter problems. For this reason, it is very urgent and necessary to propose an effective analysis method for multi-parameter optimization of the lubrication performance.

Membrane computing, a branch of natural computing, is a computing model abstracted from the structure and the functioning of the biological cells, organs and colonies of bacteria [22,23,24]. Membrane computing was initiated at the end of 1998 by Gh. Păun. In 2003, membrane computing, called membrane systems or P systems, is listed by Thompson Institute for Science Information (ISI) as an emerging research of computer science. Currently, P systems are divided into three basic types depending on the membrane structure: cell-like P systems [25,26], tissue-like P systems [27,28,29] and neural-like P systems [30,31,32].

In recent years, the research on neural-like P systems mainly focused on spiking neural P systems (SNPS), which were introduced by Ionescu et al. in [33]. SNPS are a class of distributed and parallel computing devices which are inspired by the way neurons communicate by means of electrical impulses (spikes). Optimization spiking neural P system (OSNPS), is proposed by Zhang et al. in [34,35], is one of SNPS to solve optimization problems without the aid of evolutionary operators. An OSNPS includes a family of extended spiking neural P system (ESNPS) and a guider algorithm adjusting rule probabilities, where an ESNPS consists of the probabilistic selection of evolution rules and multiple output neurons. The proposed future work in [34] pointed out that

OSNPS can be used to solve various application problems, such as fault diagnosis of electric power systems and optimization of mechanical parameters.

Therefore, this paper uses optimization of lubrication parameters of reducer based on OSNPS [34,35] combined with simulation analysis for churning losses during lubrication. This method shortens the analysis time as much as possible and improve the accuracy of the optimization results at the same time. It takes the churning power losses as the objective function and uses the parameters that affect the churning power losses as variables. Considering the influence of multi-variable coupling, the corresponding churning losses calculation model (the objective function) is constructed using dimensional analysis method. Finally, the OSNPS is used to efficiently optimize multiple parameters and use the relevant parameters obtained by the optimization algorithm to establish the corresponding CFD model. Using the relevant parameters obtained by the optimization algorithm to establish the corresponding CFD model simulation analysis verified the accuracy of the optimization results. It captured the corresponding lubricant flow field distribution, which effectively solves the multivariate analysis of the existing reducer lubrication optimization problem.

The remainder of this paper is organized as follows. Section 2 takes the ZHPRER as an example to establish the corresponding churning losses objective function and the constraint range of the parameters to be optimized; Section 3 constructs an optimization algorithm based on ROSNPS; Section 4 solves the corresponding optimization parameters and uses MPS to build a simulation analysis model to verify its accuracy, the distribution of lubricating flow field under optimal parameters is also discussed in this part; Section 5 is the conclusion.

2 The ZHREPR is Built for the Churning Losses Model

2.1 Traditional Single Gear Churning Losses Model

In a closed-type gear transmission, the churning loss of the reducer is simultaneously affected by the geometric parameters of the gear, the operating parameters during transmission and the relevant parameters of the lubricant oil [24]. The relationship between C_{ch} and its influence parameters is shown in (1).

$$C_{ch} = f(m, D_P, b, v, \rho, h, V_0, g, \Omega), \quad (1)$$

where, m is the gear module; D_P is the gear fractional round diameter; b is the gear tooth width; v is the oil motion viscosity; ρ is the oil density; h is the deep of the gear immersed in oil; V_0 is the volume of the oil; g is gravity acceleration; Ω is gear speed.

Further research on the churning losses of a single gear shows that: the churning power losses of a single gear can be equivalent to the churning power

losses of a disc with the same diameter [36]. the churning losses of the single tooth is calculated by the dimensional analysis method [37] as follows:

$$C_{ch} = \frac{\rho}{16} S_m \Omega^2 D_P^3 [\psi_1 (\frac{m}{D_P})^{\psi_2} (\frac{b}{D_P})^{\psi_3} (\frac{h}{D_P})^{\psi_4} (\frac{V_0}{D_P})^{\psi_5} Re^{\psi_6} F_r^{\psi_7}], \quad (2)$$

where, S_m is the area of the gear immersed in lubricating oil; $\psi_1 - \psi_7$ is the coefficient determined by the analysis of the experimental results; $Re = \frac{\Omega D_P^2}{4v}$ is the Reynolds number; $F_r = \frac{\Omega^2 D_P}{2g}$ is the Froude number.

The specific values of $\psi_1 - \psi_7$ are shown in Table 1:

Table 1. The specific values of $\psi_1 - \psi_7$

	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6	ψ_7
Medium-low speed $Re_c < 6000$	1.366	0	0	0.45	0.1	-0.21	-0.6
High speed $Re_c > 9000$	3.644	0	0.85	0.1	-0.35	0	-0.88

When $Re_c < 6000$, it is medium-low speed. When $Re_c > 9000$, it is high speed. When $6000 < Re_c < 9000$, the churning losses are determined by interpolation between the medium-low speed and high-speed calculation results, as shown in 3.

$$Re_c = \frac{\Omega D_P b}{2v} \quad (3)$$

2.2 The ZHREPR Churning Losses Model

When the worm is in the upper position, only the worm gear is directly involved in churning losses when the worm is on top of the worm gear in the ZHPRER[27], and its working condition is similar to that of the single gear oil mixing loss, so the calculation formula of the churning losses is similar to the churning losses of the single gear. Compared with the churning losses of the single gear, in the ZHPRER, the roller diameter D and the index circle diameter of the worm gear d_2 can be equivalent to the gear width b and the gear index circle diameter D_p , respectively. According to (2), (3) and Table 2, the churning losses of ZHPRER is calculated as follows:

$$C_{ch} = \begin{cases} \frac{\rho}{16} S_m \Omega^2 d_2^3 [1.366 (\frac{h}{d_2})^{0.45} (\frac{V_0}{d_2^3})^{0.1} (\frac{\Omega d_2^2}{4v})^{-0.21} (\frac{\Omega^2 d_2}{2g})^{-0.6}] & Re_c < 6000 \\ \frac{\rho}{16} S_m \Omega^2 d_2^3 [3.644 (\frac{D}{d_2})^{0.85} (\frac{h}{d_2})^{0.1} (\frac{V_0}{d_2^3})^{-0.35} (\frac{\Omega^2 d_2}{2g})^{-0.88}] & Re_c > 9000 \end{cases} \quad (4)$$

Because the gear width D_p and the gear width b are replaced by the index circle diameter of the worm gear d_2 and the roller diameter D , respectively, the corresponding Re_c is expressed as follows:

$$Re_c = \frac{\Omega d_2 D}{2v}. \quad (5)$$

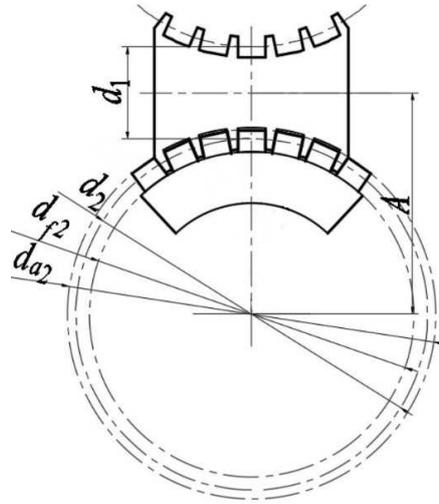


Fig. 1. Parameters of worm gear

In (4), the calculations of S_m and V_0 are related to the design parameters of the worm gear and the cavity of the reducer. Fig.1 illustrates a schematic diagram of the worm gear parameters, where A is the center distance, d_1 denotes the reference diameter of the worm throat part, d_{a2} represents the tip diameter, and d_{f2} means the root diameter. As shown in the Fig. 1, the parameters have the following geometric relations:

$$\begin{cases} m = \frac{d_2}{i} \\ d_1 + d_2 = 2A \\ d_{a2} = d_2 + 2h_a^* m \\ d_{f2} = d_2 - 2(h_f^* + c^*) m \end{cases}, \quad (6)$$

among them, i is the transmission ratio, h_a^* represents the addendum coefficient of the worm gear, which is usually 0.5, the root coefficient h_f^* is usually 0.5, and the value of c^* generally is 0.2.

Fig. 2 interprets the design parameters of the cavity of the reducer, where b is the width of the cavity, d_c represents the diameter, α and β is the geometric angle required in the process of deriving S_m and V_0 , respectively.

According to the parameters of the worm gear and the cavity of the reducer and the corresponding geometric calculations, the expressions of S_m and V_0 can be approximately solved, as shown in 7.

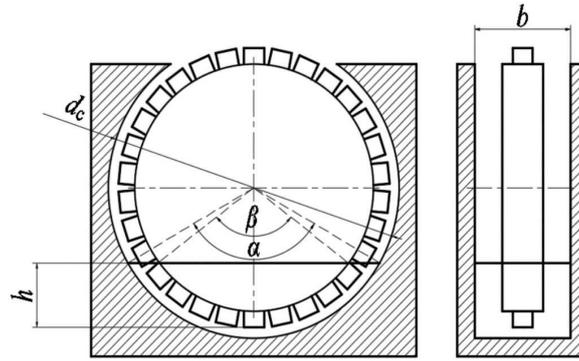


Fig. 2. Gear box parameters indication

$$\begin{cases} S_m \approx \frac{\beta b d_{f2}}{2} + \frac{15\beta D(d_{a2} - d_{f2})}{2} \\ V_0 \approx \frac{b d_c^2}{8}(\alpha - \sin\alpha) - \frac{b d_{f2}^2}{8}(\beta - \sin\beta) - \frac{15\beta D^2(d_{a2} - d_{f2})}{8} \\ \alpha = 2\arccos\frac{d_{a2} - 2h}{d_c} \\ \beta = 2\arccos\frac{d_{a2} - 2h}{d_{f2}} \end{cases} \quad (7)$$

For the immersion depth h of the worm gear, generally speaking, when the worm gear is placed under the state, the immersion depth requires at least one tooth height, and the highest depth does not exceed the central axis of the worm gear. From this, the range of values can be calculated through geometric relations:

$$\frac{d_{a2} - d_{f2}}{2} \leq h \leq \frac{d_{a2}}{2}. \quad (8)$$

According to the using standard of worm gear lubricating oil, the range of lubricating oil kinematic viscosity ν and the range of lubricating oil density ρ are $60 \sim 200$ cst and $850 \sim 900 \text{kg/m}^3$, respectively. The speed Ω is set to $10 \sim$

100 rpm (the speed of the worm gear Ω is usually slow due to the characteristics of the large transmission ratio).

For the values of other influencing parameters, it is necessary to give a more specific size of the reducer to determine. Therefore, according to the design parameters of the reducer in the literature [18], it is assumed that the center distance of the worm gear $A = 80\text{mm}$, the transmission ratio $i = 30$, and the required parameters are within a reasonable range of variation (the worm gear reference diameter d_2 is $118.5 \sim 138.5\text{mm}$, the roller diameter D is $5 \sim 12\text{mm}$, the oil cavity diameter d_c is $d_{a2} + 40 \sim 175\text{mm}$ and the oil cavity width b is $D + 30 \sim 52.5\text{mm}$ to 52.5mm).

Taking the minimum value of the churning losses power function in (4) as the objective function and the variation range of the parameters to be optimized as the constraint function, the specific optimization process will be introduced in Section 3.

3 Optimization of Lubrication Parameters of Reducer Based on OSNPS

OSNPS in [34] is introduced to solve the objective function obtained above. Because the outputs of OSNPS are a binary spiking trains, binary spiking trains are converted to decimal parameters. Here, OSNPS is described in Subsection 3.1, and the optimal solution is proposed in Subsection 3.1.

3.1 Optimization Spiking Neural P Systems

The OSNPS is constructed from the probability selection matrix, pulse train, and multiple ESNPSs from multiple neurons. The structure of the specific ESNPS is shown in Fig. 3. And the OSNPS is composed of multiple parallel ESNPSs and Guider algorithm, as shown in Fig. 4.

An ESNPS [34], of degree $(m, 2)$, is a

$$\Pi = (O, \sigma_1, \dots, \sigma_{m+2}, \text{syn}, I_0), \quad (9)$$

where:

- (1) $O = \{a\}$ is the singleton alphabet (a is called *spike*);
- (2) $\sigma_1, \dots, \sigma_{m+2}$ represent $m+2$ neurons. Its form is $\sigma_i = (1, R_i, P_i), 1 \leq i \leq m+2$.
 - (a) 1 means that there is only one initial spike in the $\sigma_i, 1 \leq i \leq m+2$.
 - (b) $R_i = \{r_i^1, r_i^2\}$ is a set of rules with the firing rule and the forgetting rule, where $r_i^1 = \{a \rightarrow a\}$ is firing rule and $r_i^2 = \{a \rightarrow \lambda\}$ is forgetting rule.
 - (c) $P_i = \{p_i^1, p_i^2\}$ represents two probabilities, where the selection probabilities of r_i^1 and r_i^2 are p_i^1 and p_i^2 , respectively, and satisfy $p_i^1 + p_i^2 = 1$.
 - (d) $\sigma_1, \dots, \sigma_m$ are the m working neurons, $1 \leq i \leq m+2$. Each neuron outputs 0 or 1 at each step.

- (e) σ_{m+1} and σ_{m+2} are the two supply neurons. σ_{m+1} and σ_{m+2} supply spikes to other $m + 1$ neurons other than themselves.
- (3) syn is the relationship of neurons ($\sigma_1, \dots, \sigma_m, \sigma_{m+1}$ and $\sigma_{m+2}, \sigma_1, \dots, \sigma_m, \sigma_{m+2}$ and σ_{m+1}).
- (4) $I_0 = \{1, 2, \dots, m\}$ is a finite set of *output* neurons.

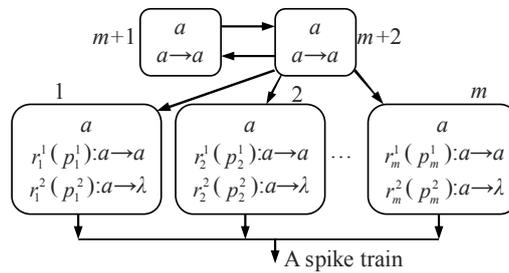


Fig. 3. Logical and functioning scheme of the specific ESNPS

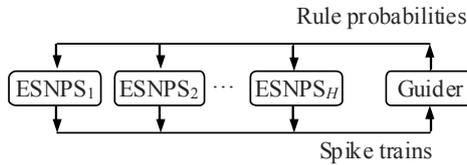


Fig. 4. Structure of OSNPS

In [34], an ESNPS contains the subsystem consisting of neurons σ_{m+1} and σ_{m+2} , which are supplier of spikes to neurons $\sigma_1, \dots, \sigma_m$. σ_{m+1} and σ_{m+2} are the same neurons, each of which fires at each moment of time and sends a spike to each of neurons $\sigma_1, \dots, \sigma_m$. Each of neurons $\sigma_1, \dots, \sigma_m$ performs the firing rule r_i^1 by probability p_i^1 and the forgetting rule r_i^2 by probability p_i^2 , $i = 1, \dots, m$. Thus, this system outputs a spike train consisting of 0 and 1 at each time unit. The outputted spike train is controlled by adjusting the probabilities p_i^1, \dots, p_i^m . Hence, an adjustment strategy to adjust probabilities p_i^1, \dots, p_i^m by introducing a family of ESNPS is presented in the following subsection.

Algorithm 1 The Guider algorithm of OSNPS [34].

Input: Spike train T_s , probabilities p_j^a , learning rate Δ , number of ESNPS H and the current best solution $\mathbf{x} = (x_1, x_2, \dots, x_m)$ of length m

```

1: Rearrange  $T_s$  as matrix  $\mathbf{P}_R$ 
2:  $i = 1$ 
3: while ( $i \leq H$ ) do
4:    $j=1$ 
5:   while ( $j \leq m$ ) do
6:     if ( $rand() < p_j^a$ ) then
7:        $k_1, k_2 = ceil(rand() * H), k_1 \neq k_2 \neq i$  and correct  $\mathbf{x}_{k_1}$  and  $\mathbf{x}_{k_2}$ 
8:       if ( $f(x^{k_1}) > f(x^{k_2})$ ) then
9:          $x_j = x_j^{k_1}$ 
10:      else
11:         $x_j = x_j^{k_2}$ 
12:      end if
13:      if ( $x_j == 1$ ) then
14:         $p_{ij}^1 = p_{ij}^1 + \Delta$ 
15:      else
16:         $p_{ij}^1 = p_{ij}^1 - \Delta$ 
17:      end if
18:      else
19:        if ( $x_j == 1$ ) then
20:           $p_{ij}^1 = p_{ij}^1 + \Delta$ 
21:        else
22:           $p_{ij}^1 = p_{ij}^1 - \Delta$ 
23:        end if
24:      end if
25:      if ( $p_{ij}^1 > 1$ ) then
26:         $p_{ij}^1 = p_{ij}^1 - \Delta$ 
27:      else
28:        if ( $p_{ij}^1 < 0$ ) then
29:           $p_{ij}^1 = p_{ij}^1 + \Delta$ 
30:        end if
31:      end if
32:       $j = j + 1$ 
33:    end while
34:     $i = i + 1$ 
35:  end while

```

Output: Rule probability matrix \mathbf{P}_R

The detailed execution process described by the pseudo-code in Algorithm 1 has been described in [34] and will not be repeated here. At the same time, the literature [34] also pointed out that the director can be revised to adapt to and solve different types of optimization problems. The output directory is a probability matrix $\mathbf{P}_R = [p_{ij}^1]_{H \times m}$, which is composed of H ESNPS ignition

probabilities to form a matrix of H rows and m columns, and its form is shown in (10) .

$$\mathbf{P}_R = \begin{pmatrix} p_{11}^1 & p_{12}^1 & \cdots & p_{1m}^1 \\ p_{21}^1 & p_{22}^1 & \cdots & p_{2m}^1 \\ \vdots & \vdots & \ddots & \vdots \\ p_{H1}^1 & p_{H2}^1 & \cdots & p_{Hm}^1 \end{pmatrix}. \quad (10)$$

3.2 Optimization of Lubrication Parameters of Reducer Based on OSNPS

In order to illustrate the process of OSNPS to the problem of parameters optimization of lubrication, the sketch map is depicted in Fig. 5.

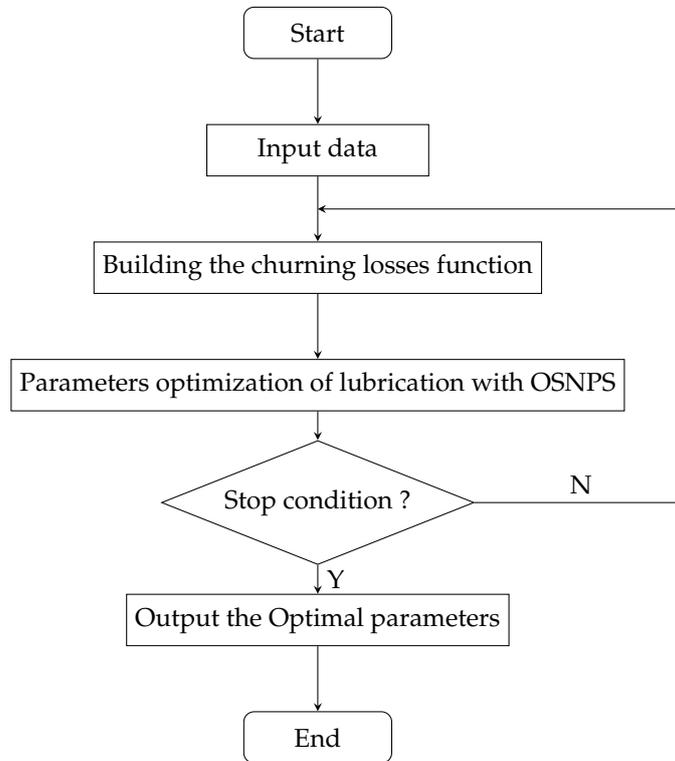


Fig. 5. The sketch map of parameters optimization based on OSNPS.

The optimization of lubrication parameters is addressed by means of the following step-by-step procedure.

Step 1: Input data. To start the method, parameters of OSNPS (the learning probability p_j^a , the learning rate Δ and population size are set the range of $([0.005, 0.02])$, the range of $([0.05, 0.2])$ and 50 ($H = 50$)). The basic parameters of the churning losses function (the oil density ρ ; the area of the gear immersed in lubricating oil S_m ; the gear speed Ω ; the pitch circle diameter of worm gear d_2 ; the deep of the gear immersed in oil h ; the volume of the oil V_0 ; the kinematic viscosity of lubricating oil v ; the gravity acceleration g ; the roller diameter D ; the Reynolds number Re_c) are set as follows:

Step 2: Initialization parameters. According to Subsection 2.2, S_m and V_0 are calculated by (7), and Re_c is calculated by (5). The other parameters (the oil density ρ , the gear speed Ω , the pitch circle diameter of worm gear d_2 , the deep of the gear immersed in oil h , the kinematic viscosity of lubricating oil v and the roller diameter D) are expressed in binary with different lengths according to the actual constraints.

Step 3: Building the fitness function. An objective function (the churning losses function) is established by initialization parameters, is shown in (4).

Step 4: Parameters optimization with OSNPS. Perform OSNPS to produce and update spike trains (including a series of parameters) to find the minimum value of (4).

Step 5: Stop condition. The optimization process is terminated when either reaching the maximum iterations or concluding that no better solution would appear in the following iterations.

Step 6: Output the Optimal parameters. The spike trains corresponding to the minimum value of (4) is outputted.

4 Optimization Results Analysis and Simulation Verification

In order to check the correctness of the objective function and OSNPS, OSNPS is used to optimize the parameters and calculates the corresponding churning losses. MPS is used to establish the corresponding CFD modeling according to optimized parameters to verify the accuracy of the objective function and OSNPS. Optimization results analysis and simulation verification are described in Subsection 4.1 and Subsection 4.2, respectively.

4.1 Optimization Results Analysis

In this subsection, an OSNPS (the learning probability p_j^a ($j = 1, 2, \dots, m$) and the range of the learning rate Δ is $[0.05, 0.2]$, respectively) consisting of $H = 50$ ESNPS, each of which has a certain number of neurons such as 69 (the number of bits of ρ , Ω , d_2 , h , v and D are 14, 11, 9, 12, 12 and 9, respectively), is used to solve the optimal parameters corresponding to the minimum value C_{ch} in 4. In other words, the framework of OSNPS is applied to solve the optimization problem, in which 69 neurons represent that each ESNPS contains 67 output neurons and two auxiliary neurons. Therefore, the overall framework is the same as Fig.3, except that the value of m is 67. All experiments are implemented

on the platform Python and on a work station with Intel i7 2.93 GHz processor, 32GB RAM and Windows 10.

According to the actual constraints, the actual value range of each parameter is as follows:

- (1) ρ : the range of the lubricating oil density ρ is $850 \sim 900 \text{kg/m}^3$.
- (2) Ω : the range of the gear speed Ω is $0 \sim 7000 \text{rpm}$.
- (3) d_2 : the range of pitch circle diameter of worm gear d_2 is $118.5 \times 10^{-3} \sim 138.5 \times 10^{-3} \text{m}$.
- (4) h : the range of the deep of the gear immersed in oil h is $7 \times 10^{-3} \sim 66.25 \times 10^{-3} \text{m}$.
- (5) ν : the range of the kinematic viscosity of lubricating oil ν is $60 \times 10^{-6} \sim 200 \times 10^{-6} \text{m}^2/\text{s}$.
- (6) g : the value of gravity acceleration g is 9.8m/s^2 .
- (7) D : the range of the roller diameter D is $5 \times 10^{-3} \sim 12 \times 10^{-3} \text{m}$.

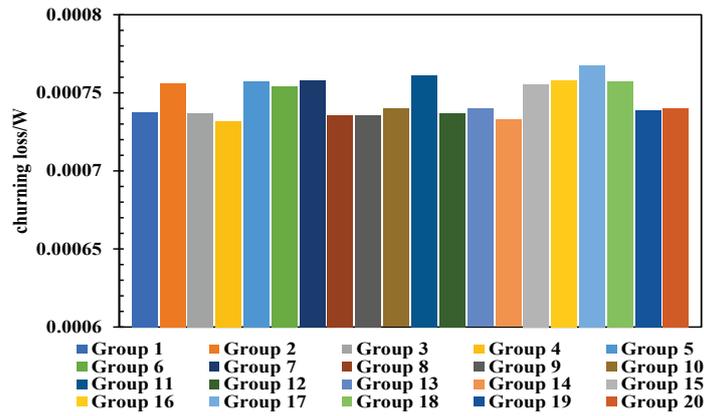


Fig. 6. The churning losses of the ZHPRER obtained by 20 independent optimizations

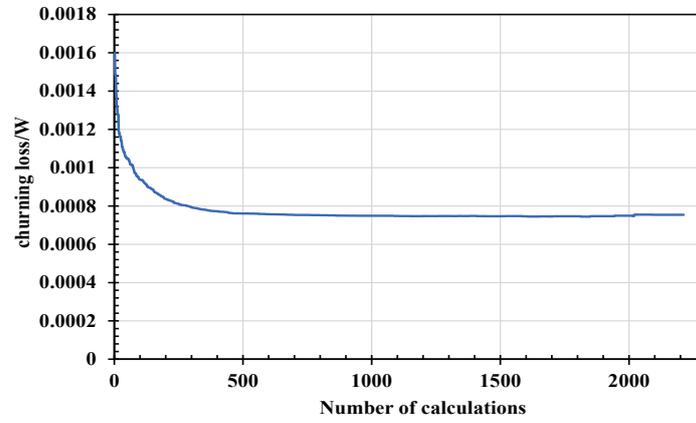


Fig. 7. Trend variation of average value of churning power losses for 20 independent optimizations

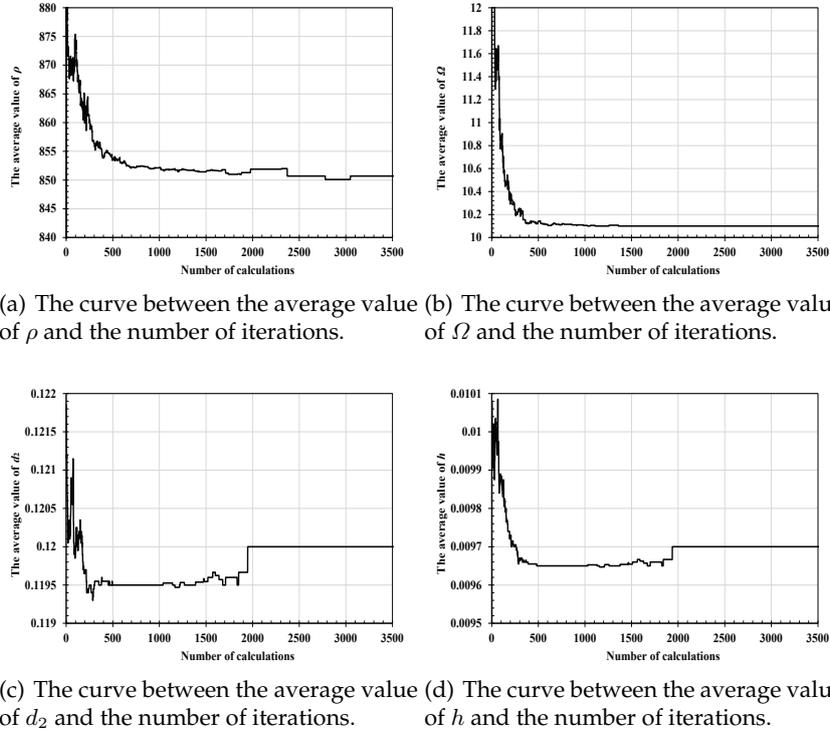
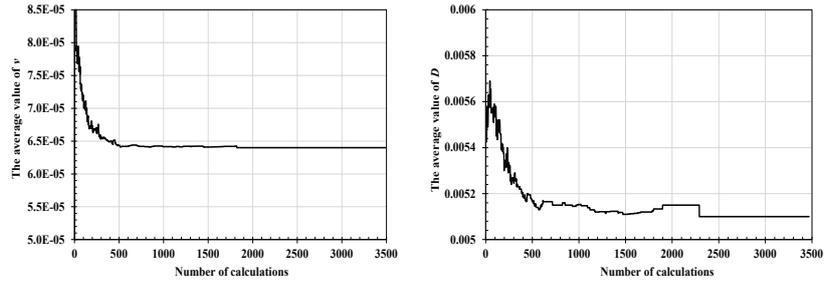


Fig. 8. The curve between the average value of ρ , ω , d_2 , and h and the number of iterations.

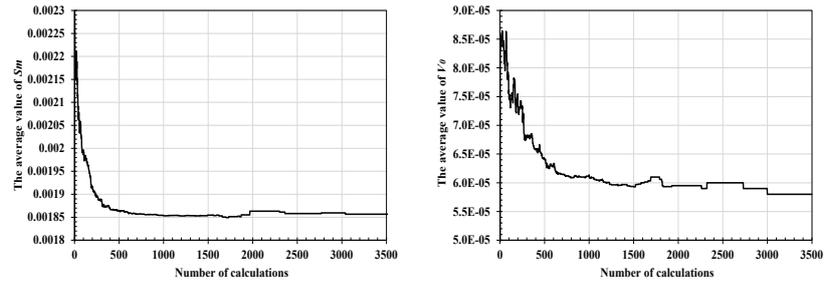
In Figs. 6-7, we use the optimization process in Subsection 3.2 to perform 20 independent optimizations and solve the objective function (4) under the actual constraints. Experimental results show that the method is feasible and effective for solving the problem of the churning loss of the **ZHPRER**. According to Fig. 6, the mean value and the standard deviation of the churning loss of the **ZHPRER** are $0.00074w$ and 1.120×10^{-5} , respectively, which indicates that OSNPS for solving the problem of the churning loss of the **ZHPRER** is stable relatively. In Fig. 7, when the number of iteration is greater or equal to 500, OSNPS is starting to converge, which destrate that OSNPS for solving the problem of the churning loss of the **ZHPRER** is fast.

In order to illustrate furthmore the details of each parameter in the optimization process, Figs. 8-9 show that the curve between the average value of each parameter and the number of iterations. When the churning loss value is constant for 500 times, algorithm stop and output the parameter values. Each parameter is convergence (such as ρ is constant before 3047, Ω is constant before 1581, d_2 is constant before 1946, h is constant before 1936, v is constant before

1819 and D is constant before 1896), which indicates that OSNPS is useful for the churning losses model in Subsection 2.2. The final optimization parameters are shown in Table 2.



(a) The curve between the average value of v and the number of iterations. (b) The curve between the average value of D and the number of iterations.



(c) The curve between the average value of S_m and the number of iterations. (d) The curve between the average value of V_0 and the number of iterations.

Fig.9. The curve between the average value of v , D , S_m , and V_0 and the number of iterations.

Table 2. Optimized parameters and churning losses

$\rho/\text{kg}/\text{m}^3$	v/cst	d_2/m	D/m	Ω/rpm	h/m	S_m/m^2	V_0/m^3	C_{ch}/w
850.7	6.4×10^{-5}	0.12	0.00515	10.1	0.0097	0.00185	5.8×10^{-5}	0.00074

4.2 Simulation Verification

In order to verify the accuracy of the optimization results, a CFD model is established for verification and analysis. Because the tooth surface of the worm

gear is a complex curved surface and the traditional mesh method is challenging to guarantee the accuracy of the mesh, so MPS method [39] of meshless and incompressible flow is used in this article. The accuracy of the analysis by using MPS method mainly depends on the quality of the model and the number of particles. All the simulations involved in this subsection is performed on professional graphics workstations (graphics card: NVIDIA Tesla K80, CPU: Intel 12-core 24-thread, max. RAM capacity: 32 GB).

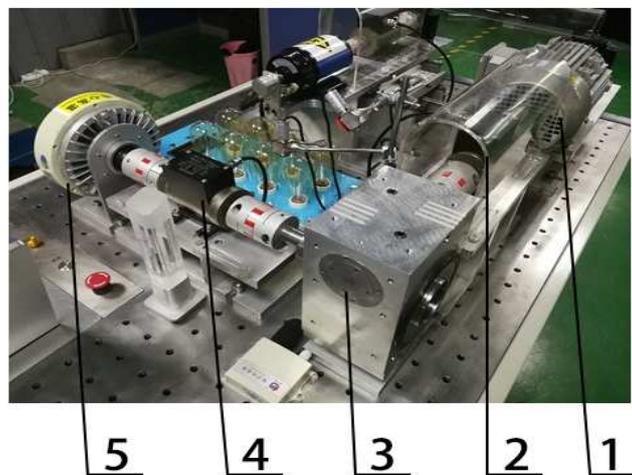


Fig. 10. Picture of experimental setup: (1) motor; (2) sensor for input torque; (3) speed reducer ; (4) sensor for output torque; (5) loader .

The lubrication analysis of the ZHPRER using the MPS method has been thoroughly researched and verified [18,19,38]. Fig. 10 is the experimental setup for verification, and Fig. 11 compares the trajectory of oil splash between experimental and simulation analysis. It can be seen from the position of oil accumulation and the splash that the two have a good consistency, which shows the effectiveness of the simulation.

Since the size of the 3D model studied in this article is similar to the size of the model in the reference [18], the particle diameter in the simulation is set to 0.7mm to ensure the accuracy of the simulation according to the setting parameters in the relevant literature. Secondly, simulation duration is set to 5s to obtain the flow field parameters in a relatively stable state. Furthermore, in order to avoid the sudden change of the initial speed and cause the turbulence of the flow field, $0 \sim 0.5s$ and $0.5 \sim 5s$ are the accelerating rotation stage and the uniform rotation stage, respectively.

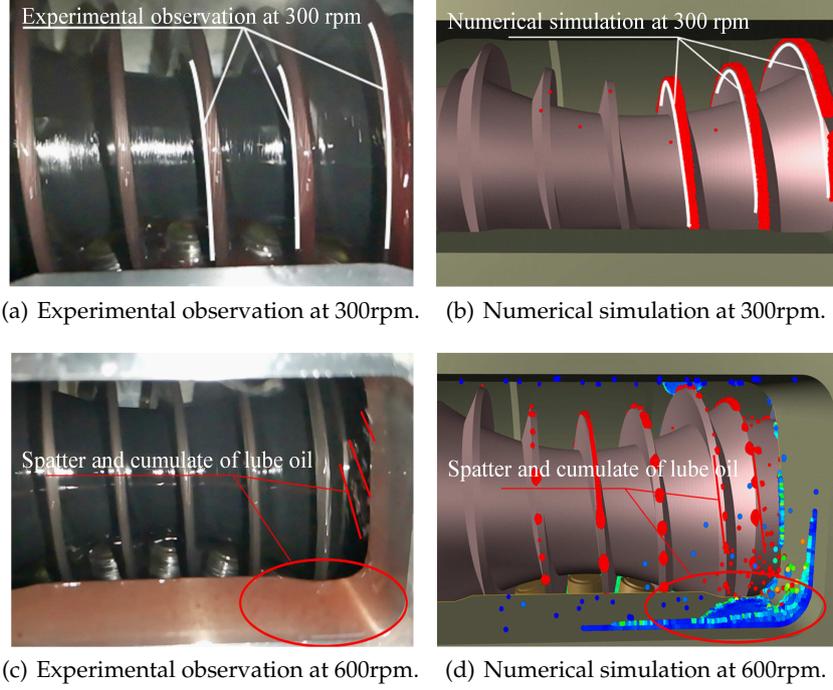


Fig. 11. Comparison of experimental and simulation results on lubricant distribution.

In addition, because the definition of the initial step size Δt is significant in the MPS method, the initial step size can limit the maximum step size needs to be calculated before the simulation. If Δt is too large, the results are not convergence. If Δt is too small, the simulation time lengthens. Δt is defined as:

$$\Delta t = \min(\Delta t_{init}, \frac{C_{max}l_0}{u_{max}}, \frac{d_i l_0^2}{2(v + v_{max})}), \quad (11)$$

where, l_0 is the particle size, u_{max} is the maximum particle velocity, C_{max} is the Courant coefficient, set to 0.2. The detailed introduction of the Courant number is described in [40]. d_i is the diffusion coefficient; set to 0.2. v is the particle kinetic viscosity coefficient. v_{max} is the particle's maximum kinetic viscosity coefficient. Δt_{init} refers to the initial time step. $\frac{C_{max}l_0}{u_{max}}$ is the time step calculated by CFL conditions [40]. $\frac{d_i l_0^2}{2(v+v_{max})}$ is to ensure the stability of the viscosity.

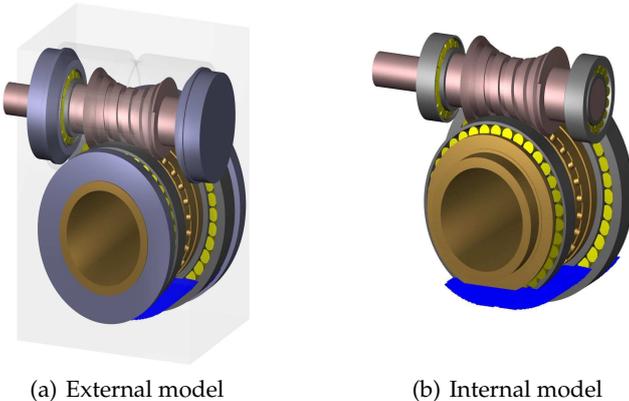


Fig. 12. Initialflow field model

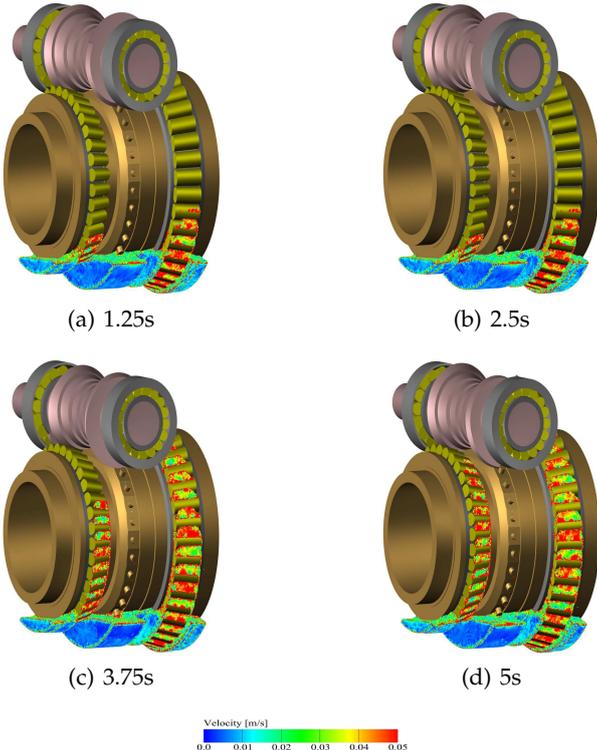


Fig. 13. Velocity flow field at different moments

According to the optimized geometric parameters in Table 3, in order to ensure the high accuracy of the simulation results, the internal characteristics of the reducer are not simplified in the modeling process, except for the installation hole, keyway and chamfer. The model as shown in Fig. 12.

Fig. 13 shows the velocity flow field distribution at 1.25s, 2.5s, 3.75s, and 5s, respectively. In order to facilitate the observation of the velocity in each area, the maximum velocity is set to 0.05m/s. It can be seen from the Fig. 13 that as the reducer rotates, the lubricating oil at the worm gear bearing is also driven so that the worm gear bearing is lubricated. However, it can be seen from the Fig. 13 that only a small amount of lubricating oil adheres to the roller surface of the worm gear. There are two reasons for this phenomenon. First of all, in order to minimize the power losses, the corresponding roller diameter, speed and lubricating oil viscosity parameters are too small, which does not conducive to the driving the lubricating oil to the worm meshing position. Secondly, due to the limitation of the simulation algorithm (the diameter of each particle is millimeter-scale), it is impossible to simulate the micro-nano-scale oil film on the tooth surface. In fact, an oil film will adhere to the surface so that the meshing part is lubricated after the tooth surface is immersed in lubricating oil.

Fig. 14 is a pressure cloud map at 5s. It can be seen from the Fig. 14 that the pressure of the oil pressure on the wall surface of the box is relatively large, while the pressure near the worm gear and the worm gear bearing is small. As the pressure at the attachment of the rotating part is small, the resistance to rotation is also small, which can also reduce part of the churning losses.

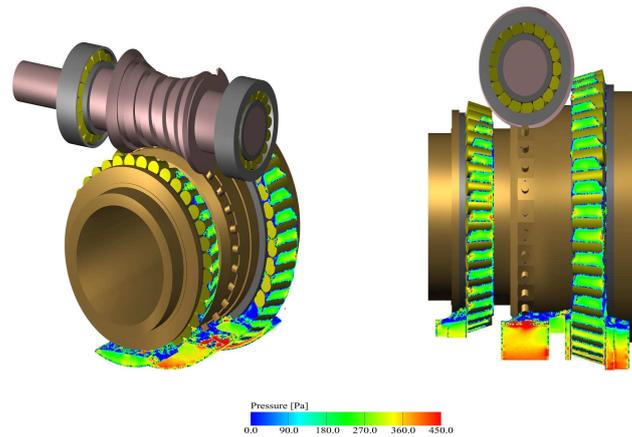


Fig. 14. Pressure cloud map at 5s

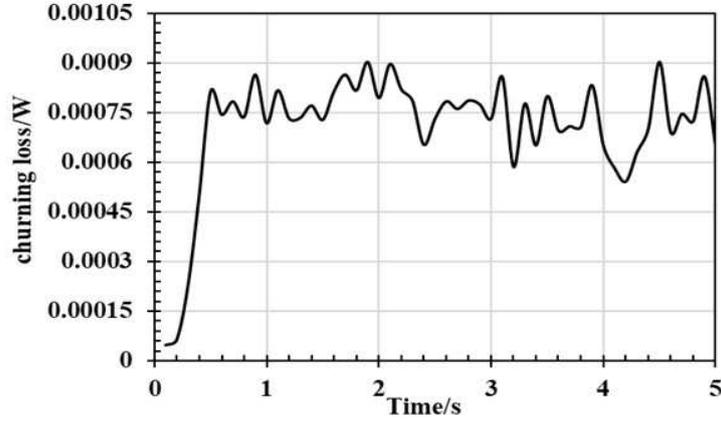


Fig. 15. 0 to 5s of churning power losses

The instantaneous churning power losses obtained by simulation calculation are demonstrated in Fig. 15. It can be seen from the Fig. 15 that at $0 \sim 0.5s$, the churning power losses increase with the increase of the speed. When the rotation speed reaches the maximum, the churning power losses are in a relatively stable state at $0.5 \sim 5s$ (fluctuating in the range of $6 \times 10^{-4} \sim 9 \times 10^{-4}w$). The optimized calculation of churning power losses of $7.4e-4W$ is also within this range.

In order to observe the error of simulation calculation and optimization calculation more specifically, we take the average value of the churning power losses ($0.5 \sim 5s$) after the speed is stabilized and the result is $7.53 \times 10^{-4}w$, and the relative error from the optimized value ($7.4 \times 10^{-4}w$) is 1.79%. Through the above simulation analysis, the accuracy and feasibility of OSNPS in the optimization of churning losses parameters are proved. The optimization analysis can be carried out for the multiple lubrication performance influence parameters of the reducer.

5 Conclusions

It is not easy to ensure the accurate and efficient optimization of multiple parameters in the lubrication research of the reducer. To do this, this paper proposes a method for optimizing the lubrication parameters of the reducer based on the ROSNPS. Moreover, take the ZHPREER as an example to carry out the corresponding research and get the following conclusions:

- (1) We were taken C_{ch} as the optimization target and $\rho, v, d_2, D, \Omega, h, d_c, v, V_0, b$ as the parameter to be optimized. The corresponding objective function

was established, and 20 optimization analyses were carried out. Due to the randomness of the optimization algorithm, it cannot be guaranteed that the numerical optimal solution will be obtained every time while we will get a result with significantly reduced churning losses;

- (2) In the optimization results, the group (the 20th group, 0.00074W) closest to the average value of the 20 groups (0.000747W) was selected as the final optimization result. MPS was used for fluid simulation modeling verification, and the optimization results (0.00074W) were compared. Moreover, in the simulation calculation result (0.000753W), the error between the two is only 1.79. The error is mainly caused by the simplified processing of the model in the analysis, proving the correctness of the optimization algorithm established in the article;
- (3) Due to the limitations of the established optimization model and simulation algorithm, it is currently only possible to optimize analysis and verify the churning losses. It is challenging to optimize the analysis for the microscopic oil film thickness of the meshing tooth surface. In order to realize the efficient analysis and optimization of the oil film thickness of the meshing tooth surface with different lubrication parameters, the following work will continue to explore the application of the ROSNPS in the optimization of the lubrication parameters of the reducer.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61972324, 61672437, 61702428), the Sichuan Science and Technology Program (2021YFS0313, 2021YFG0133, 2020YJ0433, 2021YFN0104), Beijing Advanced Innovation Center for Intelligent Robots and Systems (2019IRS14) and Artificial Intelligence Key Laboratory of Sichuan Province (2019RYJ06).

References

1. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182-197(2002).
2. Gao, F., Huang, L., Liu, S., Dai, C.: Artificial Bee Colony Algorithm Based on Information Learning. *IEEE Transactions on Cybernetics*, 45(12), 2827-2839(2015).
3. Han, K., Kim, J.: Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE Transactions on Evolutionary Computation*, 6(6), 580-593(2002).
4. Han, K., Kim, J.: Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions. *Information Sciences*, 181(16), 3508-3531(2011).
5. Maulik, U., Bandyopadhyay, S.: Genetic Algorithm-Based Clustering Technique. *Pattern Recognition*, 33, 1455-1465(2000).
6. Hu, Y., Li, G., Zhu, W., Cui, J.: An Elastic Transmission Error Compensation Method for Rotary Vector Speed Reducers Based on Error Sensitivity Analysis. *Applied Sciences-Basel*, 10(2), 22(2020).

7. Sun, Y., Hu, B., Li, F., Jiang, Z., Xiong, H., Tao, B., Zheng, Z., Jiang, D.: Gear reducer optimal design based on computer multimedia simulation. *Journal of Supercomputing*, 76(6), 4132-4148(2020).
8. Yang, W., Tang, X., Liang, Q.: Tooth Surface Contact Analysis of Involute Rotate Vector Reducer Based on a Finite Element Linear Programming Method. *IEEE Access*, 7, 19-31(2019).
9. Yang, W., Tang, X., Liang, Q.: NUMERICAL SIMULATION OF BUBBLE MOTION IN HORIZONTAL REDUCER PIPELINES. *Engineering Applications of Computational Fluid Mechanics*, 5(4), 517-529(2011).
10. Ghazijahani, T., Showkati, H.: Experiments on conical shell reducers under uniform external pressure. *Journal of Constructional Steel Research*, 67(10), 1506-1515(2011).
11. He, H., Li, H., Lyu, S., Tak, S., Moon, S., Zhang, Q.: Optimal design of a tilling machine reduction gearbox using Matlab. *International Journal of Precision Engineering and Manufacturing*, 10(2), 63-66(2009).
12. Savsani, V., Rao, R., Vakharia, D.: Optimal weight design of a gear train using particle swarm optimization and simulated annealing algorithms. *Mechanism and Machine Theory*, 45(3), 531-541(2010).
13. Tudose, L., Buiga, O., Stefanache, C., Sobester, A.: Automated optimal design of a two-stage helical gear reducer. *Structural and Multidisciplinary Optimization*, 42(3), 429-435(2010).
14. Korta, J., Mundo, D., Stefanache, C., Sobester, A.: Multi-objective micro-geometry optimization of gear tooth supported by response surface methodology. *Mechanism and Machine Theory*, 109, 278-295(2017).
15. Daoudi, K., Boudi, E., Abdellah, M.: Genetic approach for multiobjective optimization of epicyclic gear train. *Mathematical Problems in Engineering*, (2019).
16. Korta, J., Mundo, D., Stefanache, C., Sobester, A.: Numerical simulation and optimization of oil jet lubrication for rotorcraft meshing gears. *International Journal of Simulation Modelling*, 17(2), 318-326(2018).
17. Dai, Y., Jia, J., Ouyang, B., Bian, J.: Determination of an Optimal Oil Jet Nozzle Layout for Helical Gear Lubrication: Mathematical Modeling, Numerical Simulation, and Experimental Validation. *Complexity*, 2020, (2020).
18. Deng, X., Wang, S., Hammi, Y., Qian, L., Liu, Y.: A combined experimental and computational study of lubrication mechanism of high precision reducer adopting a worm gear drive with complicated space surface contact. *Tribology International*, 146(5-6), 106261(2020).
19. Deng, X., Wang, S., Youssef, H., Qian, L., Liu, Y.: Study on the influence of key design parameters on lubrication characteristics of a novel gear system applying Taguchi method. *Structural and Multidisciplinary Optimization*, 62(5), 2833-2847(2020).
20. Chen, L., Ma, P., Tian, J., Liang, X.: Prediction and optimization of lubrication performance for a transfer case based on computational fluid dynamics. *Engineering Applications of Computational Fluid Mechanics*, 13(1), 1013-1023(2019).
21. Wang, Y., Song, G., Niu, W., Chen, Y.: Optimized design of spray parameters of oil jet lubricated spur gears. *Tribology International*, 120, 149-158(2018).
22. Păun, Gh.: Computing with membranes. *Journal of Computer and System Sciences*, 61(1), 108-143(2000).
23. Pan, L., Păun, Gh., Zhang, G.: Foreword: Starting JMC. *Journal of Membrane Computing*, 1(1), 1-2(2019).
24. Zhang, G.: Membrane computing. *International Journal of Parallel, Emergent and Distributed Systems*, 36(1), 1-2(2021).

25. Orellana-Martín, D., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.: Minimal cooperation as a way to achieve the efficiency in cell-like membrane systems. *Journal of Membrane Computing*, 1(1), 1-2(2019).
26. Pan, L., Orellana-Martín, D., Song, B., Pérez-Jiménez, M.: Cell-like P systems with polarizations and minimal rules. *Theoretical Computer Science*, 816, 1-18(2020).
27. Kari, L., Rozenberg, G.: Tissue P systems with channel states. *Theoretical Computer Science*, 330(1), 101-116(2005).
28. Kari, L., Rozenberg, G.: Tissue-like P systems with evolutionary symport/antiport rules. *Information Science*, 378, 177-193(2017).
29. Song, B., Pan, L., Pérez-Jiménez, M.: Tissue P Systems with Protein on Cells. *Fundamenta Informaticae*, 144(1), 77-107(2016).
30. Ren, T., Cabarle, F., Adorna, H.: Generating context-free languages using spiking neural P systems with structural plasticity. *Journal of Membrane Computing*, 1(8), 161-177(2019).
31. Jiang, Y., Su, Y., Luo, F.: An improved universal spiking neural P system with generalized use of rules. *Journal of Membrane Computing*, 1(8), 270-278(2019).
32. Dong, J., Rong, H., Neri, F., Yang, Q., Zhu, M., Zhang, G.: An Adaptive Memetic P System to Solve the 0/1 Knapsack Problem. *Communications of the IEEE*, 1-8(2020).
33. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking Neural P Systems. *Fundamenta Informaticae*, 71(2), 279-308(2006).
34. Zhang, G., Rong, H., Neri, F., Pérez-Jiménez, M.: An optimization spiking neural P system for approximately solving combinatorial optimization problems, *International Journal of Neural Systems*, 24(5), 1440006(2014).
35. Zhu, M., Yang, Q., Dong, J., Zhang, G., Neri, F.: An Adaptive Optimization Spiking Neural P System for Binary Problems, *International Journal of Neural Systems*, 31(1), 2050054(2020).
36. Changenet, C., Vexel, P.: A Model for the Prediction of Churning Losses in Geared Transmissions-Preliminary Results. *Journal of Mechanical Design*, (2007).
37. Misić, T., Najdanović-Lukić, M., Nesic, L.: Dimensional analysis in physics and the Buckingham theorem. *European Journal of Physics*, 31(4), 893(2010).
38. Deng, X., Wang, S., Qian, L., Liu, Y.: Simulation and experimental study of influences of shape of roller on the lubrication performance of precision speed reducer. *Engineering Applications of Computational Fluid Mechanics*, 14(1), 1156-1172(2020).
39. Liu, J., Koshizuka, H., Oka, Y.: A hybrid particle-mesh method for viscous, incompressible, multiphase flows. *Journal of Computational Physics*, 202(1), 65-93(2005).
40. Lax, P., Wendroff, B.: The Courant-Friedrichs-Lewy (CFL) Condition. *Communications on Pure and Applied Mathematics*, 15(4), 363-371(1962).

A Review of Computing Models for Giant Panda Ecosystems

Yingying Duan¹, Haina Rong¹, Gexiang Zhang^{1,2*}, Dunwu Qi³, Luis Valencia-Cabrera⁴, Mario J. Pérez-Jiménez⁴

¹School of Electrical Engineering, Southwest Jiaotong University, Chengdu 611756, China

²School of Control Engineering, Chengdu University of Information Technology, Chengdu 610225, China

³Chengdu Research Base of Giant Panda Breeding, Chengdu 610081, China

⁴Research Group on Natural Computing. Department of Computer Science and Artificial Intelligence, University of Sevilla. Sevilla 41012, Spain;

Abstract. As both a flagship and umbrella species, the giant panda (*Ailuropoda melanoleuca*) has been an endangered species for decades, and important decisions affecting its conservation have succeeded in improving its situation (currently considered vulnerable by the IUCN Red List). To aid in such decision-making processes, mathematical models, and particularly computational models, are helpful in the study of the population dynamics of the species. Based on solid theoretical foundations, and providing conceptual models bringing aspects from different scientific disciplines, some of these modelling frameworks present the capability to deal with complex systems in different areas, involving a number of processes and interacting organisms, among other elements. Specifically, given the relevance of the species, it is highly interesting to model giant panda ecosystems making use of computing models. In this work, we provide an overview of some of the main types of models applied to giant panda ecosystems (*e.g.* differential dynamic models, age-classified matrix models, Vortex models and membrane computing models). The existing modelling approaches typically attempt to address one or more of the following aspects: describing and unravelling giant panda ecosystem components and interactions; making predictions about the future states; and identifying important uncertainties. The main attributes of four types of computing models studied are compared to evaluate these approaches.

Keywords: Giant panda ecosystems, differential dynamic models, matrix models, VORTEX models, membrane computing models

1 Introduction

The giant panda (*Ailuropoda melanoleuca*) is a world-renowned iconic species. According to the fossil records, the giant panda has a long evolutionary history dating back to 7-8 Mya, from the late Miocene to the Pleistocene era [94, 95]. Historically, this species was once widely distributed in Myanmar (Burma), northern Vietnam, and much of eastern and southern China as far north as Beijing [37]. Giant pandas were first discovered

* Corresponding author.

by the French biologist David in 1869 [15]. However, the giant panda had so far become the only extant species of the panda lineage [86, 95], being one of the world's most well-known endangered species, since the number of individuals in the species plummeted about 250 years ago [63]. Although giant panda numbers are currently surging, according to the estimation provided by the fourth large-scale survey carried out in China [92], this species still faces serious threats, *e.g.* low reproductive rate, inbreeding, and human activities affecting its habitat, among others [46, 83]. It is still crucial to keep paying a special attention to the population dynamics of the giant panda and the influence of different aspects on the evolution of the species. Nature must be protected respecting its own evolution, but at the same time research is required and can affect the observed phenomena. Those two goals may conflict: on the one hand, scientific achievements frequently require a strong experimentation to analyse the influence of certain factors over a system; on the other hand, direct experiments on an ecosystem to study its evolution could affect negatively its natural evolution. Therefore, as the researchers cannot directly apply any experiment carelessly and observe the effects on the population dynamics of the species over time under certain threats. Consequently, it is necessary to find other approaches to gain knowledge about the processes and factors involved and the possible evolution of the species under different scenarios, with the least possible influence on the system under study. In this context, ecosystem models can provide helpful tools to increase such understanding of the phenomena. A model provides a representation of certain system or concept, aiming to increase the comprehension of the underlying phenomena. In our particular case, we will be focused on the types of models that can be specially useful to manage the ecosystems under study. Such is the case of mathematical models, including formal definitions of the elements involved in the representation, and more specifically computational models, conceived to be particularly suitable for their handling through systematic processes in a machine (generally with computers, but possibly also in other types of electrical or biological machines). Such models overcome the limitations imposed by direct experimentation in Nature, as its harmless alternative with so-called *virtual experiments* instead.

The foundations of computational models were laid in the second half of the 20th century [80, 88]. Modeling analysis of giant panda ecosystem has since been the topic of extensive research, and earned a significant leap forward in the late 1980s/early 1990s with the formulation of concepts and techniques such as path method, DNA technique, remote sensing technique, or spline sampling method, among others [59, 92, 31]. However, until rather recently, in most cases, these modeling methodologies were hardly constrained by the limited biological data available about giant pandas. With recent modeling technological advances, in a number of domains, the availability of various models that estimate population sizes of giant pandas in different areas has increased, leading to a growing interest and better options to exploit data and methods efficiently [44, 45]. One of the approaches that can be followed in the modelling of ecosystems are the computational models, suitable to handle complex dynamic systems. They are founded on theoretical frameworks with interesting properties for the handling of the systems under study, and in the particular case of population dynamics they try to represent and understand the structure and dynamics of the ecosystems in order to predict how they will evolve over time [57]. In general, such models attempt to incorporate

ecosystem components (e.g. populations and species) and processes (e.g. predator-prey interactions, large and small perturbations or dispersion) into a model using a representation based on a certain modelling framework [8, 19]. The aim is to enable capturing, unraveling the essence of the system behavior, and then predicting the outcomes of the complex interactions between the ecosystem components in a meaningful way [14, 78]. Many of the multi-view, multi-relationship and diversification models are associated with the complex behaviors of giant pandas, their physical environment and their interactions in a particular spatial unit. Thus, the interest in developing new and efficient analytical methodologies is high and goes far beyond pure modeling interest.

Motivations for modeling giant panda ecosystems using different computational models are numerous. They include, among others: a) obtaining a more comprehensive and complete understanding about the overall system with all the processes involved in the population dynamics of the giant pandas; b) improving the decision making conducted by the managers of the systems under study; c) leading exploratory research go gain knowledge about the species; d) handling some specific problems about giant panda ecosystems, such as identifying common versus distinctive ingredients; e) in general, extracting biological behaviours observed in giant panda ecosystems and study possible disturbances due to external factors in various study areas. However, despite the clear potential benefit, and vast amount of work that has been done in the field (see, for example, DDMs [85], PDP systems [90] and the references listed therein), many aspects affecting the related ecosystems and their evolution remain uncertain. Much work is still open in order to increase the existing knowledge about how to handle the natural environments constituting the habitats of giant pandas, with the support of ecosystem models that are still at a relatively preliminary stage in what concerns the wild environment.

Modeling giant panda ecosystem is a challenging task for several reasons [65, 90]. First, the data about giant pandas are collected by tedious manual and statistical methods. Second, in any real physical environment, there are a large number of underlying variables (*e.g.*, nature disasters) for which we have no prediction. Due to the wide variety of variables, the number, type, and scope of new research models that can potentially be posed is overwhelmingly large. Third, working with suitable computational models whose advantages can be properly exploited, while overcoming certain drawbacks, is not an easy task. Each model has its unique characteristics. For instance, in differential dynamic models, completely new functions are specially designed to estimate the continuous change of population size of the giant panda over time. In general, the results predicted by these models are the same in each simulation, that is, they are not affected by the number of simulations. In addition, in each computation, the calculated population size of the first case can be pre-estimated once the form of the function is given (which means that the population size at every moment have been determined). For other models, the pre-designed modeling software are used to simulate the discontinuity change of population size of this species over time. At any time the calculated results of software simulation cannot be predicted ahead because the researchers cannot acquire time of birth or mortality of an individual, which means that the computational results at each steps need be obtained by the chose simulation software and a series of results can only be determined after reaching the halting condition. The problems for

the two types is that it is critical to decide the values of the used parameters so as to avoid overfitting of functions or the bigger deviation between the statistical data and the estimated results. However, this is very difficult to achieve and in most of the cases this property is not satisfied.

All the evidences push in favor of further investigating models for giant panda ecosystems. This way of observing, modeling and predicting has evolved for hundreds of years. Studying this meaningful modeling process might help us to further understand the changing state of population dynamics of giant pandas in the future. In what follows, the main contributions of this paper are summarized:

(1) At present, the number of giant pandas is less than 1500, being one of the most heavily endangered species. Despite the considerable efforts of the Chinese people and government to conserve pandas in the wild, the population continuous to decline and at the present rate the giant panda will become extinct perhaps as early as the twenty-first century [24, 53]. Hence, studying giant panda ecosystem possesses higher scientific research value and significant in conservation biology.

(2) This paper presents the review of computational models for *endangered* giant panda ecosystems. These models provide a tool to analyse the trend of the population dynamics of the species in certain areas, targeting a complete solution taking care of efficiency, flexibility, scalability and robustness. Such models can involve a number of parameters related with the biology of the species and include processes considering the impact of external environmental phenomena on the viability of such vulnerable animals. This successful applications of computational models provide hope for protection of the giant panda from the negative trends in biodiversity and ecosystem integrity that are common worldwide.

(4) On the basis of historical references, this paper reviews the research status, summarize the existing research findings, and analyze their attributes. More importantly, we elaborate on various extensive models used for dealing with these matters that traditional models cannot solve in some cases, and analyze the advantages and disadvantages of each type of models. For differential models, we elaborate on the approaches identifying the existence of periodic solutions, boundness, stability, and permanence of differential models in order to make these models feasible when modeling giant panda systems. For Vortex models, we discuss several critical issues on how to generate *random numbers*, how to select *individuals breeding*, and how to evaluate *environmental variation*. For membrane computing models, we can not only give the reasons that the models are used for modeling, but also list a compared table of different models aiming to analyze the specific applications of every model.

The paper aims to survey and emphasize the necessity and importance of advanced computational models based on biological data to the protect of giant panda ecosystem. More specifically, we would like to promote modeling approaches, that is, approaches with minimal and weak constraints, such as environmental ingredients, climate elements, and human factors, among others, that can be useful to predict the trends of population dynamics of this species. The goal is to provide some ideas, perspectives, and guidelines as to how to model giant panda ecosystems.

In order to make this paper accessible for readers with various interests and backgrounds, the rest of this paper is organized as follows: Section 2 gives a brief description

of geographical distributions, influential ingredients, and the population status of the giant panda studied. Section 3 summaries various computational models, where section 3.1 elaborates on different types of differential dynamic models, section 3.2 covers the main concepts, techniques and applications for the age-classified matrix models, section 3.3 reviews the Vortex models (software) applied to various types of environments such as climate, migration, and supplements, section 3.4 gives more details on existing membrane computing models used for studying population dynamics of this species, and the computational models are compared and analyzed in section 3.5. Section 4, concluding remarks and future research lines are discussed.

2 Giant Panda Ecosystem

In this work, we restrict our attention to the geographical distributions, the influential ingredients in geographical situations and the population distributions of giant pandas, where the first item focus on the habitats of giant pandas, the second one focus on the impacts of natural disasters on giant pandas and the third focus on population size of giant pandas.

2.1 Geographical Distribution

According to four national survey reports of giant pandas conducted by the Chinese government, giant pandas mainly inhabited 17 cities (prefectures), 49 counties (cities, districts) and 196 townships in Sichuan, Shaanxi and Gansu provinces. However, human activities like farming, deforestation, or other development, have driven the giant panda out of the lowland areas where it once lived [54, 87]. Now the species remains only in six mountains. Since the establishment of the first giant panda reserve in the early 1960s, Chinese government, with the support of the international conservation community, has invested enormously in giant panda conservation [71]. So far, 69 nature reserves in six mountains from these regions have been built for them in China. Among these mountains, the largest number is Minshan Mountains (MS) with 31 giant panda nature reserves. Both Qionglai Mountains (QLS) and Liangshan Mountains (LS) are the second and third with 17 reserves and 3 reserves, followed by Daxiangling Mountains (DXL) with 3, and Xiaoxiangling Mountains (XXL) with 11, Qinling Mountains (QL) with 1, respectively. There are an amount of reserves used for studying the population dynamics of giant pandas, *e.g.*, Wolong nature reserve in QLS, Yele nature reserve in XXL, Tangjiahe in MS, Baoxing nature reserve in QLS, and Wanglang nature reserve in MS, *etc.*, (the fourth column of Table 1). These reserves are home to most giant pandas.

All the studies published in the references encompassed these six mountain ranges that constitute the extant geographic distribution of giant pandas. Habitat types transition vertically through altitude changes within the giant panda distribution, from subtropical evergreen broad-leafed forests, to mixed coniferous and deciduous broad-leafed forests, and up to subalpine coniferous forests [16, 67]. In order to clarify how the giant pandas are placed, a geographical distribution map is provided in Figure 1. In addition, a detailed list is given in Table 1, including the habitat of the giant panda studied in each model analyzed.



Fig. 1. Historic distribution of giant pandas (inset map) and present-day distribution (red). (Adapted from [49])

2.2 Influential Ingredients

At present, temperature and rainfall vary widely within the distribution of giant pandas. Besides, a very significant variation is present in their soils, hydrology, slope, and other factors. All these aspects impose environmental differences influencing the evolution of the species [82, 97]. However, for these regions published in the references currently available, the main challenges faced by the species mainly focus on bamboo flowering, hunting, forest fire, diseases, species invasion and habitat fragmentation, where the period of bamboo withers caused by blooming is determined by local climate conditions. Each of these types of threats involves different influencing elements, including causes derived from, or strongly influenced by, human activities, along with other aspects difficult to control, with a high degree of uncertainty. The common factors are primary drivers for the death of giant pandas. Hence, when modeling giant panda ecosystems, it is necessary for researchers to consider the actual situations for determining which factor should be analyzed and added to the designed model.

Here, for simplicity, we also termed these challenges faced and potential threats, including possible catastrophes, as *parameters*. When several types of parameters needed to be considered in a model (that is, a region is affected by many kinds of disasters), it is critical to analyze the existing forms of parameters in the model (for further detailed information on how to model parameters, please carefully read the contents of the following sections).

For sections 2.1 and 2.2, in order to clarify the specific areas of study in the giant panda ecosystems published in references, the geographical distributions and their influential ingredients are summarized in Table 1.

2.3 Population Distribution

Baseline data on giant pandas is from four national surveys for giant pandas [92]. In Table 2, we list the number, population density and the deviation calculated by dividing the difference between third and fourth surveys by the number of third survey of giant

Table 1. Some mathematical models of modeling giant panda ecosystem.

Year	Type of Model	Factor Type	Region	Reference
1989	ODE, nonlinear dynamic model	Bamboo	Yele Nature Reserve	(Yuan <i>et al.</i> [88], 1989)
1995		Bamboo	Wolong Nature Reserve	(Wang <i>et al.</i> [80], 1995)
1996	Leslie matrix model	Natural environment, hunting	Foping Nature Reserve	(Guo and Yuan[?], 1996)
1997		Actual environment; environmental carrying capacity; poaching; immigration	Qinling Mountains	(Zhou and Pan[96], 1997)
1997	VORTEX model	Environmental variation, carrying capacity, catastrophes (hunting, bamboo flowering, forest fire, diseases, species invasion)	Foping Nature Reserve	(Li <i>et al.</i> [47], 1997)
1997		No catastrophes; catastrophes (bamboo die off→the environmental decrease of carrying capacity); heterozygosity	Wolong Nature Reserve	(Wei <i>et al.</i> [84], 1997)
1998	ODE, nonlinear dynamic model	Bamboo (phyllostachys bissetii and fargesia robusta Yi), density dependent	Wolong Nature Reserve	(Wu and Yuan[85], 1998)
1999		Bamboo biomass dynamic	Xiangling Mountain	(Carter <i>et al.</i> [3], 1999)
1999		Inbreeding, catastrophes (Single bamboo and bamboo blooming)	Yele Nature Reserve	(Guo and Hu[27], 1999)
2002	Leslie matrix model	Heterozygosity, inbreeding, immigrating, environmental carrying capacity, catastrophe (blooming, flood), outer supplement	Tangjiahe Nature Reserve	(Zhang and Hu[93], 2002)
2002		Inbreeding; single bamboo, bamboo blooming; Hunting	Mabian Defengding NR	(Ren <i>et al.</i> [60], 2002)
2002		Lack of food, bamboo blossoming; the permutation and combination between inbreeding and catastrophes	Yele Nature Reserve	(Guo and Hu[25], 2002)
2002	ODE, nonlinear dynamic model	The periodic flowering and die off of bamboo species	Xiangling Mountains	(Guo <i>et al.</i> [26], 2002)
2002	Population stochastic model	Density dependent factors	Qinling Mountains	(Wang <i>et al.</i> [79], 2002)
2007	Difference equation	Actual environment	Changqing Nature Reserve	(Yuan[89, 54], 2001)
2008	VORTEX model	Inbreeding, bamboo blooming, forest fires	Xiaoxiangling Mountains	(Zhu <i>et al.</i> [98], 2008)
2010		Inbreeding, environmental carrying capacity, catastrophes	Baoxing Nature Reserve	(Jiang and Hu[34], 2010)
2012	ODE, nonlinear dynamic model	Habitat fragmentation	Xiaoxiangling Moutains	(Gui <i>et al.</i> [21], 2012)
2012	ODE, nonlinear dynamic model	Deforestation, bamboo (forest-bamboo-giant panda)	Wanglang Nature Reserve	(Gui <i>et al.</i> [22], 2012)
2012	ODE, nonlinear dynamic model	Deforestation, bamboo with Periodic solution and chaos strang attractors	Wanglang Nature Reserve	(Gui <i>et al.</i> [23], 2012)
2013	Impulsive DE, nonlinear model	Bamboo flowering	Qionglai Mountains	(Shi and Song[65], 2013)
2016	ODE with Allee effect, nonlinear	Diffussion loss	Qinling Mountains	(Zhang and Song[91], 2016)
2017	Membrane computing model	Single-environment, rescue	GPBB	(Huang and Zhang[32], 2017)
2018	VORTEX model	Bamboo, carrying capacity	Xiaoxiangling Mountains	(Yang <i>et al.</i> [86], 2018)
2018	Membrane computing model	Single-environment, rescue, release	GPBB	(Tian and Zhang[73], 2018)
2019		Multi-environment, rescue	GPBB, CCRCGP	(Tong and Zhang[75], 2019)

pandas, that is, there are only 2nd, 3rd and 4th, since the first survey is not recorded systematically. Up to now, for the 2nd survey report recorded from 1985 to 1988, there are 909 giant pandas, where the largest number is in MS with 485 giant pandas. QLS and LS are at the second and the third, with 233 pandas and 155 pandas, respectively. For 3rd report recorded from 1999 to 2003, there are 1206 giant pandas, where MS is still at the top position, followed by QLS with 233 and LS with 155. For 4th recorded from 2011 to 2014, there are 1387 giant pandas, where MS contains 666 giant pandas, ranking the first. QLS with 528 giant pandas is at the second, followed by LS with 124 giant pandas. In sum, there are 909 giant pandas published by 2nd report, 1206 published by 3rd report and 1387 published by 4th report, respectively. According to the last two investigations, the number of giant pandas in six mountain ranges are increasing except for XXL, where the growth rate of DXL is 137.50% on average, ranking first among six mountains. And it is followed by QLS with 16.81%, MS with 12.69%, LS with 7.83%. These increases led to the downgrading of the giant panda from “endangered” to “vulnerable” on the International Union for Conservation of Nature (IUCN) Red List on 2016.

Table 2. The distribution, number, population density and population dynamics of giant pandas in various mountains of Sichuan province. (Note that these data are provided by the 4th survey report of Sichuan province [99]).

Reserve	County	Reserves	2nd	3rd	4th	2st	3rd	4th	Devi(%)
Minshan	11	32	485	591	666	0.1057	0.0770	0.0844	12.69
Qionglai	14	17	233	452	528	0.0729	0.0738	0.0767	16.81
Daxiangling	5	3	20	15	38	0.0702	0.0197	0.0309	137.50
Xiaoxiangling	3	3	16	32	30	0.0719	0.0399	0.0251	-6.25
Liangshan	8	11	155	155	124	0.0789	0.0522	0.0410	7.83
Qinling	1	1	0	0	1	0.0887	0.0680	0.0684	15.01
Total number		69	909	1206	1387				

3 Computational Models for Giant Panda Ecosystem

In several references [21, 25, 90], it is concluded that the computational models are a class of suitable models to predict the population dynamics of giant panda ecosystems at one region or multi-regions, overcoming some drawbacks related to traditional statistical approaches.

At present, there are 26 papers that have been published, as shown in Table 1. The models in these papers focus on four types of computational models: (a) Differential dynamic models (DDMs for short); (b) Age-classified matrix models (ACMM); (c) Vortex

models (Vortex); and (d) Membrane computing models (mostly PDP systems). Computational models used for modeling giant panda ecosystem have a common feature, in which for each simulation execution, first the biological data are input into the system (as initial input data), and secondly these models are executed, generating a series of population sizes of giant pandas over time. It is necessary to add several ingredients from external environments into the corresponding models in the process of calculations in order to allow more realistic predictions on the population dynamics of this species (we will elaborate on them in the following sections).

Among all the models, *differential dynamic model* is firstly proposed to model giant panda ecosystem. As the studied areas, also termed natural environment, become more and more complex, this will make the functions non differentiable because of not meeting the following conditions: (a) stability, (b) existence of equilibrium point, (c) contingency events, *etc.* Hence, other types of models have emerged one after another. For these models, there is no need to handle these problems due to estimate according to the basic behaviors of each giant pandas rather than the abstracted behavior functions. Also, when modeling giant panda ecosystem using computing models, the main difficulty comes from the fact that there exists a large number of uncertain factors (parameters) from (real) environments directly or indirectly causing the fluctuation of population size of the giant panda. This poses a great challenge for the conception of a general computational framework to accurately model this species.

In the following subsections, the existing computational models are summarized according to the time sequence of the models of Table 1.

3.1 Differential Dynamic Models

With the advent of differential dynamic models that are used for modeling ecosystems by building the relationship between the population of some species and different types of ecological environments, designing a novel and suitable differential dynamic model predicting population dynamic of giant pandas is no longer an exceedingly difficult work. The first attempt to use the differential dynamic approach to model giant panda ecosystem dates back to 1989 [88]. Since then, different types of the models emerged, considering impulsive effect, Holling effect and Allee effect to tackle *contingent events* encountered by traditional models.

3.1.1 Basic differential dynamic models

Essentially, a basic differential dynamic model is used to look for the simplest mathematical model describing a relationship between the population dynamic of the giant panda and other species. Generally, for each model like this, its density dependence is nonlinear. Suppose we want to study a two-population predator-prey model to understand the impact of bamboo destruction on the population dynamic of the giant panda. We denote the density of the giant panda as x_1 , and the density of the bamboo as x_2 . Then, a_{ij} is used to denote a coefficient measuring the degree of influence between two species. The problem of modeling giant panda ecosystem in this framework can be formulated as follows:

$$\begin{cases} \dot{x}_1 = x_1 \overbrace{(a_{10} + a_{11}x_1 + a_{12}x_2)}^{\text{density-dependence}}, \\ \dot{x}_2 = x_2(a_{20} + a_{22}x_2 + a_{21}x_1) \end{cases} \quad (1)$$

where

(a) a_{i0} , the instinct growth rate of giant pandas, is the reproductive rate of this species minus its mortality rate.

(b) a_{ii} denotes the density dependent coefficients of a population: if $a_{ii} < 0$, then the population is density dependent; if $a_{ii}=0$, then it is density independent.

(c) a_{ij} ($i \neq j$) denotes a relationship between species i and j :

- if $a_{ij} < 0$ and $a_{ji} < 0$, there is a competitive relationship between species;
- if $a_{ij} > 0$ and $a_{ji} < 0$, there is a predator-prey relationship between different species where x_i is the predator and x_j is a prey;
- if $a_{ij} > 0$ and $a_{ji} > 0$, there is a mutualistic or commensal relationship between different species.

For more than two types of species, the resulting models can be viewed as a variant of the model above. Unlike a two-population model, this one is initially designed to model the effect of more than one type of bamboos on giant pandas. Obviously, the models of these populations are more complex than those involving two species, but the modeling techniques are similar to those applied for two-dimensional models. In the following sections, we review and analyze the key modeling types on three populations. For the framework of Formula (1), please observe Fig. 2 (B) and (E). In this figure, an arrow around a circle represents that there exists the density restriction in this population.

In Formula (1) or other model like this, the model is highly sensitive to coefficient a_{ij} because the model easily yielded oscillations causing the instability of the system when we slightly alter the parameter a_{ij} . For each model especially a linear model considering these models that have not been studied before, the first is to search all the equilibrium points and the second discusses whether the model is stable at each equilibrium position. If there are equilibrium points to stabilize this system, the isoclinical equations of system (2) x_i ($i = 1, 2, \dots$) can be calculated.

(a) Two-population differential dynamic model: The two-population differential dynamic model is chosen to model the change of the growth rate of the giant panda with time as well as one kind of bamboos. In general, Formula (1) is regarded as a idealized model without considering real natural environment. To solve the problem, Reference [80] used the designed function instead of the common constant. More specifically, the growth rate of the giant panda can be modified with the amount of bamboo. In addition, it is necessary to solve all the equilibrium points of this model and then obtain these points making the designed system in a stable state. Note that the stable state of the system can be broken if some parameters are introduced to the model or removed from the model. Different from previous models, the migration behavior is added to the previous designed model. It can be seen from the simulated results that there is only an equilibrium position satisfying the Lipschitz condition compared with two positions solved by the previous model under the stable state.

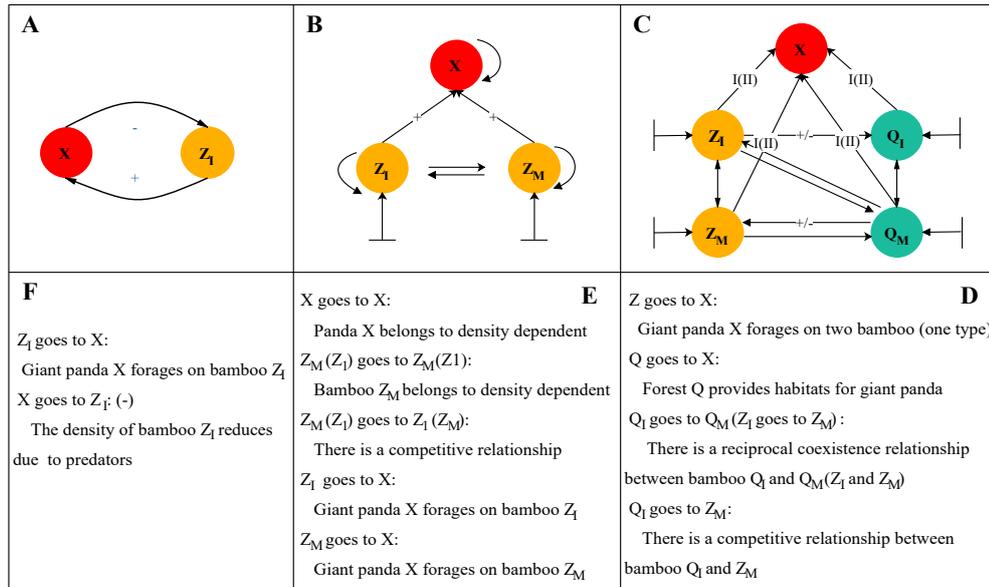


Fig. 2. Modeling process of three types of differential dynamic models. (A-C) Schematic representation of the predator-prey relationships composed of giant panda (red circle) and the bamboo (yellow circle) (A), panda and two bamboo (B), or giant panda and the bamboo in two stages (C). (D-F) mainly describe the concepts of schematics (A-C). In (A) and (C), II (III) represents two different Holling functional response functions, and +(-) represents the impact of a population on another one.

The problem with the predator-prey model above is that the bamboo flowering and then dying off occurred over many time steps have not been included in this model, making the growth equation of giant pandas change slightly. To address this problem, the *bamboo flowering-giant panda* model [26] is designed as an extended type of the model above to evaluate the population dynamic of giant pandas over a relatively suitable duration. Similar to the model of Reference [80], when attempting to obtain all the equilibrium points, the computation of this model faces the problems of whether a stable relationship exists between panda population and bamboo density and whether or not the changes of the parameters will affect the equilibrium of the system. To get the answer, we need to retrieve data information of the studied region. In Reference [26], use the data information of Yele Nature Reserve to obtain all the coefficients used in the model equations, the formulation of the *bamboo flowering-giant panda* model is determined (please refer to system (3) in [26]). For the first question, the first step is to obtain all the non-negative equilibrium points. After this, the characteristic values of the corresponding variational matrix of each point are obtained and then identify the stability of some equilibrium one, obtaining the stable focuses; the second is to constitute a closed region having a center stable focus. In this region, the system will eventually get a stable status over a large area with its centre of attraction even though all the points may leave this region at some moment as time increases. In general, the non-negative equilibrium stable points stands for the number of giant pandas and bamboos having a long period of co-exist time. The saddle point making the system under an instability status will go far beyond the equilibrium position as time increases, which indicates that that the density of bamboo will decrease rapidly directly makes giant pandas go extinct nearly. These two cases indicates that sometimes changes in the ecological environment do not affect the final balance between species, but sometimes occurrence of a sudden event will alter the environment significantly. For another question-branching issues of the model, [26] used Hopf-bifurcation method to get Hopf-bifurcation value, also termed the boundary point, of the system in the stable status. If less than this value, then the equilibrium point can change from a stable focus to an unstable one; If larger than this value, then the branching value of this system indeed exists. Finally, a stable limit cycle derived from the data of Yele nature reserve is obtained by using Hopf-bifurcation method, which verified the correctness of the designed model in Reference [26].

(b) Multi-population differential dynamic model: The model is mainly used for modeling the change of the growth rate of the giant panda with time as well as more than two types of bamboos. In [85], the ecological model of differential equations on three populations of the giant panda and two kinds of bamboo (*Phyllostachys Bissetii* and *Fargesia Robusta Yi*) was set. Different from References [26], Wu *et al.* can use Cramer's Rule and Negative Definite Matrix at every time step t of system (2) to get all the stable points of the system and the conditions of the stability. Take the data from Wolong Nature Reserve as an example, a predator-prey mathematical model on the giant panda and bamboo is built. In this system, the only positive equilibrium point is produced, which indicates that giant pandas and these two bamboos can coexist and reach a balance state at this stable focus. In general, the difference of coefficients can affect the positions of the stability. But the system will eventually

reach a new steady state even if the current survival conditions are broken. According to this steps, Reference [85] calculated the range of the number of giant pandas at each equilibrium point in different arrays of parameters.

3.1.2 Impulsive Differential Dynamic Models

Impulsive differential dynamic models (IDDM) can be viewed as another variant of differential dynamic models. Different from traditional models, which is initially designed to perform the continuous function without any natural perturbations, IDDM is biologically inspired by a change of state abruptly.

According to the references published so far, two types of impulsive perturbations have been studied in the giant panda ecosystem. One is simple impulse models that add a correction term which takes into account the effect of a sudden collapse of bamboo as food source, and the other is complex impulsive models with more impulsive perturbations that are considered locally ordered combination to several impulses. In these models, a singularly perturbed system such as bamboo flowering is the most common and simplest model among them. A ordered combination of three singularly perturbed systems is much more complex than singularly perturbed systems due to the involvement of the correlation.

The acquisition of impulsive differential dynamic models, however, is only a first step. Additionally, we need to discuss some of the issues that should be addressed in the actual processing of impulsive models. In this survey, we focus on the existence of periodic solutions, the stability of the system and the permanence of the systems. A number of approaches used for analyzing the properties of the impulsive models are briefly mentioned in this section.

(1) General framework of IDDM

Impulsive differential equations, that is, differential equations involving an impulse effect, appear as a natural description of observed evolution phenomena of several real-world problems [48, 70]. It is known that many biological phenomena involving thresholds, bursting rhythm models in biology, do exhibit impulse effects. For example, bamboo flowering and bursting earthquake. The differing varieties of bamboo go through periodic die-offs as part of their renewal cycle. The bamboo, at the end of its life cycle, will flower and drop and die, and then vast areas of the bamboo region disappear. Generally died-back bamboo should take from 10 to 20 years before it can support panda population again [35, 72]. So in [65] Shi *et al* choose impulse effect to explain bamboo flowering phenomena. A basic impulsive differential system of giant panda ecosystem involving two kinds of bamboo is described as:

$$\begin{cases} \dot{x}_1 = x_1(a_{10} - a_{11}x_1 - a_{13}x_3), t \neq (n + l - 1)T \\ \dot{x}_2 = x_2(a_{20} - a_{22}x_2 - a_{23}x_3), t \neq nT \\ \dot{x}_3 = x_3(-a_{30} - a_{33}x_3 + a_{31}x_1 + a_{32}x_2), \\ \Delta x_1(t) = -\alpha x_1(t), t = (n + l - 1)T, \\ \Delta x_2(t) = -\beta x_2(t), t = nT, \\ (x_1(0^+), x_2(0^+), x_3(0^+)) = (x_{10}, x_{20}, x_{30}), \end{cases} \quad (2)$$

Subsystems. There are three types of subsystems expressed by variables in the IDDM model: the first type of bamboos (also termed the first subsystem), the second type of

bamboos (the second subsystem), and giant pandas (the third subsystem). In this article, we use variables x_1, x_2, x_3 to denote bamboo, bamboo and giant panda. Usually, these three parameters are used to describe the number of two bamboos and giant pandas at time t and derivatives $\dot{x}_1, \dot{x}_2, \dot{x}_3$ denote the increasing rate of the number of bamboos or giant pandas with time t . we use $(x_1(0)^+, x_2(0)^+, x_3(0)^+)$ to denote the initial biomass labeled as (x_{10}, x_{20}, x_{30}) . The form of a bamboo-bamboo-giant panda system is modeled according to the method of Formula (1). In Formula (2), Each one of the species in the ecosystem is density-dependent. In addition, there are prey-predator relationships between the giant panda and each type of bamboo.

Two kinds of impulsive perturbations. Formula (2) shows that an improved mathematical model by adding a correction term which takes into account the effects of two sudden collapses of bamboos as a food source. Typically, this improved model would be a non-differentiable piecewise function with two discontinuous points. In this formula, it is seen that there are two impulsive perturbations such as *Bamboo I* at time $(n+l-1)T$ and *Bamboo II* at time nT . For *Bamboo I*, $\lim_{t \rightarrow t_k^+} (t) - \lim_{t \rightarrow t_k^-} (t) = -\alpha x_1(t)$ where $t_k = (n+l-1)T$; For *Bamboo II*, $\lim_{t \rightarrow t_k^+} (t) - \lim_{t \rightarrow t_k^-} (t) = -\beta x_2(t)$ where $t_k = nT$.

In the field of ecology, set $\{(nT, (n+l-1)T) | n \in N^+\}$ is termed as impulsive set and functions $-\alpha x_1(t)$ and $-\beta x_2(t)$ are named as impulsive functions. Note that usually $-\alpha x_1 \neq 0$ at time $t = (n+l-1)T$ and $-\beta x_2(t) \neq 0$ time $t = nT$. Also, the left limit is not equal to the right limit, which indicates that systems $x_1(t)$ and $x_2(t)$ are continuous systems except for these two points of discontinuity. Because $(n+l-1)T - nT = (l-1)T$, there will be an impulsive perturbation at every $(l-1)T$ cycles. Concretely, this system can take various forms of solutions due to introduce the impulsive perturbations (introduced at the following part).

The above-mentioned periodic solution of the type IDDM will be labeled as x_r^* . And then, using the impulsive perturbations in the above as a constraint, the generative process of impulsive differential dynamic model is defined as follows:

- a) For every moment t of system (2),
 - (a) Let system (2) be $\dot{x}(t) = U(t)x(t)$ where $\dot{x}(t)$ denotes a derivate vector of the left-hand side of formula and $U(t)$ denotes a linear coefficient matrix;
 - (b) Introduce a fundamental matrix O and let $\dot{O}(t, t_0) = U(t)O(t, t_0)$;
 - (c) According to features of differential equations, it is seen that $x(t) = O(t, t_0)x(t_0)$ where $O(t_0, t_0) = I$;
- b) Floquet theorem have verified that $O(t, t_0) = P(t, t_0)e^{(t-t_0)\Lambda t_0}$ if $U(t)$ is periodic, where both P and Λ are matrices related to time t .
- c) For $O(t, t_0)$, its performance is up to Λ : if all the eigenvalues of Λ are less than 0, then the fundamental matrix O is stable, periodic and exponential decay; if greater than 0, then this matrix O is unstable and exponential increase.
- d) Draw an array of periodic vector for each moment t : $(x_1^*(t), x_2^*(t), x_3^*(t))$.

Note that the middle matrix Λ serves as a bridge between the $O(t, t_0)$ and $x(t)$. This middle matrix, along with time t , is the key that decides $O(t, t_0)$ to simultaneously compute the stability and the period of a system by the trends of exponential function.

(2) Stability of the Giant Panda-Free Periodic Solutions

When considering the stability of the giant panda-free periodic solutions, all the parameters could be treated as random variables so that fully bamboo-bamboo-giant panda model may be applied. However, such treatment typically incurs high computational cost. Additionally, impulsive perturbations have effects only on bamboos. Therefore, the model of the growth rate of the giant panda $x_3(t)$ is eradicated from system (2) as well as the models of the impacts of giant pandas on bamboos. Like most of references such as [65], we are led to consider the properties of the periodically forced subsystems, which describes the dynamics of two resources respectively.

For the simplified subsystems, each periodic subsystem in the objective function, i.e., Formula (2), above is composed of a kind of bamboos using the growth rate of bamboos, impulsive functions and initial solutions as the target while the third subsystem is removed to minimize the difficulty of the recomputation. After that, each system will be simplified as a density-dependence logistic model (also termed a single-population model) with impulsive effects. Seeing from the view of impulsive models, this model is much more complex than that of system (1) due to the involvement of impulsive effects. Seeing from the view of considering the properties of the system, the models of two previous subsystems on bamboos will be much more simplified than that of the typical system due to the ignorance of the model of the third subsystem on the giant panda.

In this section, we shall introduce a few definitions and notations together with a few auxiliary results relating to the properties of solutions in the system and the stability theory for impulsively perturbed systems of ordinary differential equations. The biological well-posedness of the permanent problem associated to our systems for strictly positive initial data will also be stabilised.

(a) Existence and boundness of periodic solutions.

Existence: Based on the simplified IDDM model mentioned, a special solution of this model can be produced by means of the method of Reference [18]. First, it is easy to see that

$$u(t) = \frac{(a_{10}/a_{11})u(t_0)e^{a_{10}(t-t_0)}}{(a_{10}/a_{11}) + u(t_0)(e^{a_{10}(t-t_0)} - 1)}, t \neq (n + l - 1)T \quad (3)$$

For any solution u of the first equation in system (2), and so we can produce $u((n + l)T)$ according to this formula. Additionally, it can be seen from the first impulsive equation of system (2) that the upper-left derivative $u((n + l)T-)$ is equal to function $u((n + l)T)$, which implies that the upper-right derivative $u((n + l)T+)$ is equal to $(1 - \alpha)u((n + l)T)$. Using the same steps, it follows that $x_r^*((n + l - 1)T+)$ and $x_r^*(lT+)$ will be obtained, and we can see from the right-hand sides of new calculated functions x_r^* at $t = (n + l - 1)T+$ and $t = lT+$ that both formulas are the same, which shows that the function is a periodic function. Obviously, the periodic solution does indeed exist and is unique and strictly positive.

Boundness: Let $V(t)$ be the observed composite function. Similar to Reference [18], Shi [65] use the linear combination function $V(t)$ of $x_1(t)$, $x_2(t)$, $x_3(t)$ to verify the effectiveness of the system:

$$V(t) = \frac{a_{31}}{a_{13}}x_1(t) + \frac{a_{32}}{a_{23}}x_2(t) + x_3, t \geq 0$$

In the feasible domain, we can obtain that the derivative of this composite function as showed in formula (9) of Reference [65]. Recall that in the differential equation above, the sum of $\dot{V}(t)$ and $a_{30}V(t)$ will be less than a bounded maximum positive constant calculated according to formula (10) of Reference [65]. In general, the upper-right derivative of V at time t^+ with respect to the systems (2) is less than $(1-\alpha)V(nT)$, which implies that the solution $V(t)$ must be limited to a certain positive threshold only related to an initial solution and the obtained threshold. As verified in Lemma 3 of Reference [65], it is now possible to prove that the Cauchy problem with strictly positive initial data remain strictly positive and bounded on their whole domains. This completes the proof.

In the following subsections, we analyze the stability of the system from different perspective.

(b) Stability of the system.

The local stability of the system. The giant panda-free periodic solution $(x_1^*(t), x_2^*(t), 0)$ is produced using *dsolve function* developed by `Matlab`. If it does exist, one can also infer the stability of the system. Usually, the local stability of the periodic solution may be determined by considering the behavior of small-amplitude perturbations of the solution, drawing $(u(t) + x_1^*(t), v(t) + x_2^*(t), 0)$. And then substituting this perturbation solution into system (2), a linearization of the system shall be generated, drawing $X(t)=(u(t), v(t), w(t))$, which can be written as the formula of the matrix $(u(t), v(t), w(t)) = M(t) * (u(0), v(0), w(0))$ where all the variables in the matrix $M(t)$ are the derivative coefficients. By the integral operation, the matrix $M(t)$ is calculated to analyze the performance of impulsive models. And then, the resetting impulsive conditions of system (2) and matrix $M(t)$ are multiplied by each other, obtaining a new matrix. All of the eigenvalues of this matrix are recalculated in the form of expressions and are drew as $(\lambda_1, \lambda_2, \lambda_3)$ developed in formula (28) of [65]. For each eigenvalue of this system, it plays the role of determining the stability of the system. It is obvious that in general the system is stable if these three eigenvalues are greater than 0 and less than 1. Assuming the contrary, this implies that the system is not stable. Note that the local stability of the system can be analyzed only limited to the interval $[(n+l-1)T, (n+l)T]$.

The global stability of the system. Following the introduction of *local stability* in the above section, we infer that two subsystems about bamboos are stable in a periodic. Lemma in [40] have been proved that, for $t > 0$, if the derivatives, limits and initial values of the functions are less than previous right-hand side equations, then the limit of the difference between any solution and a periodic solution approaches to this infinitesimal number in this interval. Therefore, it is seen that two subsystems are globally stable with $t > 0$. After replacing the solution of each subsystem with the sum of a periodic solution and a infinitesimal number of two previous subsystems respectively, it implies that the derivative of the third subsystem is also less than the previous equation. Integrating this inequalities over $[(n+l-1)T, (n+l)T]$ yields $x_3((n+l+k)T) \leq x_3((n+l-1)T)\rho^k$ where ρ is one of the eigenvalues of the matrix mentioned, that

is, ρ^k approaches to 0 as k becomes 0. This also shows that $x_3(t)$ approaches to 0 as t closes to 0. Next, we need to prove that any solutions of each subsystem approaches to a periodic solution as t tends to infinity if the limit of the third subsystem approaches to 0. If we assume that $0 < x_3(t) < \varepsilon_3$ for all $t \geq 0$. By formula (40) and lemmas 2 and 3 of [65], then we have $x_1(t) \geq x_*(t) - \varepsilon_1$ for an infinitesimal number $\varepsilon_1 > 0$. Similarly, the second subsystem was also verified to achieve the same conclusion. This completes the proof.

(c) Permanence of the system.

In this part, we will introduce the permanence of the system. The purpose of studying the permanence is to incorporate the permanent condition into the IDDM model. Here we first make mention of the description of permanence before starting the permanence of the system.

Description. A system is said to be permanent if there exist two positive constants m and M such that every positive solution $(x_1(t), x_2(t), x_3(t))$ of the system satisfies $m \leq x_1(t) \leq M$, $m \leq x_2(t) \leq M$ and $m \leq x_3(t) \leq M$ for sufficiently large t .

Analysis Processes for the permanent system. For the right-hand side of the inequalities in *Description*, Georgescu have proved in Lemma 3.2 [18] the effectiveness of the inequality. For the left-hand side as mentioned in the above, in order to verify its effectiveness, Shi *et al.* [65] make a detailed analysis mainly from the following steps:

- By Lemma 4.1 from [18], the system has a periodic solution starting with strictly positive initial data. And a positive constant will be obtained after integrating the periodic solution.
- By Lemma 3 from [65], a solution is greater than the difference between a periodic solution and an extremely mini number if the derivative of a subsystem is greater than previous equations.
- We assume that a positive constant m is equivalent to this difference. By Lemmas 4.1 and 3, it has been verified that the first subsystem is greater than m . Similar to the case, the second one is also greater than m . This implies that the left-hand side of the inequality is feasible.

However, for the third subsystem, it is much more complex than the first two subsystems due to the correlation with previous ones. Given the conditions of the global stability, we conclude that the limit of the difference between the solution of each subsystem and a periodic solution is less than a sufficiently small number. In the same way as the analysis the global stability, we can see that the solutions of the third one will approach to 0 as variable t becomes 0. By Lemma 3, we infer that a solution $x_3(t)$ is larger than the called minimal number m . Let m be the minimum number among three infinitesimal number, and then all the solutions of system (2) are greater than m . This completes the analysis.

(3) Impulsive differential systems for applications in giant panda ecosystems

To demonstrate how the impulsive effects can be applied to differential systems, we review some examples of impulsive-based differential systems in this section. These models consider the change of a state abruptly. IDDM has found wide applications in

giant panda ecosystems. In this section, we briefly discuss a few more applications in giant panda ecosystems that benefit from IDDM.

(a) *Single impulsive perturbation analysis.* impulsive perturbation has long been a core problem in modeling giant panda ecosystem and is recently attracting more interest with the new advancements brought by in general. Reference [65] proposed the first impulsive-based theory model for modeling the giant panda ecosystem. In this model, there are only two objects considered such as giant panda and bamboo where the effect of a sudden collapse of bamboo as a food source is taken into account. Theory analysis showed that single impulsive model makes it possible to handle the discontinuous problem of a system effectively and efficiently.

(b) *Multiple impulsive perturbation analysis.* Multiple impulsive model extends single impulsive model by perturbing other species again. Reference [21] combines two types of impulses to jointly model *forest-bamboo-giant panda* nonlinear dynamic model. The first type mainly considers deforestation of saplings and trees, and the second one studies giant pandas' death due consideration for habitat destruction. Therefore, these models involve sapling-based impulsive model, tree-based impulsive model and giant panda-based impulsive model. Take the data of pandas in Wanglang national nature reserve as an example, an impulsive differential system is built based on local data information. It can be seen that at impulsive points $t_k = nT$, time series of seedlings and trees are badly damaged and the giant panda population tends to be extinct. More important, the periodic solution will be not met in case of $T = 7$. Using the same model but different impulse types as Reference [21], Reference [23] studied the bamboo flowering and the deforestation. As we can see in the graphs of giant panda and bamboo, compared with those of Section 3.1.1, time series of population density can more really characterized the evolutionary processes of species. It implies that the theory of impulsive differential equations is much richer than the theory of traditional differential equations without impulse effects. Note that usually in these models mentioned the performance of giant panda ecosystem model is verified according to the processing steps in Section 3.1.2. Besides, both left limit and right limit exist but are not equal, which implies that time series of the studied objects will keep periodic oscillation in the whole period.

3.1.3 Holling-based differential dynamic models

Ideally, the study of Holling-based differential models dates back to 1970s. However, pioneering work at that time was not widely adopted due to the lacks of scalability. To address this issue, there has been recent development since the first Holling-based differential dynamic model has been proposed [38]. Later on, there are essentially three types of Holling functions, HFRM-I (also known as the linear HFRM), HFRM-II (also known as the nonlinear HFRM where the proportion of bamboo consumed declines monotonically with bamboo density) and HFRM-III (also known as the nonlinear HFRM which is described by a sigmoid relation in which the proportion of the bamboo consumed is positively density-dependent over some regions of bamboo density).

More recently, differential dynamic models based on Holling and Sparsity effects prevail as a complete and accurate new scheme for modeling a giant panda ecosystem and have shown promise in getting a more accurate trend of population growth rate.

In this survey, we mainly focus on HFRM-II and HFRM-III. For details on HFRM-I, readers are referred to the classic Reference [12].

(1) Holling function

Holling functional response is the principle function of population density depending on predator density. As mentioned in the previous section, one of Holling function’s motivations is to model the uncertainty of the birth rate of giant pandas, which boils down to modeling the uncertainty related to ecological environmental disturbances.

Sparse Effect (SE). Traditionally, Holling function in [38] is followed by $b(t)X(t)/(1 + X(t))$, which can be seen as a type of nonlinear predator-prey system suitable for the large-scale population. According to the statistics of giant pandas in Section 2, it is listed as a rare species having the more sparse population density, which indicates that the change of individuals can easily lead to the fluctuation of giant pandas’ increasing rate. Usually to form a complete and accurate model, Holling function in $1 + X(t)$ will alter before modeling a giant panda ecosystem using differential equations based on Holling function. One classic example is the *sparse effect*, which alternates between 0 and a number produced by function $N(t)$ based on the sine or cosine. For $N(t)$, in general, we can define the following generative process of the function $N(t)$:

$$N(t) = N_0(|\sin(At)| - \sin(At)) \tag{4}$$

here, N_0 and A are positive constants. In addition, *sine function* can be also modified as *cosine function*.

- (1) For each variable t of Formula (2),
 - (a) For each value of $\sin(At)$ greater than 0, draw $N(t)=0$;
 - (b) For each value of $\sin(At)$ less than 0, draw $N(t)=2N_0\sin(At)$;
 - (c) For each value of $\sin(At)$ equal to -1, draw $N(t)=2N_0$;
- (2) For each item $N(t)$ equal to 0, a_{10} is equal to $b(t)$, which indicates that the parameter is the birth rate of current giant pandas.

Note that if t goes to infinity, this function will be still limited in a range centered at the interval $[0,2N_0]$, where the value N_0 is calculated previously according to the data from a region studied, and the model will degenerate into a function depending on variable t . This is why we call it "Sparse" Effect.

HFRM-II based SE [21]. The modified HFRM-II is formulated as:

$$a_{10} = \frac{b(t)X(t)}{N(t) + X(t)} \tag{5}$$

here, $b(t)$ denotes the birth rate coefficient of the giant panda, $N(t)$ denotes sparse coefficient, $X(t)$ denotes the number of giant pandas.

Compared with HFRM-I [29], HFRM-II is introduced the handling time for bamboos that is consumed into the model HFRM-II to handle the prey. For HFRM-I, all the time can be used for searching, that is, the searching time is the time that it takes; however, for HFRM-II, the searching time is equal to the difference between the whole time and the handling time, which implies that in the time interval T_t the total number

of prey is therefore reduced [12]. Note that $b(t)$ can be explained as a function of calculating a ratio of two characteristic times: the product is large if the handling time is much longer than the searching time and this product is small in the opposite limit; in this case the type II response reduces to the type I case.

HFRM-III based SE[22, 23]. Similarly, the revised HFRM-III is formulated as:

$$a_{10} = \frac{b(t)X(t)^k}{N(t) + X(t)^k}. \quad (6)$$

Noted that HFRM-II is a special example of HFRM-III. But the type III response is not so easily formulated just by separating out the handling and searching behaviour. In this model learning behaviour occurs in the predator population with consequent increase in the searching rate as more preys occur. Since at higher prey densities there will have been more predator-prey actions, the searching rate a increases monotonically with the number of prey by replacing a by aX^{k-1} , for some $k-1$, when preserving the saturation at large prey density X . In two types II and III, Y is the number of prey consumed by one predator, X is the prey density, T_s and T_t are the time available for searching and handling and a is 'the searching rate' by Holling.

(2) Holling-based differential dynamic models and Its Applications

Despite the successful applications of Holling function in differential equations, very few attempts have been made to develop differential dynamic models based on Holling functions for giant panda ecosystems. Due to the nature of rare giant pandas, two main types of models modeling giant panda ecosystems need to meet the following criteria:

- (a) Type 1: *Models based immature pandas and mature ones*. Commonly, the second one is used to construct the increasing rate of the birth rate of immature pandas with time. For mature, the change of the number of mature ones with time is modelled by a constant or a linear function.
- (b) Type 2: *Models based bamboo-forest-giant panda*. Because the giant panda is endangered, its birth rate is sensitive to the increase or decrease of individuals. Considering the restricts from the number of trees or bamboos causing that the increasing rate can vary toward another direction at some time, the third one is chose to model.

In Reference [21], typical linear models can be replaced with nonlinear and periodic structures to form a *HFRM II-based differential dynamic model*. In the modeling process, the method of Type 1 is chose to model immature and mature giant pandas in two different regions. This model is a density dependent model having four types-for region 1: immature and mature and for region 2: immature and mature. Additionally, the rate of transition from immature individuals to mature ones is also considered in this model. Because that the spaces between two habitats are artificially connected makes giant pandas spread and crawl back and forth between habitats, the net exchange of the mature population from a place to another one is added into this model. Given all the corresponding parameter values, the figure of time series of each type evolved in this model is given. From these figures, it can be seen that the introduction of Holling function make the increasing rate of the birth rate of immature pandas in these two regions

with time increase fluctuate in a decrease way, and finally close to 0, which indicates that this system reaches an *almost periodic stability* at this moment.

Although the increasing rate of the birth rate of giant pandas in this model will change in a continuously decreased way by involving both sparse effect and HFRM-II, they actually alter in the same direction, because they ignore the randomness of this model due to the uncertainty in real-life natural environment, which is crucial for accurately modeling giant panda ecosystems.

Both References [22] and [23] published by the same author uses the same HFRM-III based SE (Type 2) instead of the constant to perform x_1 of Formula (1). Introducing two periodic functions $b(t)$ and $N(t)$ into the models of [22] and [23] and solving differential equations using function `dsolve` in software `MATLAB`, the increasing rate of the birth rate of giant pandas will alter in a way that increase first, then decrease and finally close to 0. Similar to the curve shapes of HFRM-III based SE but different increasing rates, various variants having different k values have been modeled. According to on-the-spot investigation, HFRM-III based SE with $k=2$ can more slowly approach this growth rate equal to 0 compared with HFRM-III based SE with other k value greater than 2. Hence $k=2$ is the most common choice for a HFRM-III function based SE [22, 23]. Also, these two articles choose this value as their model parameters. Note that besides these two cases, it is possible for HFRM-III based SE to have some periodic solutions, in which case verifying the existence of these periodic solutions would be more complex. Given all the related parameters, `Mawhin coincidence degree theory` is used for proving that there exists a periodic solution in the systems [22, 23]. As the method of proof is similar with that in [20]. However, according to these graphics of time series of giant pandas, bamboos and trees, it can be seen that there are some limited conditions for the periodic solutions. For example, several periodic solutions can be broken when periodicity T is 7. Meanwhile, a chaotic strange attractor can be appeared in the system, which increases the difficulty of modeling giant panda ecosystems.

In this section, we covered how Holling effects can be applied in the models of the growth rate of giant pandas, respectively. We can see from three types of Holling functions that the growth rate of giant pandas almost approach to zero at lower density while that rate will reach a critical threshold making the system keep the stability at higher density.

3.1.4 Allee-based Differential Dynamic Models

Traditional models of population growth, such as the logistic model, incorporate inverse density dependence only. For logistic models, population growth rate are expected to increase as a population is harvested, because of the reduction in negative density dependence. However, it is evident that populations subject to Allee effects will show reduced growth at low population sizes. Indeed, under Allee effects, growth can actually become negative below a Allee threshold. The harvesting of populations under these conditions might have serious consequences. The study of the Allee effects dates back to 1931 with notable works from References [1, 68], but the first attempt to use Allee effect to predict the growth rate of giant pandas dates back to 2016 [91].

To better understand Allee effect, here we start with the simplest type of Allee formula as an example to describe how the Allee effect works as inverse density dependence increases. After that, we will review several other types of Allee effects based on measurable components of the fitness of giant pandas.

(1) Models

Allee effect describes a scenario in which population at low numbers are affected by a positive relationship between population growth rate and density, which increase their likelihood of extinction. Hence, a key point is choosing an analytical model that represents this relationship. In mathematical terms it can be stated as:

$$AE = \frac{A + c}{x + c} \quad (7)$$

Here A is the Allee threshold which is used for scaling Allee effect, c is an auxiliary parameter which effects the growth rate of the prey. The curve of the growth rate flattens slowly as parameter c increases. It is worth noting that there are two equilibrium points across the x -axis. Reference [81] have verified that for the left equilibrium point (lower density), the system is unstable while for the right, the system is stable. There will only produce Allee effect around the lower density.

Usually, there are essentially two types of *Component Allee Effect* (Allee effects manifested by a component of fitness) and *Demographic Allee Effect* (Allee effects which manifest at the level of total fitness). In this survey, we will focus on the study of *Component Allee Effect* and *Demographic Allee Effect*.

Component Allee Effect. A component Allee effect is some measurable component of individual fitness [69]. For 'component', the Allee threshold is quite small, which indicates that the increasing speed of the growth rate decrease slowly. Also, there is a positive relationship between population growth rate and density, that is, population growth rate increases with increasing density. However, a primary issue is to only study the effect of a single component on the growth rate of giant pandas, which ignored several corresponding relations between some components.

Demographic Allee Effect. The purpose of a demographic Allee effect is to study the overall fitness that involves an ensemble of several component Allee effects [69]. For 'demographic', there is a larger the Allee threshold compared with the above due consideration for the overall fitness, which determines that the increasing speed of the growth rate decrease quickly (relatively smooth). Different from the component, demographic Allee effect studies a positive relationship between an *average* growth rate and density at the low density to to evaluate the impact of Allee effect more accurately. As we will see below, the underlying idea of 'demographic' Allee effect is that an ensemble of the overall components is more than the sum of its parts in the sense, obtaining a per capita growth rate. For 'demographic' Allee effects applied in the models, the overall components determine the extend of Allee effect. Hence, a demographic Allee effect can be either 'weak' or 'strong' below.

Weak Allee effect. Population with 'weak' Allee dynamics experience lower per capita growth rates at low densities but never experience negative per-capita growth rates and therefore have no critical threshold to exceed. Take Allee effect formula (7) as an example, the 'weak' implies that threshold A is less than 0. Drawing a curve of the

growth rate, we will see that there is only one equilibrium where the per capita growth rate is negative above the carrying capacity (the equilibrium) and positive below.

It is worth noting that the equilibrium is the point where the growth rate is zero. Above this point, for negative Allee effect, the growth rate increases but the speed of the increase is decreasing and otherwise opposite. For positive, the growth rate increases and the speed is also increasing. More important, in position 0, the growth rate is greater than 0, which implies that the growth rate still exists even if the population density is 0 which goes against the principles of biological evolution. Hence, as $A \rightarrow 0$, the reduction in fitness owing to the Allee effect becomes insignificant.

Strong Allee effect. Population subject to a 'strong' Allee effect experience negative per capita growth rates when density falls below a critical threshold. The 'strong' implies that threshold A is greater than 0. Beside there are two equilibrium points here. For right point, its principles is the same as those of the weak. Hence, we only consider the inverse density dependence of the left. For left (lower population density), the growth rate also decreases above this point, and can even become negative below a critical population threshold, which implies that the population of giant pandas will go extinct. At this moment, the phenomenon of Allee effect is the most obviousness.

Discussion. The example model above demonstrates Allee effects that clearly describes the trends of population dynamics of the species including giant pandas and its theoretical consequences. By analyzing this model, it is seen that the modified parameters can only modify the slope of this relationship around the lower equilibrium but cannot modify the essential characteristics (e.g., the number of equilibrium points) of the curves of the growth rate of some species. In addition, for $A > 0$, the slope of the nonlinear equations on the growth rate always first decreases, next decreases to zero (parallel to the x-axis) and then increases as the density increases. Note that the above analysis is only for the lower population, especially endangered or rate population. Because the Allee effect describes a scenario in which populations at lower numbers are affected, it is suitable to use this method to model giant panda ecosystems due to the endangered and rare nature of giant pandas.

(2) Implementation of Allee effects for giant panda ecosystems

In the previous section, we analyze how Allee effects can be applied in reflecting the change of the growth rate of giant pandas at low population density. In this section, we will summarize some real examples about the implementation of *strong demographic Allee effect* (Allee effect for short) on giant panda ecosystems. It is worth noting that the permanence analysis of the studied systems is the same as that of Section 3.1.2. Hence, we will give a brief analysis of this parts.

In the field of modeling giant panda ecosystems, researchers have also been adopting the Allee effect techniques to improve both evaluation and interpretability of giant pandas. For example, considering the effect of diffusion loss on time-varying population, Zhang *et al.* [91] apply an Allee effect in a real case in Qinling Mountain giant pandas nature reservation area. For any model, it is also crucial to ensure models' existence on solutions and stability in the whole period. Hence, in the model considering two types of populations from two different patches, a method from Section 3.1.2 (a) and (b) is used to verify the existence and stability of the designed model. By adopting the approaches of References [18, 64, 65] making the system with Allee effect perma-

ment, based on data information of Qinling Mountain, we conclude about the permanent condition that this system with Allee effect is permanent if the differential equation of the growth rate of giant pandas is periodically bounded. Under these conditions mentioned, the sensitivity of the number of giant pandas to Allee effect can be analyzed. We can draw a conclusion that studies of the causal mechanisms generating Allee effects in lower density population could provide a key to understanding the dynamics of their survival or extinction.

Discussion. In sections 3.1.1-3.1.4, it can be seen that each type of differential dynamic models plays a key role in maintaining the feasibility of the systems. To solving the non-differential problems after suddenly disturbing, a impulse-based approach onto the differential model is crucial. Moreover, the Holling model and the Allee effect are all challenging in dealing with all kinds of problems although they have been widely used to solve several issues. Each type of the models has a certain limitations, but they have several advantages in modeling giant panda ecosystems. Hence, in order to more clearly show the features of each model, we will list a table to compare them shown as Table 3.

3.2 Age-Classified Matrix Models for Giant Panda Ecosystems

The computation in *Age-Classified Matrix Model* corresponds to a finite sequence of population size of this species per year. For the first time it was noticed in Reference [43]. Each prediction can be seen as a sequence of four steps: (1) equally dividing the complete life into age groups, where each group contains a plethora of individuals, (2) constructing a matrix composed of reproduction rates and survival rates of each group, and (3) multiplying this matrix or its t th power by an initial vector, and obtaining a population vector at moment t . The first step is the most difficult one, as there is no unified standard of division. As we show below, in some cases it is possible to apply the matrix model to study the population dynamic of giant panda. This allows us to compute survival individuals at next moment under various parameter combinations rather than only one parameter combination. Another theoretical advantage of such a model is a stable system whose a stable equilibrium is its positive eigenvector (explained as a stable population scale) and an unchanged matrix once divided equally, which indicates that the dimension the matrix is independent of population size. Hence, this model could be possibly applied for large-scale giant panda population. We would like to remark that the matrix models are not possible for without solutions, however, they are possible for these cases whose positive eigenvectors exist.

3.2.1 Preliminaries

An *age-classified matrix model* (ACMM) is a linear population dynamic classification model that involves multiple age groups. In mathematical terms it can be stated as

$$\mathbf{N}(t+1) = M^t \mathbf{N}(0)$$

where M is the square matrix composed of the breeding rates and survival rates of each group [28, 43], $\mathbf{N}(0) = \{n_1(0), n_2(0), \dots, n_m(0)\}$ is an age distribution at initial moment. Parameter t is set as the maximum iterations of the program. We can also

Table 3. Comparisons of the performance between four types of differential dynamic models for modeling giant panda ecosystems in several natural reserves

Items	Differential dynamic models	impulsive differential dynamic models	Holling-based differential dynamic models	Allee-based differential dynamic models
Data input form	Initial population size	Individuals of age groups	Single individual	Single individual
Captive or wild giant panda individual representation	(a) The alphabet represents the giant panda. (b) Its value is denoted as the number of giant pandas. Moreover, this model only considers the number of pandas regardless of individual behaviors.	(a) The alphabet→the giant panda individual; (b) Its value→the number of giant pandas.	(a) The alphabet→the giant panda individual; (b) Its value→the number of giant pandas.	(a) The alphabet→the giant panda individual; (b) Its value→the number of giant pandas.
Parameter types	(a) Birth rate and mortality rate; (b) The staple food of giant pandas: bamboos (natural growth). For parameter (a), we define parameters as constants; for (b), that can be denoted as a function, which indicates that the researchers need to study the change of bamboos over time and discuss the impacts of bamboos on giant pandas.	(a) Birth rate and mortality rate; (b) The staple food of giant pandas: bamboos (bamboo blooming and then die off), for (b), in contrast to the above model, the model of bamboo growth can be quite difference.	There are two approaches Holling II such as formula (5) and Holling III such as formula (6) to calculate the survival rate of giant pandas.	The computation approach of the survival rate can be defined as the form of formula (7).
Forms of solutions	The simplest form of solutions is defined as $N(t) = N_0 \exp(r_m t)$ where N_0 is an initial population of giant pandas and $N(t)$ is the population of this species at time t . In this formula, only consider the intrinsic rate of natural increase.	The expression of solutions can be more complex than that of traditional one.	The expression of solutions can be more complex than those of other models.	The expression of solutions can be more complex than those of any one model.
Properties	In general, the solutions of the designed model can be calculated.	Due to the occurrence of the "contingent event", the existence of period solutions, bound conditions and stability conditions should be discussed after modeling.	The same steps.	The same steps.
Purposes of modeling	Use the simple differential equations to describe a series of changes of populations of giant pandas over time.	Deal with discontinuities of the previous differential models encountered by the termed contingent events such as bamboo flowering at some moment.	Handle the problems the survival rate keep unchanged during the whole processes. The purpose is to ensure a positive correlation between population size and survival rate before the population reaches saturation.	Handle this problem that the survival rate did not change over time when reaching the saturation. The purpose is to achieve such an effect: on the basis of this saturation, with the increase of the population, survival rate can decrease.
Difficulties	Modelling with differential models is difficult contrasting with the traditional statistical approach, and the precision of the simulation results. It usually cannot handle the contingent events, which limits the application scope of initial models.	Modeling with impulsive differential models is difficult comparing with original models. It usually requires users to handle contingent phenomenon, discuss the existence of period solutions and its stability etc.	Setting the index k can be the most difficult part when developing the function of survival rates, since this index can determine the shape of functions. Note that the entire trend of the model keeps the same that first increase and then balance.	Controlling Allee threshold A can be the relatively difficult work when attempting to change the equilibrium state, since this threshold determine the stability of the models. Need to discuss the range of this threshold: if greater than 0, there are two equilibrium points; else, one or instability.
Application	The type of models have been developed and successfully applied to modeling giant pandas.	Several applications of impulsive-based differential models for modeling this species have been achieved though this model can be difficult to explore feasible solutions.	There are only two papers published to studying the impacts of the Holling functions on the change of populations of giant pandas.	There is only one paper published to consider the Allee effect on modeling giant panda ecosystems.

consider the population distribution as a vector $N=(N(0), N(1), \dots, N(t))$ where the latter is derived from the former. After reaching the halting condition, a vector $N(t)$ is termed as an outcome [42].

When corresponding variants as well as $N(t)$, $0 \leq t \leq T$, are linear, we speak about a *linear population dynamic classification model*. We also remark that for matrix model M undoubtedly it kept unchanged during the entire operations. However, variants of matrix model M such as M^t can not be feasible, which implies that we further need to observe new solved matrix models to ensure the solvability of linear problems. In addition, it is important to note that all individuals of each group will be transferred to the group next to it. For the first group aged as 0, many individuals borned in other groups are added to this group. For maximum group, these individuals whose ages reach the complete life are removed from this group. It is possible to halt the program once the number of individuals of all groups is zero. It implies that the giant panda goes extinction.

In the process of predicting population dynamics using matrix model, typically there maybe not feasible matrix that does not make the program produced an outcome N_t . Hence, another attention is focused on the feasibility of matrix model, which indicates that it is necessary to further compute the positive eigenvalues and eigenvectors of the matrix. Due to the more complex of matrix M , Reference [43] simplifies matrix M for the purpose of reducing the actual computation. Formally, they are defined as optimization of the computation steps to linear problems, which are also called *Matrix Reduction*.

Example 3.1. For each $N(t)$, its each element can be constructed into

$$\left\{ \begin{array}{l} n_{0t} = \sum_{x=0}^m F_x n_{x0}, \\ n_{1t} = P_0 n_{0,t-1}, \\ n_{2t} = P_1 n_{2,t-1}, \\ \quad \cdot \\ \quad \cdot \\ \quad \cdot \\ n_{mt} = P_{m-1} n_{m,t-1} \end{array} \right.$$

where F_x is denoted as the breeding rate and P_m is noted as the survival rate. Usually, they are constants. Considering the existed natural disturbance phenomenons, it is translated into mortality rate, thus impacting the survival rate of this species.

For the corresponding integer vector $N(t)$, we note that it is $(n_{0t}, n_{1t}, n_{2t}, \dots, n_{mt})$ where $n_{mt} \geq 0$. Note that sum of them is the number of giant pandas.

3.2.2 Classification of Age Group

It is not difficult to see that one of the primary problems of the computation of matrix models can be simplified as solving the classification problem of ages. Age classification problem can be described as an operation that the complete life can be divided into several age groups whose intervals are the same according to a certain rule. For simplicity, we consider that the result of the classification are changeless during the whole simulation and has only one form. As we know, it can be simpler and quicker

to operate than dynamic forms. For static age classification, in general there are the following types such as sections 4.2.1 and 4.2.2

(a) **Unit classification** [43]. *Unit classification* is denoted as such a classification whose age interval is equal to 1. Hence, the number of age groups equals to half of the maximum age. The advantage of the approach is that for each transformation all the individuals are transferred into next age group synchronously. This behavior can be easier and more convenient to compute population dynamics of pandas. In present published references, its two forms contains:

Unitpoint-based method. The form of its interval is $[x, x + 1]$. That is, all individuals unites to evolve starting at initial moment t and ending at moment $t + 1$. Note that at every moment, all individuals from each age group experience a phase about breeding and dying off. After involving, these survivors aged as $x + 1$ will be transferred into next age group. Other individuals aged as x are in the current group even if increasing one year older. It is noticed that the birth rate and mortality rate of giant pandas are further subdivided, which is more in line with the actual giant panda survival conditions.

Midpoint-based method. The form of its interval is $[x + \frac{1}{2}, x + 1\frac{1}{2}]$. That is, during the interval of time 0-1 some of these individuals are dying off, and at $t = 1$ the $n_{x+1,1}$ survivors can be regarded as concentrated at the age $x + 1\frac{1}{2}$, so that at this latter time the number of individuals alive in the age group changes abruptly. The advantage of this approach is that the number of the dead individuals can be first calculated at moment $t_{(x+\frac{1}{2}, x+1)}$ and that of the survivors are at moment $t_{(x+1, x+1\frac{1}{2})}$. Compared with the above method, this approach is more suitable for female individuals in breeding season.

(b) **Equal interval classification(EQIC)** [28]. To handle the problem suffered by the above approach that causes an amount of empty sets in several groups, EQIC is developed. Usually, EQIC means that its each age interval is the same and greater than 2. At each moment, several oldest individuals from the left-hand age group are sent to current one while several individuals from current group are sent to the right-hand one, so as to form a new set of age groups. More complex, for each same group, it is necessary to consider the reproduction and mortality state of individuals except for transferring. For new birth individuals, they will be sent to the initial group aged as 0. After a complete transformation, a new age group is generated. This process is called *transformation of age group*. The purpose of studying EQIC is to make the whole individuals be assigned to different non empty groups. Due to the limits of real giant panda data, this approach can usually be a better choice to classify the age groups.

Example 3.2. Consider the complete life L , with the size of each age group m where $m \geq 2$. Then, there are $ceil(L/m)$ age groups. Directly, the individuals whose ages are in interval $[x, x + m]$ are divided into the same group. Hence, we called $\{[0, m], \dots, [x, x + m], \dots, [L - m, L]\}$ as a set of age groups. In this set, we termed $[0, m]$ as an initial age group, and $[L - m, L]$ as a maximum age group.

3.2.3 ACMM for Giant Panda Ecosystem

In Reference [28], *Leslie Matrix* is used for predicting and analyzing population dynamics of giant pandas in Fuping Natural Reserve. In this reserve, there are 64 giant pandas, where there are 24 giant pandas under the age of 6, 31 pandas between 6 and 15 and 9 individuals greater than 15.

According to the growth law of giant pandas, the maximum age of wild giant pandas is 26 year old. Hence, that would be divided into 9 age groups if 3 year old is as an age phase. Meanwhile, let $[0,2]$, $[3,5]$, ..., $[24,26]$ be 1st group, 2nd group, ..., 9th group.

In addition, Reference [28] can calculate the reproduction rate and the survival rate of each age group (see Table 1 in [28]) according to the data of giant pandas from Reference [30]. Note that usually the sex ratio of giant pandas was set as 1:1 because of the difficulty of recognition for wild giant pandas.

Based on the above conditions, Reference [30] deduced a stable age structure of giant panda population ($N(\infty)$ in [28]). And then a non-equidistant age structure can be adjusted as an equidistant one according to the ratio of stable age groups. Hence, we get an initial age distribution of giant pandas in Fuping natural reserve $N(0)$. Take population vector $N(0)$ of 1990 and Leslie matrix M as input and choose the formula of Section 3.2.1 as iteration model, we would estimate population dynamics of giant pandas in 9 age groups in the next 33 years. Simulation results shows that the number of giant pandas during 33 years increased about 30, 1.353 times of previous population size. This states that population size of this species can increases slowly, which is in line with the changes in population size of real animals. It is worth noting that adding additional individuals to this model does not lead to the redistribution of the age group of the model. This allows to achieve a strong prediction of population dynamics of this species in the next several years.

3.3 VORTEX Models for GPE

With the advent of software that realizes the idea of modeling real ecosystems by programming, conceiving a novel software simulating the complex dynamics of natural populations of giant pandas is no longer an exceedingly hard problem. The first attempt to use software VORTEX to model giant panda ecosystem dates back to 1997 [84]. Since then, the structures of different VORTEX simulation models for population viability analysis are outlined, considering environmental factors influencing the population dynamics of giant pandas in these models.

Vortex models changes to a population as a series of discrete events that occur once one year (other time interval) [39]. In the Vortex software, the procedures of these demographic events have been programmed: breeding, mortality, migrate (disperse) among population, supplementation, carrying capacity truncation. Occurrences of events are probabilistic. Demographic stochasticity emerges from change variation in which individuals breed or die. Environmental variation in demographic rates is imposed by sampling rates from specified distributions during each simulated year. Catastrophes, which occurs with specified probabilities, cause one-year reductions in reproduction and survival.

3.3.1 VORTEX-based Models

In the Vortex-based simulation approach the biological data of (real) giant pandas as input are physically located in different positions of a three-dimensional matrix (population, sex and age), while other parameters such as demographic rates (usually due to environmental variation) and inbreeding depression are sequentially stored in different

positions of several two-dimensional matrices, and then the software starts to simulate according to a series of evolutionary rules. The biggest problem is to effectively predict the population dynamics of individuals under various factor depressions in the software, as this is an uncontrollable external interference. As advantage, the designed model is highly scalable and robust. Below, we give the operation processes of the Vortex software simulating the giant panda ecosystem.

(a) *The generative process of this models.* In theory, the vortex models are a set of software simulation systems programmed by C language without the specific mathematical models and the constraint conditions limiting models. In this system, matrices are used for storing input/output data and variables are used for storing biological parameters, facilitating calculations and accumulating sums and other components needed for the basic statistics reported in the output. Additionally, each event is encapsulated as a functional module achieving the related task. And then the main modular called these modules and at last output the calculated results. To avoid the waste of the multiple memory storing the biological data of each iteration, in general a memory is repeatedly used for storing the objects at different moments of iterations, that is, the previous data can be replaced by these data at next moment. Hence, the space is limited to current data information storage and there is no increase or decrease of space length due to store the number of individuals rather than object symbols.

For the representation of the giant panda individuals, there is no actual form to express. In general the rows of arrays represent the sexes of giant pandas, the columns represent the ages of giant pandas and the values of arrays represent population size of giant pandas. The order of handling giant panda data is in accordance with the order of the life cycle. The recognition of individuals is directly realized by querying rows and columns of this array. An evolution rule defined here is in the form of $m[i][j+1] = m[i][j] * p_k$, where m is the array storing the population size of giant pandas with the same gender i ($i=1, 2$, where 1 is the male and 2 is the female) and the same age j , p_k is a survival rate. A particular module is designed to determine which demographic events are added into the model. For each type of catastrophe, if and only if a random number is less than its frequency of occurrence, this catastrophe occurs and the fecundity and survival rates for years in which a catastrophe occurs are obtained by multiplying those rates in a "normal", non-catastrophe year by the specified factor. Note that it is impossible for a matrix to denote a specific individual of the population. It implies that for each iteration, these individuals with the same sex and the same age as a whole can evolve synchronously in a given probability. The biggest problem is that we cannot observe every panda is in a survival or mortality state being important to estimate individual genes thus prohibiting inbreeding. Similar to *unit classification* of Reference [43], at the next moment all the individuals are sent into the right-hand adjoined region. To avoid the shared memory conflict, all the individuals reaching the maximum ages need to be removed from the memory.

The iteration of simulation of software Vortex is realized deterministically, which is different form other models. The consecutive iteration of simulations is regarded as the evolution process. The evolution process is decomposed into the evolution of many sub-modules. In Reference [39], Lacy *et al.* gave the primary components of the Vortex simulation. Input all the available biological data through annual census

to the Vortex software, the function module *catastrophes* is performed in terms of a predefined catastrophe type. If a selected catastrophe occurs, the mortality rate of the studied animals can increase while survival rates decreases. And then other modules such as harvest of individuals, supplement with new individuals, and so on, would be performed in sequential order. During evolution, it is necessary for Vortex to determine whether the population become extinct or it closes to the limitation *carry capacity*. If not, continue. When the halt condition is reached, all the individuals at previous moment are removed after completing a simulation while these individuals surviving at next moment are stored in the adjoined memory in turn. All the memories are updated in line with associated primary steps. And then output mean population size of this species.

(b) *Discussion*. According to the steps mentioned, the probabilistic parameters play a key role influencing population dynamics of some species. Hence, it is necessary to discuss the following issues.

- *The method generating random*. In VORTEX, random numbers are classified as the following types: random integers between 0 and 64k and random real numbers between 0 and 1. Random integers are generated by the algorithm programmed by Kirkpatrick and Stoll [36]. Random real numbers are generated by dividing an integer by 64k. Random numbers from a standard normal distribution are generated by the polar algorithm [41]. Random numbers from a binomial distribution are generated by first calculating the cumulative probability distribution for the discrete outcomes, then generating a random real number, and final assessing which binomial outcome covers the portion of the distribution encompassing the random real number.

- *The method obtaining the rates*. The rates can be specified as functions of age, sex, inbreeding, population size, year and population, *etc*. The functions are classified as two forms: if it is a fixed constant, then the rates in each stage are the same; if it is a mathematical formula that can specify a demographic rate as density-dependent, or a function of other population parameters, then these rates can change over time. For example, reproductive rate can be specified to decline in the older age classes, adult mortality could be specified to increase with catastrophes no matter what anthropogenic disturbance or nature disasters, or habitat could be specified to decrease over time.

- *The problem of how to select some individuals breeding*. The software VORTEX can also deal with the breeding problems. Firstly, all the females who are older than breeding ages are put into the breeding pool. Secondly, the proportion of males breeding are also entered the breeding pool. If the proportion in the breeding pool is given directly, VORTEX will assume that the distribution of male reproductive success follows a Poisson distribution. Else, this proportion needs to be calculated according to the rules designed in advance. Only this can the individuals breed successfully within a certain probability.

- *The problem of how to evaluate environment variation (EV)*. VORTEX evaluated the effects of EV that arise from environmental conditions on the breeding rate and mortality rate using the distribution functions. In general, EV is modeled as a binomial distribution or as a normal distribution, which are determined by the magnitude of EV. Because the binomial distribution has a standard deviation closest to the desired EV, the binomial distribution is often used for EV. However, when the number of individuals

exceeds a certain scale, this distribution will often have a slightly fluctuation, thereby causing the deviation. In such case, the normal distribution will be used to model EV. Because the rate is restricted to the interval 0 and 1, to avoid creating any bias in the mean demographic rate, VORTEX needs to truncate the distribution symmetrically. It is noted that this truncation can result in EV to be substantially less than intended by the user. Moreover, if the truncation is not symmetric, then the mean demographic rate given by the model can be strongly biased away from the input parameter.

3.3.2 Applications of Vortex Models for Giant Panda Ecosystem

The summary of research status of the Vortex models in six study areas such as *Wolong nature reserve*, *Qinling mountains*, *Yele nature reserve*, *Tangjiahe* and *Mabian regions*, *Liping region* and *Baoxing region* as showed in Tables 4, providing a simple time line to analyze the research process of the Vortex model in every area. For the sake of modeling using the Vortex model, we summary the occurred catastrophes in these six areas in Table 5. The experimental results of six reserves ran by the Vortex software are concluded in Table 6. The overall processes from the modeling steps to the simulated results in this section presents an entire analysis about the use of the Vortex software in this species.

It is worth noting that the varied application domains of the giant panda ecosystem model VORTEX have always presented a challenge to the scene condition configuration of each domain due to the complexity of the environment. The current study divided the external interference factors into nature disasters, human interventions, environment limitations and their associated relationships. In this part, we organize the existing research on these factors affecting giant panda population dynamic. In all the models simulated by VORTEX software, the scenes are classified as four grades according to the geographical environment: environment fluctuation/carrying capacity(EF/ECC), density dependent/independent (DDT/DIT), inbreeding between giant pandas (INB), immigration and dispersal (IMD), catastrophes (CATA)(*bamboo flowering*, *forest firing*, *etc.*) and human intervention (HIV), as discussed in the following literatures. The environmental conditions of all the studied areas are analyzed in Table 5.

3.4 Membrane Computing Models

Membrane computing model is an active bioinspired field initiated by Păun [55], who discussed computing models motivated by the behavior and structure of cells, understanding the processes that take place in the compartments as computations. All the computing devices considered in this paradigm are called *P system* [55]. A probabilistic approach to P systems, also termed *population dynamic of P systems* (PDP systems for short) is introduced for studying the dynamic of (real) ecological populations [90]. And then we will focus on the PDP systems for modeling giant panda ecosystems.

With the preliminaries on population dynamic P systems, we are now ready to give the informal description of some main variants of PDP systems that were targeted for modeling and list some specific examples of PDP systems studying other species, *e.g.* bearded vulture, pyrenean chamois, the bacterium *Vibrio fischeri*. In this section, we mainly focus on PDP systems used for modeling giant panda ecosystems.

Table 4. The summaries of applications of Vortex models in six study areas

Reserve	Application domains
<i>Wolong nature reserve</i> [84]	The most early researched domains is in Wolong Natural Reserve. The challenge here is to firstly use the VORTEX model to analyze giant panda ecosystems in a specified region, which typically has studied a single-region giant panda ecosystem.
<i>Qinling study area</i> [47, 79, 96]	Qinling Mountains are another domain where giant pandas are firstly studied using VORTEX models in 1997 [47, 96]. Here again, giant pandas in this area are studied in 2002 [79]. But the objectives of the research are different where the second is the supplement of the first two researches, that is, the present research on the impacts of population density on population size under the conditions of previous environments.
<i>Yele nature reserve</i> [25, 27]	Yele Nature Reserve, belonging to one of Xiaoxiangling mountains, is yet another area where small population are subject to large fluctuations at a variety of levels, and such variation can put a population at a high risk of extinction. Practical examples of Yele area analysis can be seen in [27, 25]. This introduces the basic information of this area such as the geographical distribution, survival status of giant pandas as well as nature disasters to habitats.
<i>Tangjiahe and Mabian</i> [60, 93]	Both two regions Tangjiahe Nature Reserve and Mabian Dafengdeng Nature Reserve are studied in 2002 because giant pandas in these two areas are on the brink of extinction under inbreeding and bamboo blooming.
<i>Liziping study area</i> [98]	In 2008, Liziping Nature Reserve, also belonging to one of Xiaoxiangling mountains, has been firstly set up as the research base of giant panda ecosystem and have been utilized in the data VORTEX modeling efforts. Two types of nature scenarios constitute the focus of the research: Liziping and Wutuo County where there is no corridor between these two areas. Giant pandas mainly live in the middle or upper part of the mountain slope.
<i>Baoxing study area</i> [34]	Baoxing county is located in Ya'an, Sichuan Province. It has a vertical climate from subtropical to permafrost due to the influence of mountain altitude. In this area, there is a meta-population consisting of three sub-populations, which each population can be distributed by different levels of external factors.

Table 5. The summaries of environmental conditions of six study areas.

Reserve	Application scenes
Wolong nature reserve [84]	Wei <i>et al</i> developed an assessment of giant pandas in Wuyipeng based on four combinations, <i>e.g.</i> , INB and CATA, genetic load=1 without CATA, CATA (=1.67%) without IBD, and CATA (=1.67%) and genetic load=1.
Qinling study area [47, 79, 96]	The research areas are various Nature Reserves of Qinling Mountains in these three references. In [47], the scenario is Foping Nature Reserve. In this scenario, Li <i>et al</i> discussed the impacts of single-CATA on population dynamic, <i>e.g.</i> , EF/ECC, MD, CATA involving poaching, bad weather, predator, disease, <i>etc</i> ; In [79], Wand <i>et al</i> analyzed the impacts of DDT and DIT on local population; In [96], Zhou <i>et al</i> studied ECC, Poaching, IMD, Reproductive cycle/age range (extending).
Yele nature reserve [25, 27]	Guo and Hu present an evaluation of giant pandas in Yele area based on four combinations, <i>e.g.</i> , IBD and CATA, IBD without CATA, CATA without IBD, no IBD and no CATA, in the context of more complexity environments.
Tangjiahe and Mabian [60, 93]	Zhang and Hu indicate some emerging trends different from previous studies affecting the number change of individuals in the general Tangjiahe area. They recognize that as in many application scenes, such as without IBD and CATA, with MD, with EF/ECC and with CATA.
Liziping study area [98]	Zhu <i>et al</i> mainly considered two combinations, <i>e.g.</i> , CATAs including bamboo flowering and forest firing, and adding IBD to the aforementioned combination.
Baoxing study area [34]	Jiang and Hu discussed four major factors affecting population dynamics of the giant panda, <i>e.g.</i> , IBD (equivalence coefficient = 3.14 and 1.57), ECC (upward and downward), CATAs involving bamboo blooming (cycle=60a) and forest firing.

Table 6. The Comparison of population size of giant panda ecosystems in six study areas modeled using the Vortex models

Reserve	Simulation Result Analysis
Wolong nature reserve [84]	Reference [84] used VORTEX to model the trends of the Wuyipeng giant panda population in 100 years under the aforementioned scenarios. The results showed that a combination between CATA such as bamboo flowering and IBD caused a 4.21% decrease in population for 5 years, which verified CATA and IBD in Wolong significantly affect results, even if two parameters are minor.
Qinling study area [47, 79, 96]	The population size is sensitive to the outside interference. Based on this, in [47], the factors on EF/ECC, bad weather, predator, disease except for CATA and IBD appeared in Qinling Mountain are added into the specific model to verify the state of population dynamic under the aforementioned factors. The results showed that in this area the population of giant panda is approximately stable in 100 years, only with a small trend of decreasing. To enhance the feasibility, [79] and [96] also discussed the population dynamic under the given constraints over 200 years. Showing that the giant panda population would go extinct, even if the initial conditions are modified.
Yele nature reserve [25, 27]	An issue that needs to be considered in the modeling scheme is the years of population extinct with time. Guo <i>et al</i> [25, 27] simulated the trends of population dynamic in Yele in 100 years. Showing that there is a stable increase in population without consideration for CATA and IBD; only considering IBD, giant pandas went extinct within 90 years; considering IBD with 1.76% of CATAs, giant pandas went extinct within 60 years, and others are similar.
Tangjiahe and Mabian [60, 93]	Another issue that needs to be considered is how to avoid this extinction of giant panda due to different levels of interference in years. Ren <i>et al</i> [60] analyzed the population dynamic of giant panda from Mabian dafengding reserve. The simulation shows that population size of giant panda for the next 100 year is to decline under the specific scenes. Zhang and Hu [93] added individual supplement measures into this model. The results verified that the number of individuals increased slightly in 100 years through the timely supplement of new individuals from another domain when distributing due to the aforementioned disasters.
Baoxing study area [34]	More recently, advanced techniques such as VORTEX 8.42 with more components have been employed in Baoxing county to actually predict the metapopulation size of giant panda suffering from CATA and IBD in 100 years. The results showed that each subpopulation of the metapopulation suffered a certain interference from the random fluctuation of environment, and the difference among three subpopulations can be more obvious.

3.4.1 A Brief History of PDP systems

PDP systems is regarded as a variant of membrane computing models. One of its characteristic features is that PDP system imposes a probability on its rewriting rules, aiming to capture the inherent randomness of the processes to be modeled. During the execution phase of the systems, such rewriting rules are then implemented to produce final survival real ecological populations. In general such a system can be seen as theoretically predicting the dynamics of the species (increasing or decreasing number of individuals over time), and then simulating this system through simulator MeCoSim.

The study of Population P systems dates back to 2004 with notable works from Reference [2], although this initial work featured an approach quite different from PDP (tissue-like structure, catalytic rules and no probabilities, instead of cell-like arrangement and probabilistic rules). Over the years, a large body of works [32, 33, 73, 74, 90] have emerged to enable substantially better scalability and incorporate recent advancements of PDP systems. In a survey, there are 18 papers that have been published since 2004 [13]. Specifically, PDP systems can be classified into categories: *cell-like PDP systems* (hierarchical arrangements of membranes corresponding to trees) and *tissue-like PDP systems* (nets of membranes corresponding to graphs). The significant case studies of real ecosystems mainly focus on *scavenger birds* from Spain, *zebra mussel*, *pyrenean chamois* and *giant pandas* from China. For many species and their living environment, usually the types of membrane structures are different from each other. Due to PDP system's various structures, the term "PDP system" sometimes specifically refers to "single- or multi-environment P systems" [7, 10, 90]. In this study, the probability is determined by a random number generator following the binomial distribution function. To see this, note that the semantics in PDP systems implies making several rules selected in a probabilistic way to imitate the uncertainty of species evolution.

Today PDP system is gaining more and more popularity, has found successful applications in areas such as giant pandas, and appears as the theme of various conference workshops (e.g., the CMC workshop, the ACMC and the WMC). Interestingly, through PDP systems started in 2004, the study of PDP systems in modeling giant panda ecosystems started roughly in 2017 [33, 90]. In this paper, we will introduce an overview of PDP systems for giant panda ecosystems.

3.4.2 PDP Systems and Several Examples

As pointed out in References [7, 9, 10, 76], most types of (static structure) PDP systems can be seen as variants of membrane computing models. Since rules originating from membrane computing modes are not practical for modeling ecological data and predicting, a probability-level concept called *population dynamic P system* was introduced in Reference [2]. This model augments object rewriting and the flexibility of rules choosing with the notion of spatial locations (membranes) as well as the corresponding operations and it can be seen as a particular interpretation of the environmental uncertainty. References [7, 9, 10] define some basic probability rules measuring the possibility of specific behaviors of giant pandas in terms of PDP systems and give several examples of ecosystems.

Below, we provide the definition of single environment PDP systems on the giant panda ecosystem, taken from Reference [74] and multienvironment PDP systems, taken

from Reference [90]. We remark that two types of PDP systems are put it together to introduce because the model derived from Reference [90] is an extension of that derived from Reference[74].

Definition 1. A multienvironment P system and T time units, is a tuple

$$\Pi = (G, \Gamma, \Sigma, T, \mathcal{R}_E, \mu, \mathcal{R}, \{f_{r,j} \mid r \in \mathcal{R} \wedge 1 \leq j \leq n\}, \\ \{\mathcal{M}_{i,j} \mid 1 \leq i \leq m \wedge 1 \leq j \leq n\}, \{E_j \mid 1 \leq j \leq n\})$$

where:

- $G = (V, S)$ is a directed graph with $V = \{e_1, \dots, e_n\}$.
- Γ and Σ are alphabets such that $\Sigma \subsetneq \Gamma$.
- T, m, n, p are natural numbers greater than 1.
- \mathcal{R}_E is a finite set of rules of the type $(x)_{e_k} \xrightarrow{f} (y_1)_{e_{k_1}} \cdots (y_h)_{e_{k_h}};$
 $(\Pi_j)_{e_k} \xrightarrow{f'} (\Pi_j)_{e_{k_1}}$, being $x, y_1, \dots, y_h \in \Sigma, (e_k, e_{k_l}) \in S, 1 \leq l \leq h,$
 $1 \leq j \leq p$ and f, f' computable functions over $\{1, \dots, T\}$.
- μ is a rooted tree with q nodes injectively labeled from $\{1, \dots, m\} \times \{0, +, -\}$
($[]_i^\alpha$ denotes a membrane labeled by (i, α)).
- \mathcal{R} is a finite set of rules of the type $r \equiv u[v]_i^\alpha \rightarrow u'[v']_i^{\alpha'}$, being u, v, u', v' finite
multisets over $\Gamma, (u \neq \emptyset \text{ or } v \neq \emptyset), 1 \leq i \leq m$ and $\alpha, \alpha' \in \{0, +, -\}$.
- $f_{r,j}$ is a computable function over $\{1, \dots, T\}$, for each $r \in \mathcal{R}, 1 \leq j \leq n$.
- $\mathcal{M}_{i,j}$ is a finite multiset over Γ , for $1 \leq i \leq q, 1 \leq j \leq n$,
- E_j is a finite multiset over Σ , for $1 \leq j \leq n$,

A multienvironment P system of degree (m, n, p) with T time units can be viewed as a set of m environments e_1, \dots, e_m interconnected by arcs from a graph and a set of cell-like membrane systems, $\{\Pi_j \mid 1 \leq j \leq n\}$, having the same working alphabet Γ , membrane structure and set of rules. Each environment e_j initially contains a finite multiset E_j of objects from Σ , and also contains some membrane system Π_j , whose initial multisets $\mathcal{M}_{1,k}, \dots, \mathcal{M}_{q,k}$ depend on e_j . The membranes from Π_j initially have neutral charge. For the sake of simplicity, neutral polarization will be omitted.

A multienvironment P system has two types of rules: a set \mathcal{R} of rules associated with the cell-like membrane systems, and a set \mathcal{R}_E of rules associated with the environments. Every rule r is associated with a computable function ($f_{r,j}, f_r$, or f'_r), that provides the affinity of the rule, to be taken into account when several applicable rules compete for the available objects.

A *configuration* of the system at a given instant t is a tuple that contains: (a) the multisets of objects over Σ present in the environments; (b) the membrane systems and the corresponding polarizations and multisets of objects over Γ , contained on each environment; and (d) the values of functions associated with the rules of the system.

A rule $r \in \mathcal{R}$ of the type $u[v]_i^\alpha \xrightarrow{f_r} u'[v']_i^{\alpha'}$ associated with Π_j is *applicable* to a configuration \mathcal{C} at a given instant t , if the membrane i from Π_j has polarization α in \mathcal{C} , it contains the multiset v , and its parent membrane (the environment, in case i is the skin membrane) contains multiset u . When the rule is applied: (a) multisets v and u are removed from the respective regions; (b) at the same time, multiset u' is added

to the parent membrane of i and multiset v' is added to membrane i ; and (c) the new polarization of membrane i will now be α' (which might be the same as α).

A rule $r \in \mathcal{R}_E$ of the type $(x)_{e_k} \xrightarrow{Pr} (y_1)_{e_{k_1}} \dots (y_h)_{e_{k_h}}$ is applicable to a configuration \mathcal{C} of the system at a given instant t , if the environment e_k contains object x in this configuration. When the rule is applied the object x is removed from environment e_j and environments e_{k_1}, \dots, e_{k_h} get new objects y_1, \dots, y_h respectively. A rule $r \in \mathcal{R}_E$ of type $(\Pi_j)_{e_k} \xrightarrow{Pr'} (\Pi_j)_{e_{k'}}$ is applicable in environment e_k if this environment contains P system Π_j . When the rule is applied, Π_k move from e_k to $e_{k'}$.

A *transition* from a configuration \mathcal{C} to a *next* configuration \mathcal{C}' is obtained by applying the rules of the system in a simultaneous, maximal and non-deterministic manner. A *computation* of Π is a (finite or infinite) sequence of configurations such that: (a) the first term of the sequence is the initial configuration of the system; (b) each non-initial configuration of the sequence is obtained from the previous configuration by applying rules of the system in a maximally parallel manner with the restrictions previously mentioned; and (c) if the sequence is finite (called halting computation) then the last term of the sequence is a halting configuration (a configuration where no rule of the system is applicable to it). All computations start from an initial configuration and proceed as stated above; only halting computations give a result, which is encoded by the objects present in the output region i out in the halting configuration.

Definition 2. A *population dynamics P system (PDP system, for short)* of degree (m, n) , where $(m, n \geq 1)$ and having $T \geq 1$ time units, is a multienvironment P system of degree (m, n, p) and with T time units

$$\Pi = (G, \Gamma, \Sigma, T, \mathcal{R}_E, \mu, \mathcal{R}, \{f_{r,j} \mid r \in \mathcal{R} \wedge 1 \leq j \leq m\}, \{\mathcal{M}_{i,j} \mid 1 \leq i \leq q \wedge 1 \leq j \leq m\}, \{E_j \mid 1 \leq j \leq m\})$$

satisfying the following conditions:

- At the initial instant, each environment e_k contains exactly one cell-like membrane system, which will be denoted by Π_k . Therefore, the number p of P systems matches the number n of environments.
- Function f_r associated to rule r from \mathcal{R}_E of the type

$$(x)_{e_k} \xrightarrow{f_r} (y_1)_{e_{k_1}} \dots (y_h)_{e_{k_h}}$$

have their range included in $[0, 1]$ and they verify:

- For each $e_k \in V$ and $x \in \Sigma$, the sum of the functions associated to rules of the above type is the constant function 1.
- Function $f_{r'}$ associated to rule r' from \mathcal{R}_E of the type $(\Pi_j)_{e_k} \xrightarrow{f_{r'}} (\Pi_j)_{e_{k'}}$ are all constant and equal to 0; that is, one may as well assume that this type of rule is forbidden, or equivalently, P systems residing in an environment cannot “travel” to any other environment.
- For each rule $r \in \mathcal{R}$ of the system Π_k located in e_k , $1 \leq k \leq n$, the computable function $f_{r,k}$ also depends on the environment and its range is contained within $[0, 1]$. Moreover, for each $u, v \in M_f(\Gamma)$, $1 \leq i \leq m$ and $\alpha, \alpha' \in \{0, +, -\}$, the sum of the functions $f_{r,k}$ with $r \equiv u[v]_i^\alpha \rightarrow u'[v']_i^{\alpha'}$, is the constant function 1. We write \mathcal{R}_{Π_k} instead of \mathcal{R} .

Note that at any instant t , $1 \leq t \leq T$, for each object x in environment e_k , if there exist communication rules of type $(x)_{e_k} \xrightarrow{f_r} (y_1)_{e_{k_1}} \dots (y_h)_{e_{k_h}}$, then some of them will be applied. In case several of such rules exist having $(x)_{e_j}$ as their left-hand side, then the rules will be applied according to their corresponding probabilities associated to this particular instant t .

This formalism described is employed to model complex dynamic systems, and more specifically to capture the processes taking place in a real ecosystem. This is the case of some first attempts to model giant panda ecosystems where the aim of the study is the analysis of population dynamics of the species. Such studies involve much more than simply considering the number of giant pandas. For example, researchers need to consider the distribution of individuals per age and age groups, and of course including all the biotic and abiotic factors, along with the processes affecting their life, from feeding, reproduction and mortality to the influence of human actions or the potential appearance of natural disasters.

In what follows, we give example descriptions of some real ecosystems that were modelled using PDP systems or their variants. It is worth noting that at present, there are at least 18 papers in different categories, as listed in Ref. [13].

Example 3.3. Scavenger Birds [4–6]. Consider system $\Pi = (\Gamma, \mu, \{f_{r,1} | r \in R_\Pi\}, M_1, M_2, R, \{c_r\}_{r \in R})$, having the alphabet $\Gamma = \{X, Y, V, Z\}$ where these symbols represent the same *Bearded Vulture* but in different states and the set of rules $R = \{\{r_1 : [Xh] \rightarrow [VYh']\}, \{r_1 : X \xrightarrow{1-p_1-p_2} Y, r_2 : [X] \xrightarrow{p_1} [B^b], r_3 : [X] \xrightarrow{p_2} [\lambda]\}, \{b[] \rightarrow [ba^{m_1}e^{m_2}]\}\}$ where the first type is a set of rules about nomadic species, the second one is a set of mortality ones and the third is a set of density regulation ones. Other rules are the same as rules mentioned. Inputting the size of the initial population of the *Bearded Vulture* from Catalan Pyrenees in Spain into this system Π , along with some parameters related to the biology of the *Bearded Vulture*, the environment and the possible other interventions, and then executing all the corresponding rules imitating various behaviors of this species, the number of the animals is predicted using this PDP system.

It is worth noting that symbol V represents some animal from other ecosystems influencing the survival of this bearded vulture. An auxiliary variable h is used for ensuring the nomadic species. B represents the bones which are the bones of other ecosystems and b is the biomass of bones. More important, different from other species, mortality rules consists of three parts rather than two ones because these individuals can be recycled.

Example 3.4. Pyrenean Chamois [9]. Consider the system $\Pi = (G, \Gamma, \Sigma, R_E, \Pi, \mu, \{f_{r,1} | r \in R_\Pi\}, \{M_{i,v} | i \in [1, 10], v \in [1, 4]\})$, having the alphabet $\Gamma = \{X, Y, Y', Z, V, W\}$ where object Y' presents some animal which is retired from the ecosystem and the set of rules $R = \{\{r_1 : ([d \xrightarrow{p} S]_k)_{e_v}, r_2 : ([d \xrightarrow{1-p} N]_k)_{e_v}\}, \{r_1 : [Y \rightarrow Y']\}, \{r_1 : [V \xrightarrow{p} W], r_2 : [V \xrightarrow{1-p} \lambda]\}\}$ where the first type simulate the presence (S) or absence (N) of disease, the second one simulates the transformation of the animals and the third simulates the animals hunting or hunted. Other rules are the same as these ones mentioned. The design of the system is based on the biological characteristics of the *Pyrenean Chamois* and the living environment of this species. For example, different

from the system of the *Scavenger Birds*, the proposed model is a multienvironment PDP system with active membranes of degree (4,11). For each of 4 areas (also termed environment), there are the similar environments. Each of 10 cells of every environment above provides a set of rules R describing the biological phenomenon of the species and several unknown disturbances such as pesti-virus infections. Additionally, there are the different climatic scenarios that the model envisages. Using the initial populations of the *Pyrenean Chamois* in 4 areas as input, the model can be applied to automatically evaluate the population size of the *Pyrenean Chamois* in 22 years.

It is important to note that if the hardware, *e.g.* GPU, permits, all subsystems of the multienvironment system run synchronously. From the time point of view, time consumed by this system maybe equal to that consumed by a single environment system. Due to its parallel nature, the model is easy to extend to consider other areas or to perform other tasks.

Example 3.5. Zebra Mussel [7, 10]. *Zebra Mussel* is an aquatic bivalve mollusk, originally described in the reservoir of Riba-roja, located in northeastern Spain [61, 66]. The river is longitudinally structured in 17 different areas. Consider the system $\Pi = (G, \Gamma, \Sigma, R_E, \Pi, \mu, \{f_{r,1} | r \in R_{\Pi}\}, \{M_{i,j} | i \in [1, 4], j \in [1, 17]\})$, having the alphabet $\Gamma = \{X, Y, Z, W\}$ where object X is defined as $X_{i,j,c}$ instead of $X_{i,j}$ where variant c represents the time unit and the set of rules $R = \{r_1 : [D] \xrightarrow{p_1} [t_1, t_2, t_3], r_2 : [D] \xrightarrow{p_2} [t_1, t_2, D'], r_3 : [D] \xrightarrow{p_3} [t_1, D'], r_4 : [D] \xrightarrow{1 - \sum_{i=1}^3 p_i} [D']\}$ where symbol D represents the temperature and an auxiliary variable t_i is that when the temperature reaches this value, adult mussels suitable for this temperature initiate reproduction. Significantly, because the breeding temperature of *Zebra Mussel* is different in waters, we remark that there might be several such temperature intervals. Running this system, it is easy to observe that at each step each type of rules are applicable. After being chose, the mussels maybe initiate evolve or move from a water to another one due to the hydraulic regime. Finally, results of the model *population size* can be output after reaching the maximum number of iterations.

Example 3.6. Other ecosystems [77, 62]. There were attempts to apply different types of PDP systems to other ecosystems such as *logic gene network* [77] and *the quorum sensing system in vibrio fischeri* [62]. Although these species were not animals, these works are useful for biologists to understand the interactions among *genes* or *vibrio fischeri* in living organisms. If you are interested in these problems, please read References [77, 62].

3.4.3 Applications of PDP systems for giant panda ecosystems

The discussion about membrane computing models for giant panda ecosystems concerns mainly the prediction of population dynamics of giant pandas using a cell-like membrane system that studies these objects from the same region or a tissue-like membrane system that studies these objects from various regions with different environment conditions. In most of the models the primary steps are the construction of components in models. The model is composed from three parts: (1) membrane structure which decides the types of a model; (2) objects of the models, which based on giant panda individuals (mapping); (3) rules for their behaviors; Before discussing different types of models, we will firstly and briefly describe the components of the models. For the

Table 7. Recent Studies Claiming to Model Giant Panda Ecosystems with Membrane Computing Models

Item	Cell-like P system [33]	Cell-like P system [32]	Cell-like P systems [73, 74]	Tissue-like P system [75]
Period	2017	2017	2018	2019
Authors	HuangZ, ZhangG	HuangZ, ZhangG(thesis)	TianH, ZhangG	TongY, ZhangG (thesis)
Types	Single-environment, where there is only one habitat studied where giant pandas live.	Single-environment, where the studied habitat is the same as the area of [33].	Single-environment, where the studied habitat is the same as the area of [33].	Multi-environment(the number of the studied areas is greater than or equal to 2, and there are various communications between different cells, which indicates that some objects in one region can be transferred to another one.
Membrane structure.	$[[[]_2]_1$. Cell 1 stores the multiplicity value of food, the multiplicity value of each individual, and the variables of other biological data; Cell 2 stores the rules guiding the behavior evolution of individuals, food consumption, or other changes.	$[[[]_2]_1$ (the meanings of each cell are the same as these of [33])	$[[[]_2]_1$ (the meanings of each cell are the same as these of [33])	$[[[[[]_2]_1]_{e_1}, [[[]_2]_1]_{e_2}]_0$ where e_1 and e_2 represent the habitats where giant pandas live, and two labels 1 and 2 are cell number whose functions are the same as the above cells.
Objects	Objects represent giant panda individuals, and the multiplicity of each object represents the number of giant pandas with the same sex and the age.	Objects→giant panda individuals; the multiplicity of each Object→the number of giant pandas individuals	Object→individual; Multiplicity→the number.	Object→individual; Multiplicity→the number.
Environment	Chengdu Research Base of Giant Panda Breeding (GPBB for short).	GPBB	GPBB	Chengdu Research Base of Giant Panda Breeding (GPBB); China Conservation and Research Center for Giant Panda (CCRCGP).
Rules	Reproduction rule (RR), mortality rule (MR), feeding rule (FR) and updating rule (UR).	RR, MR, FR and UR	RR, MR, FR and UR	RR, MR, FR and UR
Rescue behavior	In the PDP system, rescue behaviors of giant pandas can not be studied due to the difficulty for finding giant pandas.	Several pandas from the forest are rescued and put into the study natural reserve. There are two types of rescue schemes: one is for several individuals in danger to protect endangered species and another for individuals in estrus to increase the diversity of the population.	In the PDP system, study rescue behavior of giant pandas. However, there are different constraint conditions about the age and sex of rescue individuals.	In this system, study the rescue behaviors. However, in contrast to other models, this model can be more complex because there were rescue behaviors in different areas.
Wild release behavior	In the PDP system, the wild release behaviors of giant pandas were not studied due to the limits of the techniques.	In the PDP system, no study wild release behavior due to lack of released data about giant pandas.	Because healthy giant pandas have been successfully released into the wild, this behavior can be studied initially in the PDP system.	In the PDP system, there is no consideration of wild release behaviors due to the limits of the techniques.
Results	The model is a basic model only considering several basic behaviours and not considering any external factors. The more the forecast takes, the greater the difference between prediction data and statistical data is.	Females preferred larger and stronger wild pandas as extra-pair mates to increase the diversity of pandas in a local region; In addition, extra-pair offsprings were more heterozygous than internal-pair mates	Wild introduction for trained giant pandas, especially for individuals in estrus, can increase the diversity of gene of giant pandas and modified the population viability, thereby improving the evaluation accuracy for a population.	Modeling for multi-environment giant panda ecosystem is initially explored in 2019. At presents, the author only involves in the communication between the cared regions without due consideration for rescued behaviour or released one.
Contributions	Modeling giant panda with the PDP system for the first time.	Introduce a new modular <i>Rescue</i> into the PDP system (RPDP system).	Propose a new modular <i>wild release</i> on the basic of RPDP system.	Modeling giant panda with a multi-environment PDP system for the first time and all cells are executed in a parallel.

Table 8. Summary of the Quantitative Attributes of Computing Models in modeling giant panda ecosystem.

Item	Differential dynamic model	Age-classified Matrix Model	Vortex Model	Membrane Computing Model
Period	1989-2016	1996-1999	1997-2018	2017-2019
Papers	10	3	9	6
Parameter expression	Function (the results calculated by functions are positive decimals)	Probability (positive decimals)	Probability (positive decimals)	Rules (positive decimals or distribution functions)
Resolutions	Formulas 1-7 (Section 3.1)	Formula 8 (Section 3.2)	Formulas of Reference [39] (Section 3.3)	the form of rules of <i>Definition 3.2</i> (Section 3.4)
Operation process	Set initial population and survival rate of giant pandas at every stage; choose a suitable and efficient differential dynamic model; execute; output the results.	Age classification; calculate population size, birth rates and mortality rates of giant pandas at each age group; construct an age distribution matrix; iterate and output the results.	Input initial population; set some basic parameters such as rates; choose the types of catastrophes encountered; iterate and output the results.	Build the membrane structure; Input the objects; Design the rules; execute rules (from feeding rule to birth rules, mortality rules, external rules (rescue rules, release rules, catastrophe rules <i>etc.</i>) and update the configuration; and then cycle again according to current configuration; halt membrane systems.
Programming Language	The C Programming Language.	The C Programming Language.	The C Programming Language.	P-Lingua and Java where the rules are coded using <i>P-Lingua language</i> [17] and the evolution process is coded using <i>Java</i> .
Software	Matlab software	Matlab software	Vortex software	MeCoSim software
Experiment results	Due to the more limitations of the model, the deviation between real data and prediction data is a little larger.	Due to the same rates at any moment in each age group regardless of the uncertainty of natural evolution, there also exists the deviation between them.	Due to the fact that the parameters of the models can be optimized, the computational results are better than those of other approaches mentioned.	Due to more consider the actual situation (nature conditions), the simulation results can be even better than those of Vortex model, especially when the parameters are optimized.
Extensibility	Several parameters can be added into the model in the form of functions, since their growth can promote or inhibit the giant panda population.	The parameters cannot be added in, since the model only considers two indices: birth rate and mortality rate.	Extra parameters cannot be added in, but the existed parameters can be chose freely. So its extensibility is limited.	Several external ingredients such as bamboo can be added into the model in the form of rules that are represented as regular language. So this model has good extensibility.
Scalability	With the increase of kinds of parameters, the odds of which the model with different functions has feasible solutions can be small.	In this model, its parameters can be added in the form of probability, hence the performances is not affected when the number of parameters increases.	With the increase of parameters users choose, the computational capability of the software decrease linearly.	With the increase of rules, the computation complex of dealing with rules using the software MeCoSim rises approximately proportional.
Contributions	The model makes the research on giant panda ecosystems from qualitative stage to quantitative stage. Moreover, the model can be applied to handle the continuous problems.	The model discards the functions that have the drawbacks related to the traditional ways of modeling.	Put up with a new methodology to handle the more complex discrete problems; Moreover, firstly use software to simulate the population dynamics of giant panda ecosystem.	The model can break through the limitations of traditional model because researchers can model natural disasters by programming rather than choosing. Moreover, such models can simulate their daily behaviors and build the interrelations to clearly observe the dynamic change of giant pandas.
Drawbacks	The differential equation is not differentiable; there is no equilibrium point; the system is unstable.	For each computation, it is necessary for the model to obtain the positive eigenvalues, verify the existence of eigenvalues, and adjust the stability of the age distribution. If any condition is not satisfied, this model cannot be chose to model giant panda ecosystem.	The user can only consider these factors that the developer have designed them ahead. It implies that the researchers cannot make experiments when the studied factors are not designed in this software.	The types of membrane computing models that can be modelled are confined to those whose rules can be represented as regular language.

models published so far, we will concentrate on two types: (a) cell-like membrane systems in Section 3.4.3 (2) and (b) tissue-like membrane systems in Section 3.4.3 (3), and make comparisons between these models.

(1) Data Organization

- *Membranes.* The main functionality of membranes is to perform a topological division of the space allowing membrane systems to compute in a distributed manner. According to Reference [56], a membrane represents a habitat pandas live (each one identified by its number label). In each membrane, an amount of objects and rules are put into this one. Notably, in the model of giant panda ecosystems, objects represent the same giant pandas but in different states, so in general each of membranes cannot be dissolved. It implies that other configurations are the same besides object changes at every moment. According to Reference [90], the functions of each membrane are different, but the cooperation among the membranes ensures the successful evolution of the giant panda system.

- *Configuration.* The representation of the configuration in all cases is done as vector of a series of the evolutionary states of the giant panda. We remark that this vector corresponds to a constantly readjusted system, so it is updated at every moment. As a simple example, in an initial configuration this system takes *the statistical data* as input and computes *population size* and *the total amount of food* after running a series of rules based on an array of scalar parameters. This process is termed as a transformation of configuration. In the case of References [6, 7, 32] it can be argued that corresponding behaviors of many species are different from each other, leading in general to different steps (internalized into the corresponding rule types). Hence, the transformation processes of states representing the configuration to be processed are determined by the designed rules.

- *Evolution Rules.* As it can be seen from Definitions 3.1 and 3.2 in the simple case (without permitting and forbidding) rules can be defined by two integer vectors indicating the multiplicities of corresponding objects in the right-hand side and left-hand side of each rule. After executing an evolution step, all the objects in the left-hand side of the rules are removed from this system while these objects in the right-hand side are produced. An important aspect should be noted: all rules should be applied once selected except for probability rules (chosen probabilistically). In giant panda ecosystems, for four basic rules mentioned, only both reproduction rules and mortality rules are chosen in a probabilistic way. In the whole period, each rule cannot disappear because this causes some phenomenon that giant panda cannot go through some behavior which goes against the law of natural evolution.

- *Simulation Algorithms.* Due to the nature of PDP systems, the algorithms selecting the rules need to meet the following criteria: (1) they should divide all rules into the corresponding modules; (2) they should be efficient enough to select correctly and linearly in each iteration. Criterion (1) implies that each type of rules should be in the same category. Criterion (2) is needed because there are typically a large number of free rules in the corresponding membranes. This means methods based on non-deterministically choosing rules [56] are not realistic for PDP systems, since they cannot consider the single choice of probability rules. To tackle this problem that only one rule is chosen at every turn, algorithms DNDP [50] and DCBA [51] are developed (for References [50,

51, 90] for more details). We remark that the above algorithms do not rewrite objects, but only ensure which rules can be executed in the system.

(2) Cell-like PDP systems (CPDP systems).

Cell-like PDP systems are the common choice for modeling giant panda ecosystems in a nature reserve. As mentioned in this section, CPDP can predict population size of giant pandas by naturally imitating the everyday behaviors of (real) giant pandas and handle uncertainty. Besides CPDP introduced above, several more straightforward examples, i.e., *rescue-based CPDP systems* and *release-based CPDP systems*, are proposed. This is because in such a system, one can clearly define a generative process of how a giant panda is rescued from the wild habitat or how a captive giant panda returns to the wild habitat.

Rescue-based CPDP systems (Zhang and Huang [33, 90]). Consider the system \prod_1 , having a membrane structure $\mu = [\llbracket \rrbracket_2]_1$ where cell 1 is used for storing static objects and several auxiliaries and another one for rules and the set of *rescue* rules $R_1 = \{r_1 : [A] \xrightarrow{p_1} AC^c[], r_2 : [C] \xrightarrow{p_2} [C_i], r_3 : [C_i] \xrightarrow{p_3} [C_{i,j}]\}$ where objects C and c represent rescue individual and the number of individuals respectively, i represents the sex of individuals and j is its age. Note that p_1, p_2 and p_3 represent the probability of executing a rule where its interval is in $[0,1]$, where if $p_1 = 0$, then both p_2 and p_3 are automatically regarded as 0 and if $p_1 \neq 0$, then p_2 and p_3 are not 0. Because the purpose of rescuing giant pandas is to improve the gene diversity of captive giant pandas, the age of these individuals rescued is basically limited to a certain range. It implies that giant pandas in breeding season are given priority to rescue.

Due to the uncertainties and more complex difficulties in actually rescuing individuals, rescue model forecasting is a long-standing core problem in protecting rare giant pandas. Based on this, References [33] and [90] can provide valuable guidance to put rescue rules into current PDP systems. In general, rescue modular can be executed according to the order of rules in a probability way at next moment while $C_{i,j}$ becomes $X_{i,j+1}$. For these four rules, in software MeCoSim, at first, by provoking A, c objects are incorporated into this system at once (r_1), next recognize the sexes of these pandas though c iterations (r_2), and then count age of pandas though c iterations (r_3). If permits, MeCoSim can record the reproduction and mortality information of these individuals, thus providing many better rescued strategies for researchers.

Release-based CPDP systems (Zhang and Tian [74, 90]). Consider the system \prod_2 , having a membrane structure $\mu = [\llbracket \rrbracket_2]_1$ and the set of *release* rules $R_2 = \{r_1 : [G] \xrightarrow{p_1} GD^d[], r_2 : [D] \xrightarrow{p_2} [D_i], r_3 : [D_i] \xrightarrow{p_3} [D_{i,j}], r_4 : [D_{i,j}] \rightarrow D_{i,j}[]\}$, where objects D and d represent release individuals and its number, similar to auxiliary variant A , variant G is also an induction or excitation parameter aiming at provoking release behavior. Likewise, it is necessary to consider the sex and age of giant pandas returning to the wild habitat in order to enable giant pandas to healthily survive for a long time. It is important to note that rule r_4 indicates that a giant panda leaves captive habitat for a wild forest. The order of these rules is as follows: first, provoke release behaviors (r_1), next select some giant pandas with sex i (r_2), then among them select several giant pandas aged j (r_3) again and at last released giant pandas (r_4), end.

In order to reduce the extinction risk of local small population and rejuvenate wild population, it is desirable for the researchers to incorporate release modular into models

enabling theoretical researches, either to closely forecast population dynamic of giant pandas or more importantly to improve the computational results of models. It is also crucial to ensure models' robustness. In this system, the probability of release modular being executed is p_1 at each iteration, that is, if and only if greater than p_1 , then all the rules about release behaviors are applied according to the above steps. In real environment, it indicates that several healthy and adaptive adult animals are returned to the bamboo forest. It is noted that we cannot further study on these giant panda that have been released to the wild because we only study the population dynamic of present giant pandas in the captive area. It also implies that the number of giant pandas can decrease after running release modular. In many other models [5, 6], this situation is regarded as the mortality of individuals. It indicates that the sum of mortality rate and release rate is 1. However, for References [74, 90], release rate is a single probability value independent of mortality. The advantages of this approach is that the interference caused by multi-information fusion is avoided, especially for these probabilities calculated by a fusion function.

(3) Tissue-like PDP systems (TPDP systems).

In the previous sections, we covered how various CPDP systems can be applied in the single-environment giant panda ecosystems, respectively. In this section, we will discuss how TPDP can be applied in the multienvironment giant panda ecosystems in general, using the examples of Reference [75] as an application.

The model of TPDP systems (Zhang and Tong [75]). Consider the systems Π_3 , having the membrane structure $\mu = [[[[]_2]_1]_{e_1}, [[[]_2]_1]_{e_2}]_0$ where symbols e_1 and e_2 represents two different nature reserves-GPBB and CCRCGP where the substructures of TPDP are the same as that of CPDP, and the set of rules $R = \{r_1 : [X_{e_1,i,j}]_{e_1} []_{e_2} \xrightarrow{p} []_{e_1} [Y_{e_2,i,j}]_{e_2}, r_2, r_3\}$ where the first rule represents one-way transfer, *e.g.*, from reverse e_1 to e_2 , rule r_2 is expressed from e_2 to e_1 , and rule r_3 represents two-way transfer. For r_3 , two objects distributed in two different cells can be sent to other membrane at the same or different moment. The functionality of TPDP system is extended by the included set of communication rules.

Parameter uncertainty. In contrast to the previous constantly fixed parameters, parameter uncertainty fabricates a parameter value straightforwardly by using a function rather than by expert experience before a parameter is confirmed. The simplest approach is the distribution function. Reference [52] firstly proposed the modeling theory, dubbed *parameter uncertainty*, for probability prediction. Pickett [58] further modified this model using a *log-normal distribution* to generate σ effectively and efficiently. Since then, it has become popular in the prediction of parameter uncertainty. In 2019, Tong and Zhang [75] applied this approach to dynamically estimate the reproduction rate or mortality rate of giant pandas every loop (a simulation). In this reference, this approach works in three phases: at the first phase we assume that the probabilities (p_i) of several rules such as reproduction rules or mortality ones obeys *beta distribution*, $p_i \sim \text{beta}(a,b)$, where parameters a and b depend on mean and variance of an array of (manual statistical) birth rates (MBR) or mortality rates of pandas (MMR). In the second phase, we assume that temporal variance obeys *normal distribution*, $\sigma_i \sim \text{norm}(\sigma, SV)$ where σ is the sampling variance of MBR or MMR of pandas and SV equals to the difference between sampling variance and estimating variance. In the third phase, we

assume that the average probability also obeys *beta distribution*, but its two parameters depend on the rate of the first step p_i and temporal variance of the second step σ_i . And then the probability of the last phase is given as input. Note that in P-Lingua program i is the replication loop, which indicates that p_i varies at every replication loop. In addition, parameter p_i can be also other types of probability parameters.

TPDP systems for giant panda ecosystems [75]. Input the data for some year such as the number of giant pandas per age, amount of the required food, *etc* of GPBB and CCRCGP into cell 1 of environment e_1 and cell 1 of environment e_2 and put all the rules with probabilities calculated using *parameter uncertainty* approach into cell 2 of region e_1 and cell 2 of region e_2 , constructing an entire initial configuration, the system initiates execution. In the whole process, each object goes through reproduction phase, mortality phase, feeding phase and updating phase. This process is termed as one cycle (a year). Such a process can be easily repeated for imitating them, predicting the population size of giant pandas in these regions in decades (cycles). After the halting condition is reached, the system stops and outputs the number of *female* or *male* giant pandas per age of each environment.

In system II_2 , its each object represents a giant panda and multiplicity of each object represents the number of giant pandas having the same gender and the same age, which implies that the sum of multiplicities of all objects is equal to population size of this species. Each evolved rule represents a daily behavior of giant pandas. In Reference [75], all the objects and all the rules of subsystem e_1 is the same as those of subsystem e_2 except for the multiplicity of each object. It is worth noting that the implementations of these two subsystems are synchronous through theoretical analysis although at present the parallelism cannot be achieved due to the lack of the hardware devices. With the emergence of GPU or other advanced techniques, it is quite promising to execute the parallelism in the fields of modeling ecosystems. After finishing a cycle, multiplicity of each object needs to be updated again. In addition, the amount of food like bamboos is also updated again. In particular, it should be emphasized that the type and the number of rules keep unchanged in the entire process. It is noted that updating stage is responsible for updating the current configuration with the values from the previous stage.

Technically, TPDP systems are executed using software MeCoSim. In the simulator, the data information on giant pandas from two reverses is stored in an input table. Next, the structure and the rules of the TPDP system are programmed using P-LINGUE language. And then, the general function of *parameter uncertainty* is wrote using JAVA language and are added to the executable file. Finally, these files needs to be loaded into the flat of the simulator. Setting steps, cycles and simulations and click on run button, the system outputs the average number of giant pandas of simulations per year in an output flat. The entire process of the implementation is split into several consecutive stages, which take charge of different operations associated to phases of evolution of configurations. It can be seen that the software simulation is an implementation of theoretical configuration model. In addition, it is noted that during *Halting* stage, the system inspects whether the halting conditions is reached, and once reached, stops the system. To confirm this system, the performance of the TPDP system was assessed and compared in Reference [75]. It is noted that the researchers do not con-

sider the rescue modular and release modular in system II_3 due to the initial exploration for multienvironment PDP systems for modeling this species. Through simulation, for GPBB, the error difference is less than 3.6% and for CCPCGC, the error difference is controlled in the range from -7.6% to 1.9%, which verified that the TPDP system has the potential for addressing the multienvironment problem.

In order to more clearly show the membrane computing models used for predicting population dynamics of giant pandas, these methods are summarized and compared from the quantitative and qualitative perspectives in Table 7.

3.5 Summary

Most of what we believe about how population will change in the future is based on projections made by sophisticated computational models. There are currently multiple model representations for modeling giant panda ecosystem, and although most of them generally make similar predictions about the future trends of population dynamic of this species, they differ significantly in many particulars. Because it is not normally clear which models' scenarios are likely to be the most realistic, the question arises of which specific models to choose and why.

As computational models become more complex, it becomes even more important to have suitable models available through which to model and project population dynamics of the giant panda needed to guide ecological conservation. In general, these models can be performed according to the following principles:

- *Differential dynamic models*: for example, *differential dynamic models* are mainly used for dealing with some simple problems regardless of other constraints; *impulsive models* aim to handle some contingent events; *Holling models* aims to solve the problems of survival rates that unchanged during the whole period; *Allee models* aims to tackle these situations that survival rates cannot change continuously as giant panda individuals increase when the population in some area reaches a saturation state.

- *Age-classified matrix models*: eliminate derivations of differential models, thus avoiding their non-differentiable drawbacks; divide an entire population into different age groups according to the classified groups ahead; and then calculate the number of giant panda individuals per group according to the corresponding rates, *e.g.*, breeding rate and mortality rate.

- *Vortex models*: for matrix model, it cannot work on large-scale population well because of the complicated classification task. Hence, Vortex models are developed to handle the scale-limited problems of matrix models. Moreover, this system has its own module function for each different phenomenon. The only disadvantage is that these modules have been programmed ahead, which implies that users can not make several experiments about this catastrophe if it is not designed in the vortex software in advance.

- *Membrane computing models*: a complete set of modeling system is given, avoiding all kinds of the drawbacks that three previous models appeared; the rules can be devised based on the unique semantic principles of the system; in contrast to the above model, this model is relatively flexible, that is, the number of types of rules can depend on the targets the users study. Moreover, this model can simultaneously simulate the evolution of giant pandas in different areas. More importantly, this model can provide the evolutionary information of each individuals, which plays a key role in protecting

giant pandas. At present, this technique is regarded as an importance approach in the field of studying giant panda ecosystem.

In order to more clearly show the differences between different computational models, in Table 8, we list the example models of four types from the previous section and make a comprehensive comparison and analysis for them from structural and performance points of views. In all the cases, we focus on discussing some major sources of differences between models, for example, the starting and ending year of using this model, how their parameters can be evaluated, how each model runs, and how accurate the results of their projections are, and the merits and drawbacks of every model.

4 Concluding Remarks and Future Research Lines

In this paper we present a comprehensive realization of various computational models applied to the giant panda ecosystems. *The differential dynamic model* is a relatively traditional approach, predicting giant panda numbers by using differential equations, but it requires a considerable amount of work to check whether the function is differentiable. *The age-based matrix model* is a complex modeling approach because it needs to consider age groups. By contrast, *Vortex software* is bright, estimating giant panda numbers directly on software. While it provides a fixed-function framework that the users cannot run parameters they provide, this approach effectively tackles the non-differentiable problems encountered by the differential equations. In contrast with *Vortex*, *membrane computing model* is very promising, providing a powerful unique prediction model as it not only imitates the behaviour of each individual, but also provides a flexible structure that the author can estimate any parameter by programming code using P-Lingua language.

In the future, we can expect both more-in-depth studies on membrane computing models and explorations on even more complex applications of models for giant panda ecosystem. In [92], the wild population has been isolated into an amount of small populations. However, there are the inbreeding problems among these small populations due to lack of communications each other, which makes them difficult to fertilise by themselves. Consequently, it deserves to investigate *minimum viable population (MVP)* by using membrane computing models, where MVP function was added to this model, and each rule was available only in the duration of less than this value.

Membrane computing models have attributes of parallelism with a stronger capability of synchronously performing multi-regions than other modeling approaches. Consequently, the prospect of the models for communications between different minimum viable populations is optimistic and promising, and the computing approaches of MVP are well worth further in-depth study. Besides, recent progress on efficient membrane computing model also lays down the foundation for further improving this model's scalability.

5 Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (61672437 and 61972324). We also acknowledge the support of the research project

TIN2017-89842-P (MABICAP), co-financed by Ministerio de Economía, Industria y Competitividad (MINECO) of Spain, through the Agencia Estatal de Investigación (AEI) and by Fondo Europeo de Desarrollo Regional (FEDER) of the European Union.

References

1. Allee, W. C., Park, O., Emerson, A. E. (1949). Principles of animal ecology. *Philadelphia, PA: Saunders Company*.
2. Bernardini, F., Gheorghe, M. Population P Systems. (2004). *Journal of Universal Computation*, 10(5), 509-539.
3. Carter, J., Ackleh, A., Leonard, B., *et al.* (1999). Giant panda (*Ailuropoda melanoleuca*) population dynamics and bamboo (subfamily Bambusoideae) life history: a structured population approach to examining carrying capacity when the prey are semelparous. *Ecological Modelling*, 123(2-3), 207-223.
4. Cardona, M., Colomer, M., Pérez-Jiménez, M.J., *et al.* A P System modeling an ecosystem related to the bearded vulture. *In Proceedings of the Sixth Brainstorming Week on Membrane Computing*, 2008, 51-66.
5. Cardona, M., Colomer, M. A., Pérez-Jiménez, M. J. (2008). Modeling ecosystems using P systems: the bearded vulture, a case study. *In International Workshop on Membrane Computing*, 137-156.
6. Cardona, M., Colomer, M.A., Margalida, A., *et al.* (2009). AP system based model of an ecosystem of some scavenger birds. *In International Workshop on Membrane Computing*, 182-195.
7. Cardona, M., Colomer, M. A., Margalida, A., *et al.* (2011). A computational modeling for real ecosystems based on P systems. *Natural Computing*, 10(1), 39-53.
8. Caron-Lormier, G., Bohan, D.A., Hawes, C., *et al.* (2009). How might we model an ecosystem? *Ecological Modelling*, 220(17), 1935-1949.
9. Colomer, M.A., Lavín, S., Marco, I., *et al.* (2010). Modeling population growth of Pyrenean chamois (*Rupicapra p. pyrenaica*) by using P-systems. *In International Conference on Membrane Computing*, 144-159.
10. Colomer, M., Margalida, A., Valencia-Cabrera, L., *et al.* (2014). Application of a computational model for complex fluvial ecosystems: The population dynamics of zebra mussel *Dreissena polymorpha* as a case study. *Ecological Complexity*, 20, 116-126.
11. Conway, G.R. (1977). Mathematical models in applied ecology. *Nature*, 269(5626), 291-297.
12. Dawes, J.H.P., Souza, M.O. (2013). A derivation of Holling's type I, II and III functional responses in predator-prey systems. *Journal of theoretical biology*, 327, 11-22.
13. Duan, Y., Rong, H., Qi, D., *et al.* (2020). A Review of Membrane Computing Models for Complex Ecosystems and a Case Study on a Complex Giant Panda System. *Complexity*, Article ID: 1312824.
14. Evans, M.R., Norris, K.J. Benton, T.G. (2012). Predictive ecology: systems approaches. *Predictive ecology: systems approaches*, 367, 163-169.
15. Fang, S., Wan, Q., Fujihara, T. (2003). Loss of genetic variation in giant panda due to limited population and habitat fragmentation. *Journal of Applied Animal Research*, 24(2), 137-144.
16. Feng, T., Manen, V., Frank, T., *et al.* (2009). Habitat assessment for giant pandas in the Qinling Mountain Region of China. *The Journal of Wildlife Management*, 73(6): 852-858.
17. García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., *et al.* (2009). An overview of P-Lingua 2.0. *Lecture Notes in Computer Science*, 264-288.
18. Georgescu, P., MoroAnu, G. (2008). Impulsive perturbations of a three-trophic prey-dependent food chain system. *Mathematical and Computer Modelling*, 48(7-8), 975-997.

19. Geary, W.L., Bode, M., Doherty, T.S., *et al.* (2020). A guide to ecosystem models and their environmental applications. *Nature Ecology and Evolution*, 4(11), 1459-1471.
20. Gui, Z.J. (2005). Biological dynamic models and computer simulation. *Science Press, Beijing*.
21. Gui, Z., Song, G., Chen, Y. (2012). Computer numerical simulation of diffusion on the giant panda population dynamics models among small areas. *Procedia Environmental Sciences*, 13, 2085-2090.
22. Gui, Z., Song, G., Chen, Y. (2012). Simulation study on giant panda population dynamics model with due consideration for deforestation. *Procedia Environmental Sciences*, 13, 2091-2097.
23. Gui, Z., Cheng, Y., Song, G. (2012). Periodic solutions and chaos strange attractors of non-linear dynamic system on forest-bamboo giant panda. *Journal of Beijing forestry univeristy*, 34(01): 110-114.
24. Guo, J. (2007). Giant Panda Numbers Are Surging-or Are They? *Science*, 316(5827): 974-975.
25. Guo, J., Chen, Y., Hu, J. (2002). Population viability analysis of giant pandas in the Yele Nature Reserve. *Journal for Nature Conservation*, 10(1), 35-40.
26. Guo, J., Chen, Y., Zhang, H., *et al.* (2002). A mathematical model for the population of giant pandas and bamboo in Yele Nature Reserve of Xiangling Mountains. *Journal for Nature Conservation*, 10(2), 69-74.
27. Guo, J., Hu, J. (1999). The Population Viability Analysis of Giant Panda in Yele Area. *Journal of Nanjing Forestry University*, 23(5), 27-30.
28. Guo, R., Yuan, X. (1996). Leslie Matrix and Its Application-Prediction on the Development of the Population of the Giant Panda in Fupin. *Journal of Southwest Nationalities College: Natural Science Edition*, 22(2): 175-178.
29. Holling, C.S. (1959). Some characteristics of simple types of predation and parasitism. *Canadian entomologist*, 91(7), 385-398.
30. Hu, J. (1990). Research and Progress in Biology of the giant panda. *Sichuan Science and Technology Press*.
31. Huang, Y., Zhang, G., Zou, X. (2001). Demographic Analyses of the Captive Population of Giant Panda (*Ailuropoda melanoleuca*). *Journal of Northeast Forestry University*, 29(2): 109-112.
32. Huang, Z., Zhang, G., Qi, D., *et al.* (2017). Application of Probabilistic Membrane Systems to Model Giant Panda Population Data. *Computer Systems and Applications*, 26(8): 252-256
33. Huang, Z. Application of probabilistic membrane systems to model giant panda population dynamics. 2017. *Southwest Jiaotong University, MA thesis*.
34. Jiang, H., Hu, J. (2010). Population viability analysis for the giant panda in Baoxing county, Sichuan. *Sichuan Journal of Zoology*, 29(2): 161-165.
35. Kleiman, D.G., Seidensticker, J. (1985). The giant pandas of Wolong. *Science*, 228: 875-877.
36. Kirkpatrick, S., Stoll, E. P. (1981). A very fast shift-register sequence random number generator. *Journal of Computational Physics*, 40(2), 517-526.
37. Kong, L., Xu, W., Zhang, L. (2017). Habitat conservation redlines for the giant pandas in China. *Biological Conservation*, 210, 83-88.
38. Kuno, E. (1987). Mathematical models for predator-prey interaction. *Advances in Ecological Research*, 16, 252-261.
39. Lacy, R. C. (1993). VORTEX: a computer simulation model for population viability analysis. *Wildlife research*, 20(1), 45-65.
40. Lakshmikantham, V., Simeonov, P.S. (1989). Theory of impulsive differential equations. *World scientific*.
41. Latour, A. (1986). Polar normal distribution. *Byte*, 11(8), 131-135.

42. Lee, W.J., Kocher, T. D. (1995). Complete sequence of a sea lamprey (*Petromyzon marinus*) mitochondrial genome: early establishment of the vertebrate genome organization. *Genetics*, 139(2), 873-887.
43. Leslie, P.H. (1945). On the use of matrices in certain population mathematics. *Biometrika*, 33(3), 183-212.
44. Levin, S.A. (1992). *Mathematics and Biology: The Interface* (Lawrence Berkeley Laboratory, University of California, Berkeley, CA).
45. Levin, S.A., Grenfell, B., Hastings, A., Perelson, A.S. (1997). Mathematical and computational challenges in population biology and ecosystems science. *Science*, 275(5298), 334-343.
46. Li, B.V., Alibhai, S., Jewell, Z. (2018). Using footprints to identify and sex giant pandas. *Biological Conservation*, 218, 83-90.
47. Li, H., Li, D., Yong, Y. (1997). A preliminary analysis on population viability analysis for giant panda in Foping. *Acta Zoologica Sinica*, (03): 60-68.
48. Liu, B., Liu, W., Teng, Z. (2007). Analysis of a predator-prey model concerning impulsive perturbations. *Dynamics of continuous discrete and impulsive systems series B*, 14(1), 135.
49. Loucks, C.J., Lü, Z., Dinerstein, E., *et al.* (2001). Giant pandas in a changing landscape. *Science's compass*, 294: 1465-1465.
50. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., *et al.* (2010). A new simulation algorithm for multienvironment probabilistic P systems, *IEEE 5th International Conference on Bio-inspired Computing: Theories and Applications*, 59-68.
51. Martínez-del-Amor, M.A., Pérez-Hurtado, I., García-Quismondo, M., *et al.* (2013). DCBA: Simulating Population Dynamics P systems with proportional object distribution. *Lecture Notes in Computer Science*, 7762: 257-276.
52. McGowan, C.P., Runge, M.C., Larson, M.A. (2011). Incorporating parametric uncertainty into population viability analysis models. *Biological Conservation*, 144(5), 1400-1408.
53. O'Brien, S.J., Knight, J.A. (1987). The future of the giant panda, *Nature*, 325(6107): 758-759.
54. Pan, W., Lv, Z., Zhu, X., *et al.*, 2014. A Chance for Lasting Survival: Ecology and behaviour of Wild Giant Pandas. *Smithsonian Institution*.
55. Păun, Gh. (2000). Computing with membranes. *Journal of Computer and System Sciences*, 61(1), 108-143.
56. Păun, Gh. (2002). *Membrane Computing: An Introduction*. Springer, Berlin.
57. Pethybridge, H.R., Choy, C.A., Polovina, J.J., *et al.* (2018). Improving marine ecosystem models with biochemical tracers. *Annual review of marine science*, 10: 199-228.
58. Pickett, E. J., Stockwell, M. P., Clulow, J., *et al.* (2016). Modelling the population viability of a threatened amphibian with a fast life history. *Aquatic Conservation: Marine and Freshwater Ecosystems*, 26(1), 9-19.
59. Ran, J., Zeng, Z., Wang, H. (2006). A survey of the Giant Panda population and habitats in the Daxiangling Mountains. *Journal-Sichuan University Natural Science Edition*, 43(4): 893.
60. Ren, W., Yang, G., Wei, F. (2002). A simulation model for population viability analysis of giant panda in mabian dafengding nature reserve. *Acta Theriologica Sinica*, 22(4), 264-269.
61. Ricciardi, A. (2001). Facilitative interactions among aquatic invaders: is an invasional meltdown occurring in the Great Lakes? *Canadian journal of fisheries and aquatic sciences*, 58, 2513-2525.
62. Romero-Campero, F.J., Pérez-Jiménez, M.J. (2008). A model of the quorum sensing system in *Vibrio fischeri* using P systems. *Artificial Life*, 14(1): 95-109.
63. Schenkman, L. (2010). Hope for wild pandas. *Science*, 553-553.
64. Shi, X., Song, G., Li, Z. (2015). The effect of diffusion on giant pandas that live in complex patchy environments. *Nonlinear Analysis: Modelling and Control*, 20(1), 56-71.

65. Shi, X., Song, G. (2013). A mathematical model with pulse effect for three populations of the giant panda and two kinds of bamboo. *The Scientific World Journal*, <https://doi.org/10.1155/2013/137384>.
66. Simberloff, D., Von-Holle, B. (1999). Positive interactions of nonindigenous species: invasional meltdown? *Biological Invasions*, 1(1), 21-32.
67. Songer, M., Delion, M., Biggs, A., et al. (2012). Modeling impacts of climate change on giant panda habitat. *International Journal of Ecology*, doi: 10.1155/2012/108752.
68. Stephens, P.A., Sutherland, W.J., Freckleton, R.P. (1999). What is the Allee effect? *Oikos*, 185-190.
69. Stephens, P.A., Sutherland, W.J. (1999). Consequences of the Allee effect for behaviour, ecology and conservation. *Trends in ecology & evolution*, 14(10), 401-405.
70. Su, H., Dai, B., Chen, Y., et al. (2008). Dynamic complexities of a predator-prey model with generalized Holling type III functional response and impulsive effects. *Computers & Mathematics with Applications*, 56(7), 1715-1725.
71. Swaisgood, R.R., Wang, D., Wei, F. (2018). Panda downlisted but not out of the woods. *Conservation Letters*, 11(1), e12355.
72. Taylor, A.H., Zisheng, Q. (1993). Bamboo regeneration after flowering in the Wolong giant panda reserve, China. *Biological Conservation*, 63(3), 231-234.
73. Tian H., Zhang G., Rong H., et al. (2018). Population model of giant panda ecosystem based on population dynamics P system. *Journal of Computer Applications*, 5(38): 1488-1493.
74. Tian, H. Modeling of giant panda ecosystem based on population dynamics P systems. 2018. *Southwest Jiaotong University, MA thesis*.
75. Tong, Y. Modeling of giant panda population characteristics based on multi-environment membrane systems. 2019. *Southwest Jiaotong University, MA thesis*.
76. Valencia-Cabrera, L., García-Quismondo, M., Pérez Jiménez, M.J. (2013). Analysing gene networks with PDP systems. Arabidopsis thaliana, a case study. *Proceedings of the Eleventh Brainstorming Week on Membrane Computing*, 257-272.
77. Valencia-Cabrera, L., García-Quismondo, M., Pérez-Jiménez, M.J., et al. (2013). Modeling logic gene networks by means of probabilistic dynamic P systems. *International Journal of Unconventional Computing*, 9(5-6): 445-464.
78. Waltner-Toews, D., Kay, J.J., Neudoerffer, C., et al. (2003). Perspective changes everything: managing ecosystems from the inside out. *Frontiers in Ecology and the Environment*, 1(1), 23-30.
79. Wang, H., Pan, W. (2002). Population Viability Analysis of Giant Panda (*Ailuropoda melanoleuca*) in Qinling Mountains. *Acta Scientiam Naturalium Universitatis Pekinensis*, 38(6), 756-761.
80. Wang, J., Wang, F. (1995). The mathematical Model of Giant Panda and Bamboo. *Journal of East China Normal University*, 2: 8-14.
81. Wang, J., Xue, Y. (2008). The dynamic of predator-prey model with Allee effect in the prey. *Mathematics in Practice and Theory*, 38(10): 136-140.
82. Wang, Y., Li, Y., Zhang, J.. (2018). Review of Climate Change's Impacts on Giant Panda. *Chinese Journal of Wildlife*, 39(3): 709-715.
83. Wei, F., Costanza, R., Dai, Q, et al. (2018). The value of ecosystem services from giant panda reserves. *Current biology*, 28(13), 2174-2180.
84. Wei, F., Feng, Z., Hu, J. (1997). Population viability analysis computer model of giant panda population in Wuyipeng, Wolong Natural Reserve, China. *Bears: Their Biology and Management*, 19-23.
85. Wu, Y., Yuan, X. (1998). The Stability and Numerical Analysis of ecological model of the Giant Panda. *Journal of Biomathematics*, 13(1): 93-96.
86. Yang, Z., Gu, X., Nie, Y., et al. (2018). Reintroduction of the giant panda into the wild: a good start suggests a bright future. *Biological Conservation*, 217, 181-186.

87. Yang, H., Viña, A., Tang, Y., *et al.* (2017). Range-wide evaluation of wildlife habitat change: a demonstration using Giant Pandas. *Biological Conservation*, 213, 203-209.
88. Yuan, C., Hu, J., Zhang, H., *et al.* The mathematical model of the population between giant pandas and bamboos and its application in Shed 5.1, Research and Progress in Biology of the Giant Panda. *Science and Technology press of Sichuan Province*, 1989.
89. Yuan, Z., Sun, J. (2007). Monitoring report of population size of Giant Panda Golden Monkey and Takin in Changqing Nature Reserve. *Journal of Shaanxi Normal University*, 35: 100-103.
90. Zhang, G., Pérez-Jiménez, M.J., Gheorghe, M. Real-life applications with membrane computing, *Springer*, 2017.
91. Zhang, M., Song, G. (2016). The effect of diffusion loss on the time-varying giant Panda population. *International Journal of Biomathematics*, 9(4), 1650062.
92. Zhang, D., Yu, B., Yu, J., *et al.* (2015). Scheme Design and Main Result Analysis of the Fourth National Survey on Giant Pandas. *Forest Resources Management*, 1: 11-16.
93. Zhang, Z., Hu, J., Wu, H., *et al.* (2002). A Analysis on Population Viability for Giant Panda in Tangjiahe. *Acta Ecologica Sin ICA*, 22(7): 991-998.
94. Zhao, H., Denich, M. (2001). An approach on the survivorship of giant panda in wild. *Journal of Forestry Research*, 12(1), 59-62.
95. Zhao, S., Zheng, P., Dong, S., *et al.* (2013). Whole-genome sequencing of giant pandas provides insights into demographic history and local adaptation. *Nature Genetics*, 45(1), 67-71.
96. Zhou, Z., Pan, W. (1997). Analysis of the viability of a giant panda population. *Journal of Applied Ecology*, 34, 363-374.
97. Zhu, L., Hu, Y., Zhang, Z. (2013). Effect of Chinas rapid development on its iconic giant panda. *Chinese Science Bulletin*, 58(18): 2134-2139.
98. Zhu, L., Wu, P., Zhang, H. (2008). Population viability analysis of giant pandas in the Xiaoxiangling Mountains. *Journal of China West Normal University*, 29(2): 112-116.
99. The 4th Survey Report on Giant Panda in Sichuan Province. (2015). *Sichuan Science & Technology Press*.

Minimum Viable Population Estimations for Giant Pandas using Membrane Computing Models

Yingying Duan¹, Haina Rong¹, Gexiang Zhang^{1,2*}, Dunwu Qi³, Luis Valencia-Cabrera⁴, Mario J. Pérez-Jiménez⁴

¹School of Electrical Engineering, Southwest Jiaotong University, Chengdu 611756, China

²School of Control Engineering, Chengdu University of Information Technology, Chengdu 610225, China

³Chengdu Research Base of Giant Panda Breeding, Chengdu 610081, China

⁴Research Group on Natural Computing. Department of Computer Science and Artificial Intelligence, University of Sevilla. Sevilla 41012, Spain;

Abstract. Giant pandas are expected to decline range-wide and are under the risk of extinction as global climate change and habitat loss continue. Estimating when different subpopulations will likely begin to decline has been hardly possible to date because data linking pandas availability to demographic performance are unavailable for most subpopulations and unobtainable a priori for the projected but yet-to-be-observed low population density. Here, we establish the likely nature, timing and order of the future demographic impacts by estimating the thresholds of *minimum viable population* (MVP), before their populations get impacted and decline rapidly. Furthermore, the individual supplement strategy is used to alleviate and prevent the extinction of this species when population size is less than the MVP threshold at some moment. Intersecting these thresholds with projected MVP size, estimated from MC models, our study reveals when demographic impacts will likely occur in different subpopulations. Our model captures demographic trends observed during 2005-2017, showing that in some periods the number is far below the survival impact threshold even if there are captive individual supplements. The study also suggests that, with the occurrence of natural catastrophes, the steep decline in reproduction and survival will endanger the persistence of the species. Moderate protection may prolong the species' survival, but is unlikely to prevent the extinction of some subpopulations in future.

Keywords: Giant Panda; Minimum viable population; Supplement approach; Membrane computing

1 Introduction

Giant pandas occur in 33 subpopulations across six mountains according to the 4th Survey Report [26]. At present, there are 1864 giant pandas inhabiting in the wild forest and 633 saved captive giant pandas living in the *Panda Nature Reserve*. Generally, these captive pandas are more healthy but lower fertile rates than wild pandas. Scientists believe that many of giant pandas are now at threat of extinction solely due to the

* Corresponding author.

destruction of nature [9]. If we do not protect them, we are pushing them off the planet, in effect. We are causing extinction of this species already. And that's irreversible. In 2011, Zhang *et al*, a dean from chengdu reserve, announced that a new era in giant panda conservation would be just around the corner: 6 captive giant pandas would go out in the wild after being trained in *Giant Panda Valley* in order to rejuvenate small population in the forest. That raises a concerned problem before this: what is a sufficient size (*Minimum Viable Population*, MVP) of giant pandas to endure the calamities of various perturbations and do so within its particular biogeographic context? The MVP concept emerged in 1981 from Shaffer's [17] pioneering paper that defined a minimum viable population as 'the smallest isolated population having a 99% chance of remaining extant for 1000 years despite the foreseeable effects of demographic, environmental, and genetics stochasticity, and natural catastrophes'. The criteria for evaluating viability (the time frame and associated extinction risk) were 'tentatively and arbitrarily' chosen by Shaffer, recognizing that risk criteria were within the purview of society as well as science. Operationally, time horizons of 50-100 years and extinction risk of 5% became the most frequently used criteria [8, 12].

Estimating MVP threshold is challenging because data recording the biological information of the species are lacking in most populations. Indeed, even in the best-studied populations, abundance projections currently rely on extremely limited data. MVP sizes of giant pandas, however, can be projected, even in subpopulations where demographic information is absent, by assuming the parameters affecting the survival of the population [14]. There are five possible approaches to assess MVP thresholds: experiments, biogeographic patterns, theoretical models, simulation models and genetic considerations [17]. Today's projection conditions differ substantially from previous ones, thus precluding empirical measurements of how reproduction and survival will change before these changes occur. In addition, genetic determinants of MVP sizes are still unclear. Considering technicality and feasibility, the most promising approaches are the extension and refinement of analyses of theoretical models and simulation models. Theoretical mathematical models may be useful in revealing which population characteristics or processes are likely to important in affecting survival probabilities. But these models either embody unrealistic assumptions or lead to currently unresolved mathematical problems. Because they are not subject to various constraints of analytical models, computer simulations is used for evaluating MVP sizes and for assessing the effects of changes in various parameter values. Research efforts have been focused on developing software prediction such as vortex and membrane computing so that population dynamics of giant pandas can be estimated without knowing a mathematical model used for describing population dynamics.

The early models studied in previous references are *vortex model*, which is to estimate extinction probabilities that incorporate identifiable threats into models. In a vortex model, MVP of a species is regarded as an initial population size that makes the survival probability reach 95% after procedures halt. At present, several practical studies have outlined the estimations of models introduced various threat parameters. A classic example is that of *Triplophysa Venustus* [22]. Since the success of vortex simulations in other species, such models are also considered in modeling giant panda ecosystems. To experiment with vortex models, researchers commonly verified in various panda

habitats [13, 16, 28, 29]. These studies provide the basis for the MVP projections by incorporating either one threat or more than one single threat or the fusion of kinds of disasters in order to verify the impacts of disasters on population extinction of giant pandas. However, such an approach to evaluating relied on the probability function of populations rather than the evolutionary behaviors of individuals, which cannot explain which individuals were born and which died. In addition, the codes of this software cannot be modified by users, which means that authors can only choose these threat factors that have been designed in advance. Within the biological-inspired computing paradigm, a membrane computing model (MC/P systems) can be implemented as an available tool to model giant pandas ecosystems.

The importance of MC models was already pointed out in many early works [5, 27]. Since the advent of the model, it has been a very active field of research. So far, model practices are currently widely employed in various species such as Pyrenean Chamois [2], Zebra Mussel [3, 5] and Scavenger Birds [4, 10]. Due to the highly successful tests across a variety of application objects, the method are attempted to model giant pandas [27]. Since then, the interest of MC models around this topic keeps increasing. See, for example, both Huang *et al* [11] and Tian *et al* [20] dedicated to studying the single-ecoregion PDP systems, and Tong *et al* [21] focused on conquering the multi-ecoregion PDP systems with various communications between bases GPBB and CCRCGP. Previous models are generally applied in captive giant pandas facing less threats due to stronger protections. However, in fact wild giant panda populations continue to knowledge at alarming rates. In response to boost small population in the wild, several trained captive individuals need to be released into the wild. In the last few years, the national giant pandas from GPBB reintroduction program has released three individuals into the Xiaoxiangling Mountains [24]. This is only in an experimental stage. Because giant pandas are an imperiled rare species, it is not feasible for researchers to make many captive giant pandas go out in the wild directly. In general, many trained captive individuals are really reintroduced into the wild according to a series of schemes made by the model after models are firstly used for projecting population viability ability of giant pandas in the wild in the future. To protect captive pandas to be released from losing their life, an effective approach is to release the minimum but self-sustaining population size. The merits of this model are that it can project MVP of this species based on giant panda evolutionary behaviors and that it can build various interferences impacting population dynamics. Because the models have consistently outperformed traditional methods for object state recognition and computational efficiency, a common approach is to leverage membrane computing models.

Operationally, time horizons studied in recent MVP papers were 50-100 years [17]. The findings showed that the expected number to death by starvation, disease, or other disasters, *etc.* for all individuals would be less than the expected threshold, and thus the number of individuals would decline rapidly [14]. More directly, giant pandas can extinct due to the fact that the less quantities cannot be enough to keep survival for a long time. Conservation breeding is becoming an increasingly important tool to guard against extinction, providing organisms for reintroduction to re-establish or supplement wild populations, or for assisted colonization outside of historical ranges as part of disaster change mitigation strategies [15, 18]. Key to the success of supplement (re-

cruitment) programmes is the availability of a large number of reproductive individuals to guarantee population establishment. Thus, a primary goal of conserving *minimum viable population* should be to choose large numbers of reproductive candidates for assurance and supplement. In membrane computing model, there is a specified set of rules used for achieving supplement actions. Note that the design of this behavior in the MC model is a very complex process. On the one hand, the core problem to be considered is time and quantity of supplying giant pandas; on the other hand, the key issue to be studied is the ages and the sexes of recruitment. For the software MeCoSim running MC models, the most difficult problem is supplement time because the corresponding rules are executed under the control of constraints.

The paper aims to project Minimum Viable Population based on Individual Supplements (also termed MVPIS) of captive giant pandas trained at *Giant Panda Valley* to be released into the wild *Xiaoxiangling Mountains* using membrane computing models (also termed population dynamic P systems (PDP systems for short)). The sensitivity of MVP thresholds to differences in the models was explored by adjusting the birth rates and mortality rates of all giant pandas upward or downward by a specified percentage. The main contributions of this article are summarized as follows:

1) A novel type of PDP systems with recruitment is developed by introducing the notion of *recruitment* into PDP systems. More precisely, such *P* systems have a two-layer membrane structure with the capabilities of different information processing and rule controls.

2) *Minimum Viable Population* (MVP) is studied in this article. Instead of previous models, a novel model is developed to assess MVP thresholds. The main advantages of the model is to record the evolution state of each individual, thus providing an age distribution of MVP that guides to supply which individuals.

3) Parasite disease is incorporated into PDP system to make the trained place where giant pandas to be released live in such an environments closer to wild, which makes projecting MVP presumable and enhances its potential for practical applications.

4) A recruitment strategy is introduced into PDP systems to supply individuals for emergency population, thus protecting this population from reducing rapidly. Noted that there are some certain conditions to supplement individuals. Supplement rules are executed only when the number of a population is less than that of MVP.

The rest of this paper is organized as follows. In Section II, we describe data sources of giant pandas. We also outline a few concrete ecoregions where giant pandas lived. In Section III, we construct a population dynamic P system (PDP system) different from the descriptions in other literatures and we give a novel MVP concept in general, designed according to our requirements/purpose. We then detailedly introduce population-level recruitment in order for protecting this species from extinction during the whole period. In Section IV, we highlight some key challenges that need to be addressed in the future and the uncertainties in that direction. Finally, in Section IV, we make a conclusion and list several possible directions in next phase.

2 Biological data on Giant Pandas and Three areas: Habitation, Training area and Release area

In this paper we choose giant pandas from GPBB as research objects. It is necessary to notice that habitation GPBB is only used for offering giant panda individuals to other places, and therefore, here we only introduce the population size in this area regardless of its geographical environment. By the end of 2017, there are 195 giant pandas living in this base where there are 106 female individuals and 89 male individuals, increasing by 4.83% compared with that of 2018 (186, 102, 84). We aim at studying population viability analysis of giant pandas, so we simulate this scene that a plenty of giant pandas from GPBB are put into the field LLNR to observe their viability. Before this, these chose giant pandas should be trained at least two years in the region *Panda Valley* and then be released into the wild.

Dujiangyan Panda Valley [31], aka Chengdu Field Research Center for Giant Panda under Chengdu Research Base of Giant Panda Breeding, is located at mountain foot of Yutang Town of Dujiangyan City. The area enjoys favorable natural and climate condition, full of evergree bamboo and primeval forest with tranquil streams rull all year. At present, the field ecological area that belongs to a part of panda valley is about 1.4 square kilometers, accommodating 30-40 giant pandas and 50-100 companion wild animals such as lesser pandas, *etc.* In the wild-nature training and experimental area with semi original ecology, giant pandas undergo the wild-nature training and research before released into its nature habitat.

Giant pandas are released into Liziping National Nature Reserve (LNNR) in the Xi-aoxiangling Mountains. Giant panda habitat used for taking some giant pandas trained in *Dujiangyan Panda Valley* is located in the southwest corner of the giant panda distribution region. This area contains the most isolated and smallest giant panda population remaining in the wild. Total panda habitat in these mountains spans 119.36 km² but is bisected by the 108th National Road into two patches [30]. At present, three giant pandas from GPBB have been successfully released into the region since 2016 after being trained, which shows that in the actual life the release of giant pandas is a challenging task due to need to consider various complex factor including human interferences and nature disasters, *etc.*, not only rather than survival.

3 Methods

We used a membrane computing model based on probability (also termed PDP system) to estimate: (1) the MVP threshold that the smallest population can survive before extinction by local environment in the wild occurs; and (2) the number of supplementary individuals that a population released is less than minimum viable population. Subsequently, we estimated the MVP threshold and supplements as formulas (1): \prod (see **Definition 1**) and (2): $S(s, f)$ -an individual supplement's distribution in a given year (see 'the computational process of *Supplementary Individuals*').

3.1 Model

We estimated MVP of giant pandas that a population inhabiting in the wild can survive at the probability of 95% in 100 years using an improved *population dynamic P system* (PDP system for short) on the basis of the model of Tian and Zhang [19]:

Definition 1. A population dynamic P system of degree $(3,1)$ with $q \geq 1$, $m \geq 1$, taking T time units, $T \geq 1$, is a tuple

$$\Pi = (G, \Gamma, \Sigma, \mathcal{R}_E, \mu, \mathcal{R}, \{f_{r,1} | r \in R\}, M_1, M_2, M_3) \quad (1)$$

the PDP system (equation (1)) estimates minimum viable population of giant pandas under the disturbance of external factors, where

- (a) The graph of the system is $G = (\phi)$ because this is a single environment system.
- (b) The membrane structure is $\mu = [[]_2 []_3]_1$, where cell 1 is the output cell (the environment), cell 2 is used for training captive pandas from GPBB and cell 3 is used for simulating the evolution of population to be released. It is noted that the function of cell 2 is to supply individuals for cell 3.
- (c) The working alphabet is

$$\Gamma = \{X_{i,j}, S_{i,j}, Y_{i,j}, Z_{i,j}, W_{i,j}, N_{i,j}\} \cup \{F, C, H, B, O\}$$

where variants $N_{i,j}$, $S_{i,j}$, $W_{i,j}$, $X_{i,j}$, $Y_{i,j}$ and $Z_{i,j}$ represent giant pandas in different moments (steps); variables H , B , O represent food giant pandas consume; F is an auxiliary variable as well as C .

(d) In $M_{i,1}$ we specify the initial number of objects present in each regions (encoding the initial population and the initial food)

$$\begin{aligned} - M_1 &= \{ \} \\ - M_2 &= \{X_{i,j}^{q_{i,j}} : 1 \leq i \leq 2, 0 \leq j \leq 32\} \\ - M_3 &= \{S_{i,j}^{s_{i,j}} : 1 \leq i \leq 2, 0 \leq j \leq 32\} \end{aligned}$$

for variants $q_{i,j}$, $s_{i,j}$, H , B and O , please refer to paper [21].

(e) The set \mathcal{R} of evolution rules consists of reproduction rules, mortality rules, feeding rules, update rules, parasite disease rules and individual supplement rules, where the rules of the first four types are the same as reference [20], so here we mainly introduce disease rules and supplements.

$$\begin{aligned} r_1: [F]_k &\longrightarrow F + [H^{g^3}, B^{g^4}, O^{g^5}]_k, \quad 2 \leq k \leq 3; \\ r_2: X_{i,j}[]_2 &\longrightarrow +[Y_{i,j}]_2, \quad 0 \leq j \leq k_{i,5}, 1 \leq i \leq 2; \\ r_3: S_{i,j}[]_3 &\longrightarrow +[Y_{i,j}]_3, \quad 0 \leq j < k_{i,12}, 1 \leq i \leq 2; \\ r_4: X_{2,j}[]_2 &\xrightarrow{g_y} +[Y^y Y_{2,j}]_2, \quad k_{2,12} \leq j \leq k_{2,13}, 1 \leq i \leq 2, 1 \leq y \leq 2; \\ r_5: S_{2,j}[]_3 &\xrightarrow{g_y} +[Y^y, Y_{2,j}]_3, \quad k_{2,12} \leq j \leq k_{2,13}, 1 \leq i \leq 2, 1 \leq y \leq 2; \\ r_6: X_{2,j}[]_2 &\xrightarrow{1-g_1-g_2} +[Y_{2,j}]_2, \quad k_{2,12} \leq j \leq k_{2,13}, 1 \leq i \leq 2; \\ r_7: S_{2,j}[]_3 &\xrightarrow{1-g_1-g_2} +[Y_{2,j}]_3, \quad k_{2,12} \leq j \leq k_{2,13}, 1 \leq i \leq 2; \\ r_8: +[Y]_k &\xrightarrow{0.5} +[Z_{1,0}]_k, \quad 1 \leq i \leq 2, 2 \leq k \leq 3; \\ r_9: +[Y]_k &\xrightarrow{0.5} +[Z_{2,0}]_k, \quad 1 \leq i \leq 2, 2 \leq k \leq 3; \end{aligned}$$

$$\begin{aligned}
r_{10} &: +[Y_{i,j}]_k \xrightarrow{1-mm_{i,j}} +[Z_{i,j}]_k, \quad 0 \leq j < k_{i,5}, 1 \leq i \leq 2, 2 \leq k \leq 3; \\
r_{11} &: +[Y_{i,j}]_k \xrightarrow{mm_{i,j}} +[]_k, \quad 0 \leq j < k_{i,5}, 1 \leq i \leq 2, 2 \leq k \leq 3; \\
r_{12} &: +[Y_{i,k_{i,5}}]_2 \longrightarrow +[]_2, \quad 1 \leq i \leq 2; \\
r_{13} &: +[Y_{i,k_{i,5}}]_3 \longrightarrow +[]_3, \quad 1 \leq i \leq 2; \\
r_{14} &: +[Z_{i,j}]_3 \xrightarrow{1-yy_{i,j+1}} +[N_{i,j}]_3, \quad 0 \leq j < k_{i,5}, 1 \leq i \leq 2; \\
r_{15} &: +[Z_{i,j}]_3 \xrightarrow{yy_{i,j+1}} +[]_3, \quad 0 \leq j < k_{i,5}, 1 \leq i \leq 2; \\
r_{16} &: +[Z_{i,j}^{f_{i-5} \cdot s_{i,j}}]_2 \longrightarrow S_{i,j+1}^{f_{i-5} \cdot s_{i,j}} []_2, \quad 6 \leq j < 20, 1 \leq i \leq 2; \\
r_{17} &: [Z_{i,j}, H^{f_{i,1 \vee 4 \vee 7}}, B^{f_{i,2 \vee 5 \vee 8}}, O^{f_{i,3 \vee 6 \vee 9}}]_2 \longrightarrow -[W_{i,j}]_2, \quad 0 \leq j \leq k_{i,5}, 1 \leq i \leq 2; \\
r_{18} &: +[N_{i,j}, H^{f_{i,1 \vee 4 \vee 7}}, B^{f_{i,2 \vee 5 \vee 8}}, O^{f_{i,3 \vee 6 \vee 9}}]_3 \longrightarrow -[W_{i,j}]_3, \quad 0 \leq j \leq k_{i,5}; \\
r_{19} &: -[H]_k \longrightarrow []_k, \quad 2 \leq k \leq 3; \\
r_{20} &: -[B]_k \longrightarrow []_k, \quad 2 \leq k \leq 3; \\
r_{21} &: -[O]_k \longrightarrow []_k, \quad 2 \leq k \leq 3; \\
r_{22} &: -[W_{i,j}]_2 \longrightarrow X_{i,j+1} []_2, \quad 0 \leq j \leq k_{i,5}, 1 \leq i \leq 2; \\
r_{23} &: -[W_{i,j}]_3 \longrightarrow S_{i,j+1} []_3, \quad 0 \leq j \leq k_{i,5}, 1 \leq i \leq 2; \\
r_{24} &: C -[]_3 \longrightarrow [C]_3; \\
r_{25} &: F -[]_2 \longrightarrow [F]_2;
\end{aligned}$$

Previous versions of PDP systems [20] applied single cell to model giant pandas' population trend using physiological data, mortality rate, breeding rate of adult female giant pandas, and a general binomial formula determining giant pandas' survival or death at next step. Here, we add a cell used for supplying individuals to the original model by replacing the general single cell for avoiding the giant pandas' extinction during 100 years; In addition, we also introduce recruitment rules to the system; and then re-estimating giant pandas' population dynamic by running formula (1) to repeated measures of the final number of initial giant pandas in the 100th year. Note that rule r_{16} can be executed only when the total sum of mutiplicities of all the objects does not exceed a previous estimated MVP threshold.

Our study aims to make trained giant pandas go out in the wild for rejuvenation of a field isolated small population. To fully understand these released individuals' viability, the minimum viable population N_{mvp} needs to be previously estimated in trained environment (see section 'Minimum viability population'), that is, in the system we firstly exclude interference factors from the wild environment such as parasite diseases (remove rule r_{16}) before calculating threshold N_{mvp} . After this, these trained individuals will be sent to the wild environment with parasite diseases threatening giant pandas' viability at higher probability, which means that the population can extinct in the 100th year by adding the disease to previous environment. To protect the population from extinction during the period, it is necessary for researchers to *supply* or *recruit* a certain number of panda individuals in the environment (see Population-level recruitment').

Here we did not model habitat loss or climate changes impacts on giant panda survival due to a lack of data on giant panda death. Nor did we model possible impacts on bamboos due to the high sensitivity of food types to among-population variation in the relative timing of peak feeding. However, we note that giant pandas' litter size, as well as juvenile and (sub)adult growth and survival, are sensitive to these biological parameters, and will probably be impacted more easily under environmentary factors

outside. Recognizing that our approach consider only part of impact factors expected with parasite disease rather than all components, and that our model parameter choices assume optimal strategies that probably underestimate giant panda death and overestimate female giant pandas' breeding in giant panda populations optimistic throughout (see 'Estimate of future impacts and uncertainties').

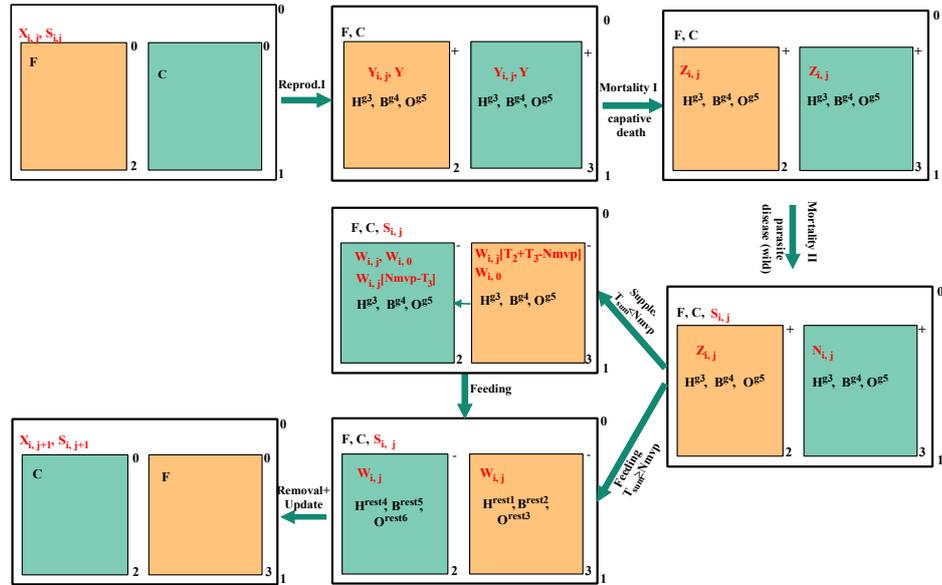


Fig. 1. The framework of PDP system based on individual supplement

Modeling processes of PDP systems. Note that if a system does not contain this situation that the sum of present population is less than the MVP threshold, then the supplementary behavior is directly excluded or skipped from our system.

The \mathbf{P} system $\Gamma(3,1)$ that estimates *minimum viable population* consists of five stages: 1) breeding stage; 2) mortality stages (normal death or parasite disease infection); 3) individual supplement stage; 4) feeding stage; 5) update stage. We remark that during the computational process, $N_{i,j}$ is the states of individuals after infecting parasite disease, and $S_{i,j}$ is the states of individuals after supplying; that is, for each computation step, disease rules are executed but supplement rules are chose only when the number of a population of this species is less than a given threshold.

Breeding Stage: In this stage, by applying breeding rules, some new infants expressed as object $Y_{i,0}$ are produced by female parents. Meanwhile, the system checks the survival states of these individuals by the corresponding truth rules. This stage consists of n iterations, and two steps are consumed for each iteration. So this stage takes $2n$ steps.

At step $l=5i+1$ of i ($1 \leq i \leq n$) iterations, by using rules of r_2-r_7 , for all the male individuals ($i=1$), and these female individuals ($i=2$) not reaching or exceeding breeding

ages, they will be directly sent to cell 3; for rest of female individuals, they will produce new infants in cells 2 and 3 at the probability of g_y where g_1 is the probability of birning single infant and g_2 is the probability of birning twins.

Mortality Stage: In this stage, there are two forms of mortality rules: (a) r_{10} and r_{11} ; (b) r_{14} and r_{15} . The former is used for individual death in the *normal* or *captive* envoriment, and the later is used for parasite disease infection in *wild* environment. The types of rules are executed in different moments where in software MeCoSim the rules of (a) firstly run, and then rules of (b) begin to operate. Note that in rules, $mm(yy)$ is an array with i rows and j columns. Because the mortality rate are the same at some age intervals, in this array there are a number of the same values. This stage begins at step $i=2$, which costs 2 steps.

At step $l=5i+2$ of i ($1 \leq i \leq n$) iterations, rules r_{10} , r_{11} are used in the form of the probabilities. When rule 10 is used, objects $Y_{i,j}$ from cells 2 and 3 disappear, and objects $Z_{i,j}$ appear in cells 2 and 3; When rule 11 is used, objects $Y_{i,j}$ become an empty variable in cells 2 and 3 (that is, individual death). Note that in the procedure if a random decimal greater than $mm_{i,j}$, then execute r_{14} ; else, execute r_{15} .

At step $l=5i+3$ of i ($1 \leq i \leq n$) iterations, when cell 3 is added into *parasite disease*, giant pandas expressed as $Y_{i,j}$ in this region will infect this disease at the probability; meanwhile, some individuals in cell 3 can die executed by using rule r_{15} , which means that the number of the population reduces. For other individauls not infecting the disease, in this moment object $Z_{i,j}$ in cell 3 disappears, and a new object $N_{i,j}$ is produced in cell 3 by using rule r_{14} . Note that the chance giant pandas going out in the wild infect parasite disease is usually greater than that in the capative region due to poor living environment, which shows that compared with other pandas, the mortality rate is more higher here. The effect of rules r_{14} and r_{15} is to ensure that some new uncertain factors can appear in cell 3 in some step.

Individual supplement Stage: The reason of supplying individuals is to protect giant pandas from extinction in complex environment outside. The first step of the system is to start to check whether or not the number of the population α in i th iteration is less than a given MVP threshold β by some practical computations. Specially, if α is greater than β , then rule r_{16} is not executed, that is, this stage is omit; if α is less than β , then rule r_{16} is executed. meanwhile, part of some objects aged 6 and 20 from cell 2 will be sent to cell 3. The stage begins at step $5n+3$.

At step $l=5i+3$ of i ($1 \leq i \leq n$) iterations, the system checks whether the multiplicity of objects $Z_{i,j}$ is less than $\alpha-\beta$. If it assesses to TRUE, then all objects are sent to cell 1; else, $\alpha-\beta$ objects are sent to cell 1. Note that for second case, it is necessary to discuss the types of individuals to be selected as supplementary individuals. For the selection process, please refer to section 3.2. We remark that if the sum of the multiplicity of objects in cell 3 is always greater than minimum population, it means giant pandas cannot be in danger, so this rule is omitted and other rules are normally executed.

Feeding Stage: by using rule r_{17} and r_{18} , object $Z_{i,j}$ ($N_{i,j}$) of cell k ($k=2,3$) transit as objects $W_{i,j}$; meanwhile, the charge of cells changes from positive charge or zero to negative charge. Note that in these two rules we use the form $f_{i,a \vee b \vee c}$ where $a \vee b \vee c$ derived from mathematical notations represents that we can choose one of the three. Here, we make some simplifications on the basis of initial expressions, that is,

$(f_{i,1}, f_{i,2}, f_{i,3})$ is a set of data representing amount of food consumed, $(f_{i,4}, f_{i,5}, f_{i,6})$ is another set of data and $(f_{i,7}, f_{i,8}, f_{i,9})$ is the third set of data (see reference [?]).

At step $l=5i+4$ of i ($1 \leq i \leq n$) iterations, as using rules in mode of feeding, rule r_{17} (respectively, r_{18}) is applied, cells 2 (3) that it contains object $Z_{i,j}$ ($N_{i,j}$) receives a new object $W_{i,j}$ instead of previous objects after consuming an amount of food. Note that here we only consider the situation that there is enough food for the species studied based on the fact that bamboo or other food is enough or even far more to feed giant pandas according to wild survey at least for now.

Update Stage: The system starts to clean the rest of objects unrelated to individuals by applying rules r_{19} - r_{21} . Meanwhile, objects $W_{i,j}$ from cells 2 and 3 are sent to cell 1 by using rules r_{22} and r_{23} . At last, an auxiliary variant F (respectively, C) is sent into cell 2 (3). These rules are executed at step $l=5i+5$ of i ($1 \leq i \leq n$) iterations. Note that if in this iteration supplement rules are executed, then the multiplicity of each object in cell 3 will increase by adding the multiplicity of these objects representing the number of supplementary individuals.

If object $S_{i,j}$ appears in cell 1, then it means giant pandas does not extinct during 100 years. Specifically, at step $l=5i+5$, rules r_{22} and r_{23} are used in at the same moment, and objects $W_{i,j}$ in cell 2, which are evolved to $X_{i,j+1}$ at the same time, are sent to cell 1 and objects $W_{i,j}$ in cell 3, which are evolved to $S_{i,j+1}$ at the same time, are also sent to cell 1. Cell 1 will obtain objects $X_{i,j}$ and $S_{i,j}$ in turn. Note that objects $X_{i,j}$ are only regarded as an auxiliary individuals of cell 3. The system halts and the computation result is object $S_{i,j}$.

If object $S_{i,j}$ does not appear in cell 1 at step $l=5i+5$, then it means giant pandas have already extincted during this period. Specifically, at step $l=5i+5$, only rules r_{22} is used at that moment, and objects $W_{i,j}$ in cell 2, which are evolved to $X_{i,j+1}$, are sent to cell 1. Cell 1 will obtain objects $X_{i,j}$ in turn. Because cell 1 does receive objects $S_{i,j}$, the system does not output any results when the system halts.

3.2 Estimation of an MVP

The MVP concept emerged in 1981 from Shaffer's [17] that defined a minimum viable population as 'the smallest isolated population having a 99% change of remaining extant for 1000 years despite the foreseeable effects of demographic, environment, and genetic stochasticity, and natural catastrophes.' The criteria for evaluating viability (the time frame and associated extinction risk) were 'tentatively and arbitrarily' chosen by Shaffer, recognizing that risk criteria were within the purview of society as well as science. Operationally, time horizons of 50-100 years and extinction risk of 5% became the most frequently used criteria. According to the statistic, the number is relative less for pandas with the rest of species on the earth. Base on this, we use time horizons of 100 years and a 95% chance for remaining extant as conditions to estimate an MVP threshold of giant pandas.

3.2.1 Projection Steps for MVP. In membrane system Π , the population of cell 2 is used for estimating an MVP threshold of giant pandas at common environment without considering rules r_{14} , r_{15} and r_{16} . In cell 2, we suppose that the total number of

objects is $G_{(X,q)}(subpop, year)$ (see the following formula (2)). Within-subpopulation distribution of (X, q) in a given, $G_{(X,q)}(subpop, year)$ records the expected number to survival by evolution for all individuals in that year, and thus $G_{(X,q)}(subpop, year)$ would be regarded as an MVP threshold that the number of pandas is at least 1 when the system halts. However, at present the future population $G_{(X,q)}(subpop, year)$ are unknown, and hence it is necessary to estimate this value by using the designed PDP system.

$$G_{(X,q)}(subpop, year) = \sum_{i=1}^2 \sum_{j=0}^{32} q_{i,j} \quad (2)$$

In formula (2), $q_{i,j}$, the multiplicity of object $X_{i,j}$, varies annually in this region due to occur the breeding or mortality situations there. It means that $G_{(X,q)}(subpop, year)$ varies annually, and cannot yet be anticipated reliably for the future, especially when input data or parameters are uncertain. To overcome these data gaps, we established baseline thresholds only using GPBB sample of 2017 year of 89 males with 49 sexually mature adult males and 106 females with 54 productive adult females (breeding non-selectively during birthing new offsprings). All the females at productive stage have an capability of breeding, but breed in the form of probability (see *rules* r_4 and r_5). Thresholds for other years and subpopulations were estimated using sensitivity analyses that ask how minimum viable population may differ in subpopulations where patterns of giant pandas' age distribution may differ, where the birth rates and mortality rates of giant pandas may differ, or where the number of giant pandas at each age are more or less than those in 2017. Note that $G_{(X,q)}(subpop, year)$ cannot be equal to 0 when the system halts if it is regraded as the MVP threshold of an input data in some year. However, we also need to notice that $q_{i,j}$ may be equal to 0 at some moment, which means that all the individuals in this age group pass away.

We estimated an MVP threshold $G_{(X,q)}(subpop, year)$ (G for short) by cumulating the multiplicity of each object at each iteration (the mean number during the study peirod) using formula (2) of system Π . From $G_{(X,q)}(subpop, year)$, we estimated the expected thresholds to MVP for all solitary males and females using equation (2). For females at breeding period, we estimated upper and lower bounds for each female's offsprings to bracket the uncertainties arising from uncertain reproductive investment strategies. Upper bounds, $q_{i,0}^{UP}$, were estimated by assuming that all mature females maximize their own breeding (birth 2 individuals) with a 100% probability. Lower bound estimate, $q_{i,0}^{LP}$, assumed minimize that behavior that is possible without having no offsprings (and hence accelerating population extinction). For all individuals facing mortality rules, the analysis is the same as the above. We also estimated upper and lower bounds for each individual's mortality. Upper bounds, $q_{i,j}^{UM}$, were estimated by assuming that all individuals end up with their life at the 100% probability (it means the extinction of the population); Lower bounds, $q_{i,j}^{LM}$, were estimated by assuming that each individual goes through its mortality date. The MVP threshold range according to these situations. In what follows, we prove that the MVP threshold also holds.

Theorem 1. *If regardless of the infected parasite rules r_{14} , r_{15} and recruitment rule r_{16} of Π , $r_i \subseteq \mathcal{R}(i \in [14,16])$, then $\exists f \in (q_{i,0}^{LP}, q_{i,0}^{UP})$ and $m \in (q_{i,j}^{LM}, q_{i,j}^{UM})$,*

$G_{(X,(f,m))}(subpop, year) = \min\{G_1, G_2, \dots, G_k\} (G_i \in [G, G + \sigma])$ where k is the number of repetitions and σ is a fluctuation factor.

Proof. Let $G_{(X,q)}(subpop, year)$ be an MVP threshold. The following single-environment PDP system with recruitment Π (see formula (1) and Fig. 1) is constructed to estimate G . Different rule set R of Π , here we design the following finite set \mathcal{R} of rules regardless of rules r_{14} , r_{15} and r_{16} . Same as before, we still regard the objects of cell 2 as the study goal.

1) The following rules in \mathcal{R} are constructed to simulate *breeding* behaviors of Π :

$$\begin{aligned} r_4: X_{2,j} []_2 &\xrightarrow{g_y} +[Y^y Y_{2,j}]_2, \\ r_6: X_{2,j} []_2 &\xrightarrow{1-g_1-g_2} +[Y_{2,j}]_2, \\ r_8: +[Y]_k &\xrightarrow{0.5} +[Z_{i,0}]_k. \end{aligned}$$

By the application of rules r_4 , r_6 and r_8 at each iteration, each female in cell 2 breeds y_j ($y_j=0, 1$ or 2) new infants $Z_{i,0}$ from Y at the rate of $1-g_1-g_2$. It means that the multiplicity of objects plus one. For all the females having an capability of breeding (age j between $k_{2,12}$ and $k_{2,13}$), there are $\sum y_j$ infants. By this way, we calculate the total number of new infants in cell 2 during 100 iterations by using our PDP system.

2) The following rules in \mathcal{R} are constructed to simulate *mortality* behaviors of Π :

$$\begin{aligned} r_{10}: +[Y_{i,j}]_2 &\xrightarrow{1-mm_{i,j}} +[Z_{i,j}]_2, \\ r_{11}: +[Y_{i,j}]_2 &\xrightarrow{mm_{i,j}} +[]_2. \end{aligned}$$

By the application of rules r_{10} and r_{11} at each iteration, each individual expires at the rate of $1-mm_j$. It means that at an iteration the multiplicity of object $Y_{i,j}$ minus one if r_{11} is executed. For all the individuals (age j between 0 and $k_{i,5}$), we suppose that y_j ($y_j=0$ or 1) represents the number of individual deaths at age j per sex, then there are $\sum y_j$ individuals expired. By this way, we know the total number of individual deaths during an entire iteration by executing our PDP system.

For each iteration, the total sum of all the objects equals to the value that previous number plus breeding number and then minus death number. Continue until the system halts, the computation halts only when the system iterates 100 times and the survival rates of pandas reach 95%. The number of initial input object $X_{i,j}$ stored in cell 2 at initial moment represents the computational result of Π , that is, the MVP threshold. Note that it is difficult for us to define effective population size (need to verify by testing the procedures many times. Based on this, the computational result are unstable, hence we add σ describing the fluctuations near the result to variant G). Hence $G_{(X,q)}(subpop, year)$ ranging in $[G - \sigma, G + \sigma]$ is an MVP threshold, and this concludes the proof.

Given the clear importance of *minimum* survival thresholds, it is perhaps necessary to discuss the effects of MVP. For cases where $G_{(X,q)}(subpop, year) \geq N_{mvp}$, the order of giant pandas' survival impacts is preserved, and thresholds estimates vary approximately steadily with differences in birth rates and mortalities. However, the number of giant panda population not exceeding an MVP impact threshold rapidly reduce, and moreover, after the minimum survival threshold are not exceeded, giant pandas' mortality is expected to increase by $z\%$ for each additional year of survival, where z is

the slope of formula II (calculated in the experiment part). Given that population persistence generally requires giant pandas' survival rates of around 95% (more if reproduction is already compromised), even slight exceedance of an MVP threshold would probably push any population into a decline at the very latest. Indeed, if the study population is less than the MVP threshold and there is no any recruitment in this population, population declines are like to occur after the MVP threshold are crossed. All impact thresholds are defined conservatively in the sense that some pandas will experience effects with greater motatilty to trend extinction in the natural wild environment.

3.2.2 Sensitivity analysis. The sensitivity of the MVP thresholds to differences was explored by adjusting birth rates and mortalities of giant panda population. The fluctuations of giant panda population largely depends on birth rates and mortalities, and therefore scenarios ranged from initial birth rates less than a given rate b_0 to birth rates greater than statistic rates b_1 or from initial mortalities less than a given rate m_0 to mortalities greater than statistic rates m_1 (for b_i and m_i , see the experiments in Section 4). This encompasses the approximate range of physiologically possible birth and mortality rates for a viable population. Note that slightly increasing or decreasing these two rates would probably push this population into a rise or a decline at a larger range usually due to the fact that each individual goes through these two stages. It means that the larger fluctuations of thresholds could be expected to occur after this. In order to match the actual situations of birth or mortality rates, ref. [11] sets the probabilities of different age groups instead of the same rates in all the age groups, which indicates that adjusting a rate, birth rate or mortality, of some group only affects the subpopulation of this group rather than all the subpopulations.

3.3 Modeling parasite disease using the PDP system

Generally, among all factors such as climate change (*effects possible*), narature catrastrrophes (*effects likely*) and various diseases (*effects very likely or inevitable*) affecting population dynamics of giant pandas in the field, the diseases are the main and common factors resulting in the death of this species.

Parasitic disease. Parasitic disease is a harmful disease, which is mainly inhabited in small intestines. It is reported that this factor can caused intestinal obstruction and infflmation, thus leading to death. Ye *et al*, taking part in the task of rescuing sick giant pandas in the field between 1974 and 1986, noted that there are various diseases in the surveyed 50 rescued sick giant pandas, where the probability of infection with parasitic diseases is 28%, ranked at the top of all the diseases [25]. Subsequently, Feng *et al* [7] have found that several giant pandas died of this types of diseases. These cases shows that the harm of parasitic disease to giant pandas is a problem worthy of attention.

Modeling parasitic disease in the PDP system. In membrane system, parasitic diseases are modeled accroding to its behaviors. Here, two rules are desgined where a rule is used for simulating the behavior without infectoins such as rules r_{14} , and another is used for simulating the behavior with infections such as r_{15} . These two rules are selected and executed at probabilities and the sum of their probabilities is equal to 1 (the purpose is to ensure that only one rule can be executed every time). Unlike other

nonprobability rules, the implementation of rule r_{15} depends on *probability distribution function* (in MeCoSim, *binomial distribution*). The detailed process is described as: if $yy_{i,j+1}$ is greater than a random number, then this rule can be chose; else, it cannot be chose. For rules r_{14} and r_{15} , they need to be executed $2k_{i,5}$ times in sum. The forms of these two rules are as follows.

$$\begin{aligned} r_{14}: &+[Z_{i,j}]_3 \xrightarrow{1-yy_{i,j}}+[N_{i,j}]_3, & 0 \leq j \leq k_{i,5}, 1 \leq i \leq 2; \\ r_{15}: &+[Z_{i,j}]_3 \xrightarrow{yy_{i,j}}+[]_3, & 0 \leq j < k_{i,5}, 1 \leq i \leq 2; \end{aligned}$$

where variant i represents the sex of a giant panda ($i=1$, male; $i=2$, female), j represents the age of a giant panda, $k_{i,5}$ is the maximum age of a giant panda; both variants Z and N represent the same giant panda but in different states; for $+[]_3$, '+' represent a positive charge aiming at controlling the actions of cells, '[' represents a membrane region (if empty, then giant pandas lose their life) and '3' represents membrane number. For $\xrightarrow{yy_{i,j}}$, $yy_{i,j+1}$ represents the probability of a rule selected, and right arrows ' \rightarrow ' represent the state transformation of pandas from one moment to another

In this system, there are two cells and an enviroment. In both cells, cell 2 is used for predicting population dynamic of the trained giant pandas and cell 3 is for appraising that of wild pandas. Although several giant pandas are trained in giant panda valley, they are still under the supervision and protection of researchers, which shows that there is still little or no chance of infection. On the contrary, giant pandas more easily infect parasitic disease in the wild due to poorer life conditions. Hence, rules r_{14} , r_{15} are introduced into cell 3. Considering the synchronization of rules in membrane system, rules on supplement individuals of cell 2 and rules on parasitic diseases of cell 3 are at the same clock. In addition to this, the order of rule sets of the whole system cannot be changed in order to exactly express the envolutionary feature of the species. Like cell 2, these probability rules having the same left objects are put together but only one rule can be executed per moment. It is important to note that rule r_{15} means the death of giant pandas after infecting parasitic diseases (this region is an empty). Note that normal death belongs to a slow process but disease death will make the animals expire speedily (sometimes giant pandas can lose their life in several days). Seen from biological view, this behavior is accelerating the risk of extinction of this species.

3.4 Population-level recruitment

In general, the number of individuals exceeding an MVP impact threshold would be rapidly reduced or even run the risk of extinction and hence the population-level recruitment is necessary for giant panda persistence. In the system, cells 2 and 3 have the almost same population at initial stage, but their populations live in different environments where cell 3 adds a parasite disease to environment on the basis of cell 2. It means that the population in cell 3 can be extinct. In order to protect giant pandas from extinction during 100 years, recruitment rules as supplying individuals for population of cell 3 are investigated.

A. Recruitment method of PDP system

$$N_{mvp} - T_t |_{T_t < N_{mvp}} = \sum_{j=6}^{j=20} f_{j-5} s_{i,j}, \quad 1 \leq i \leq 2 \quad (3)$$

where we call set $\{f_1, f_2, \dots, f_{15}\}$ Dynamic Allocation Protocol (see Section B); $s_{i,j}$ is the multiplicity of object $S_{i,j}$ of cell 3 (mapped as the number of individuals sexed i and aged j); T_t is the total number of giant pandas in cell 3 at t th iteration; N_{mvp} is an MVP threshold calculated by cell 3.

The supplementary model (equation (1)) estimates the total supplementary quantity per cycle. The variable $f_{j-5} s_{i,j}$ estimates the supplementary quantity of each individual per cycle. It is worth noting that formula (3) can be chosen and executed only if $T_t \leq N_{mvp}$. In addition, it is also worth noticing that in our system we only recruit these female and male individuals aged between 6 and 20 in breeding reason due to the higher survival rates and the higher birth rates in the wild compared with other individuals beyond the range. It is necessary to consider that for cell 3, the population of the cell can be still extinct at lower rates although there are some individuals supplied if the number of individuals is less than $N_{mvp} - T_t$. For formula (3), the variable $N_{mvp} - T_t$ is the minimum recruitment quantity. The purpose to use the minimum is to study the lowest bound condition. Obviously, the more individuals are replenished, the less the population can be extinct.

B. Dynamic allocation protocol

According to the designed idea of the PDP system based MVP, we give the definition of *Dynamic Allocation Protocol* (DYAP for short).

Definition 2. Let $H=(r_1, r_2, s_{i,j}, S_{i,j}, f_{i,j})$ be a distribution strategy where both r_1 and r_2 vary dynamically. According to formula (4), parameter $f_{i,j}$ is determined at some moment; According to formula (3), the multiplicity of object $S_{i,j}$ to be resleased into the wild field is determined. The multiplicity of $S_{i,j}$ varies with $f_{i,j}$ after $f_{i,j}$ varies with r_1 and r_2 . We call the method *Dynamic Allocation Protocol*

The process of DYAP is described as follows:

Step 1: Arbitrarily generate 15 random numbers less than 1, making their sum equal to 1. It means that there must be several numbers equal to zero among 15 numbers.

Step 2: These numbers are put into an array randomly in order to ensure each individual of cell 2 have an equal chance to be chosen and recruited.

Step 3: Each element of set $\{f_1, f_2, \dots, f_{15}\}$ is assigned a value of an array in order. Obviously, among this set, some elements f_j are equal to 0, that is, $f_j \cdot s_{i,j} = 0$, which means that individual $S_{i,j}$ is not chosen for recruitment.

Step 4: According to formula (4), these individuals of cell 2 to be sent to cell 3 are obtained. This is only one cycle. In the PDP system, there is a set H usually different from previous one per cycle.

According to these descriptions, it can be seen that the difficulty is how to determine which individuals are chosen as supplements. Here we give a formula used for calculating f_j as follows.

$$f_j = |(r_1 - r_2)/2|, \quad j = 1, \dots, 15 \quad (4)$$

where r_1 and r_2 are two random numbers, and $\sum_{j=1}^{j=15} f_j=1$.

In equation (4), random variable r_1 is replaced with $|(r_1-r_2)/2|$ in order to make f_j become smaller as soon as possible. According to $\sum_{j=1}^{j=15} f_j=1$, there are more variables like f_j not equalling to 0 if f_j is smaller, which means that there are more types of individuals chosen as supplements, thus further enhancing the survival rates of giant pandas in the wild. If $f_j \neq 0$, $f_j s_{i,j}$ individuals in cell 2 can be chosen and sent to cell 3 at the same moment. The population of cell 3 can protect from extinction by recruiting some stronger and more fertile individuals of cell 2 to increase birth rates and rejuvenate population.

C. Recruitment rules of PDP system

In this section, several fundamental rules from PDP systems are recalled (see system II of formula (1)). Also, several notions of recruitment rules are introduced.

$$r_{16} : +[Z_{i,j}^{f_j-5 \cdot s_{i,j}}]_2 \rightarrow S_{i,j}^{f_j-5 \cdot s_{i,j}} []_2, \quad 6 \leq j \leq 20, 1 \leq i \leq 2; \quad (5)$$

In the membrane system, it is noted that the rule is executed only when the number of giant pandas from cell 3 less than the threshold of MVP, that is, if $T_{sum} < N_{mvp}$, then execute r_{16} . Different from traditional programmed software, these rules written by file *pli* are executed using software MeCoSim. Hence, it is not easy to add the constraint to file *plis*. The executing process of the rule using MeCoSim is as follows. It is noted that unlike other probability rules less than 1, the rule must be directly run only if satisfying constraint conditions. In addition, in this software the rules are executed 30 times per cycle rather than just once.

The recruitment rule (equation (5)) estimates the number of supplementary individuals $f_j \cdot s_{i,j}$ for cell 3 due to population persistence. In what follows, the rule will process all the objects $Z_{i,j}$ which have $q_{i,j}$ individuals. After the results of formula (3) are calculated, the new multiplicity of objects $Z_{i,j}$ to be chosen are obtained. Next, objects $Z_{i,j}$ from cell 2 are sent to the outside of cell 2, that is, its parent membrane-cell 1 (where the rest of objects $Z_{i,j}$ are still stored in cell 2). Because all individuals from cells 2 and 3 are sent to cell 1 at last moment, supplementary individuals from cell 2 are directly sent to cell 1 after evolving into another 15 variants $s_{i,j}$, and then the multiplicity of each individual will be recalculated after cell 1 accepts these individuals expressed as $s_{i,j}$ from cell 3. According to the above evolving process, these supplementary individuals will be added to cell 3 at next moment, end.

Further, the charge of cell 2 changes when the number of individuals changes inside and outside according to biological principles. For rule r_{16} , that parts of individuals from cell 2 are moved away makes the charge of the cell change from positive charge to zero. It is important to explain that in the membrane there are another four steps such as feeding rules, breeding rules, mortality rules (phases I and II), *etc.* In these states, the charge of a cell changes when the number of individuals changes, for example, breeding phase and mortality phase, and at other moments the charge retains unchange.

3.5 Summary

This section provides a brief summaries of characteristics, advantages, uncertainties and unconsidered situations of this system when predicting MVP of giant pandas.

3.5.1 Conclusion for PDP systems

Here we summary the function of each cell in the system with disease rules and recruitment rules.

Example 1: For cell 2 of the PDP system, let $\mathcal{R}_2 = \{\{r_2, r_4, r_6, r_8\}, \{r_{10}, r_{11}, r_{12}\}, \{r_{17}, r_{18}\}, \{r_{19}, r_{20}, r_{21}, r_{22}, r_{25}\}\}$ ($\mathcal{R}_2 \subseteq \mathcal{R}$). For other variants, please refer to section 3. Here we only consider the objects and rules of cell 2 regardless of those of cell 3. Note that all the rules of cells are allowed but some of these rules are allowed with a given probability. At each step, each rule are allowed, then no rule can be used in the system, and the computation halts. After receiving objects from cell 2, the result of computation of cell 2 is the final result.

Example 2: For cell 3 of the PDP system, let $\mathcal{R}_3 = \{\{r_3, r_5, r_7, r_9\}, \{r_{10}, r_{11}, r_{13}\}, \{r_{14}, r_{15}\}, \{r_{16}\}, \{r_{18}\}, \{r_{19}, r_{20}, r_{21}, r_{23}, r_{24}\}\}$ ($\mathcal{R}_3 \subseteq \mathcal{R}$). Now, we analyze how cell 3 in the system works: in such system, all the rules can be chosen and used but rule r_{16} . Note that rule r_{16} can be chosen and used only if the nunmber of giant panda individuals is less than an estimated MVP threshold. Obviously, rules of \mathcal{R}_3 excluded r_{16} are used in every step, and the computation halts when reaching the maximum iteration. So the result is obtained.

TABLE 1. Comparisons between CELL II and CELL III with two additional behaviors (see bold font). Note that although these two cells are in the same system, they have different tasks in the process of running. CELL II is chosen and simulated the population evolution in the general field, and CELL III is chosen and simulated the evolution behaviors in the poorer wild field. It means that each cell has various behaviors under different conditions, and therefore, here we futher summary the differences of two cells on the basis of two examples mentioned.

System type	Purpose	Relationships	Types of using rules	Steps
CELL II of PDP system	1) Estimate the <i>MVP</i> threshold; 2) Supply individuals for CELL III	CELL II can be considered as a fundamental membrane	Breeding rules r_2, r_4, r_6, r_8 ; mortality rules r_{10}, r_{11}, r_{12} ; feeding rules r_{17}, r_{18} ;	5
CELL III of PDP system	Recruite individuals to protect released giant panda population from extinction	CELL III can be regarded as an extension of CELL II (CELL III=CELL II \cup {Diseases, Recruitment})	Breeding rules r_3, r_5, r_7, r_9 ; mortality rules r_{10}, r_{11}, r_{13} ; disease rules r_{14}, r_{15} ; recruitment rules r_{16} ; feeding rules r_{18} ;	5

3.5.2 Advantages for PDP systems

By analyzing the effects of the PDP system in evaluating MVP threshold, it is necessary to remark their main advantages in: 1) accuracies; 2) efficiencies; 3) flexibility. Here, we make several comparisons with other models published in the references. The

first type of compared model is mathematical models [23], and the second type of compared models is vortex models [13, 16, 28, 29]. The detailed introduction is as follows.

- *Accuracies.* The antithesis between mathematical models and PDP systems is clear. The early modeling technique aims to predict MVP of giant pandas by building a relationship (function) between two populations at adjacent moments. The fact shows that such functions might be overfitting. PDP system has the advantage of being the simplest to implement and is suitable for getting rid of fitting phenomena occurred in functions because it predicts MVP by using discrete rules instead of continuous functions. In addition to this, the method records the evolutionary actions of each individual, thus reliably estimating MVP of giant pandas. In fact, MVP is regarded as the smallest isolated population after going through all the catastrophes occurred in the studied area. However, Vortex models only simulate part of catastrophes at local place due to limitations of techniques, which can lead to predict an optimistic MVP that doesn't conform to the reality. For PDP systems, it models various catastrophes, thus predicting an effective MVP. Hence the accuracy of MVP of this model is superior to that of Vortex model. In conclusion, PDP systems have obvious advantages in prediction results.

- *Efficiencies.* The original motivation of membrane computing is to imitate the distributed and parallel capability of biological cells. The computational efficiency of the PDP system is almost the same as that of vortex model in the general computer, but the computational speed of PDP systems may be faster than that of vortex models when several independent cells of the PDP system are executed parallelly in GPU. However, this is just a theoretical analysis, not really implementing in GPU. Note that here we do not consider the efficiencies of mathematical models due to the reason that the population size per moment can be calculated directly by formulas without iterations, thus ignoring its computational time. To a certain extent, the efficiencies of these models might be approximately equal to or superior to compared models.

- *Flexibility.* In comparison with the models [23] that are built only based on population sizes without considering external factors that threaten population survivals, the PDP system can add some possible threats from the field into these models to really and reliably predict population dynamics of this species in the future. In comparison with vortex models, although two types of models accept some threats occurred in the field, the differences are that vortex models only simulate these factors that have designed in the software in advance, and PDP systems model various threats as a result of rules that are planned by implementers themselves, which means that PDP systems are extensible when designing evolutionary rules of giant pandas. Note that other threat modules can be also added to software by developers rather than users. Briefly, the advantage of PDP system lies in the fact that it can model any external random threats from the field without modifying its framework. ■

3.5.3 Estimates of Uncertainties and Analysis for model

Uncertainties are a kind of random events that cannot be determined easily. More importantly, it can result in several unsteadily simulated experimental results. For the PDP system predicting MVP, two types of uncertainties stem from unknown external factors and probabilities that several events occur. In terms of uncertainties derived

from probabilities, they are dealt with by considering the complete range of biologically feasible impact thresholds (increase or decrease in probabilities) to define risk for when population will likely begin declining under our baseline assumptions. Uncertainties arising from uncertainty in PDP model parameters (factors) are dealt with by evaluating how populations of giant pandas will shift if our predicted MVP is crossed. In fact, this projections of MVP thresholds might prove too pessimistic or optimistic, where an MVP of inhabiting in an increasingly unknown external factors can underestimate the viability of populations, and an MVP of living in a steadily comfortable environment can overestimate its viability. At present the only suitable way to deal with uncertainties is sensitivity analysis. The sensitivity of MVP thresholds was explored by adjusting the birth rates and mortality rates of giant pandas upward or downward by a specified percentage (for scenarios ranges, please refer to Section 4.2).

In summary, and as outlined above, our projections are probably conservative and predict threshold exceedance later than is likely in reality, because all model parameters and probabilities are chosen to yield optimistic estimates in cases where data scarcity necessitated a choice: we may underestimated mortality rates or overestimated birth rates in some cases, and particularly for females; we do not consider other demographic effects that will probably occur in concert with those outlined here, such as bamboo flowering, earthquake and other diseases threatening the life of giant pandas, *etc.*; and we do not consider the inbreeding of giant pandas. Moreover, we define impact MVP thresholds conservatively by only considering the 95% survival probability for all calculations; Moreover, we also note that the PDP simulations of MVP thresholds slightly overestimated population viability of giant pandas, and that age groups of this species are also given conservatively due to the unclear recognition about pandas' life stages. The parameters of these points would have effects on population dynamics of giant pandas.

4 Conclusion

The paper aims to theoretically study the minimum survival capabilities of captive giant pandas from GPBB to be released into the wild after being trained using population dynamic P systems (PDP systems). Generally, there are two problems to be considered carefully: 1) MVP of giant pandas and 2) individual supplements. In PDP systems, each cell have a special functions to solve these problems. In this paper, for example, cell 2 is regarded as the training base and cell 3 is considered as the field that giant pandas go out after being trained. Due to the limitations of techniques, there are no plenty of giant pandas to be really released, and therefore, we estimate the MVP of this species in cell 2 instead of cell 3. In comparison of the survival conditions of cell 2, that of cell 3 is even worse due to add parasite diseases into this region, which means that giant panda populations might have higher extinction probabilities. Hence, it is necessary to supply new individuals for cell 3 when the MVP threshold is crossed at some moment. For recruitment, the most important points are to analyze how to supply. In order to rapidly rejuvenate an endangered population, we only provide these healthy breeding individuals aged between 6 and 20. Note that in our system supplement rules can be executed only when the number of current population is less than the MVP size of giant pandas.

In addition to this, we use random functions to determine the number of individuals to be recruited. Also intently, the number of giant pandas can rapidly decline or even run the risk of extinction if MVP threshold is crossed and there are no available individuals in cell 2 to supply for cell 3. Another important analysis is sensitivity analysis. The future study direction to further deeply explore is a multi-environment PDP system and its model verification.

Acknowledgments. This work was partially supported by the National Natural Science Foundation of China (61672437 and 61972324). We also acknowledge the support of the research project TIN2017-89842-P (MABICAP), co-financed by Ministerio de Economía, Industria y Competitividad (MINECO) of Spain, through the Agencia Estatal de Investigación (AEI) and by Fondo Europeo de Desarrollo Regional (FEDER) of the European Union.

References

1. Cardona, M., Colomer, M. A., Pérez-Jiménez, M. J., *et al.* (2008). Modeling ecosystems using p systems: The bearded vulture, a case study. *In International Workshop on Membrane Computing*, 137-156.
2. Cardona, M., Colomer, M., Pérez-Jiménez, M.J., *et al.* A P System modeling an ecosystem related to the bearded vulture. *In Proceedings of the Sixth Brainstorming Week on Membrane Computing*, 2008, 51-66.
3. Cardona, M., Colomer, M. A., Margalida, A., *et al.* (2011). A computational modeling for real ecosystems based on P systems. *Natural Computing*, 10(1), 39-53.
4. Cardona, M., Colomer, M.A., Margalida, A., *et al.* (2009). AP system based model of an ecosystem of some scavenger birds. *In International Workshop on Membrane Computing*, 182-195.
5. Colomer, M., Margalida, A., Valencia-Cabrera, L., *et al.* (2014). Application of a computational model for complex fluvial ecosystems: The population dynamics of zebra mussel *Dreissena polymorpha* as a case study. *Ecological Complexity*, 20, 116-126.
6. Duan, Y., Rong, H., Qi, D., Valencia-Cabrera, L., Zhang, G., & Pérez-Jiménez, M. J. (2020). A Review of Membrane Computing Models for Complex Ecosystems and a Case Study on a Complex Giant Panda System. *Complexity*, 2020.
7. Feng, W., Ye, Z., He, G. (1986). Save giant pandas. *Chinese Journal of Wildlife*, (03):14-17.
8. Flather, C. H., Hayward, G. D., Beissinger, S. R. (2011). Minimum viable populations: is there a magic number for conservation practitioners? *Trends in ecology & evolution*, 26(6), 307-316.
9. Guo, J. (2007). Wildlife conservation-Giant panda numbers are surging-or are they? *Science*, 316(5827), 974-975.
10. Colomer, M.Á., Margalida, A., Sanuy, D., Pérez-Jiménez, M. J. (2011). A bio-inspired computing model as a new tool for modeling ecosystems: the avian scavengers as a case study. *Ecological Modelling*, 222(1): 3347.
11. Huang, Z., Zhang, G., Qi, D., Rong, H. (2017). Application of Probabilistic Membrane Systems to Model Giant Panda Population Data. *Computer Systems and Applications*, 26(8): 252-256.
12. Jamieson, I. G., Allendorf, F. W. (2012). How does the 50/500 rule apply to MVPs?. *Trends in ecology & evolution*, 27(10), 578-584.

13. Jiang, H., Hu, J. (2010). Population viability analysis for the giant panda in Baoxing County, Sichuan. *Sichuan Journal of Zoology*, 29(2): 160-165.
14. Molnár, P. K., Bitz, C. M., Holland, M. M., Kay, J. E., Penk, S. R., & Amstrup, S. C. (2020). Fasting season length sets temporal limits for global polar bear persistence. *Nature Climate Change*, 10(8), 732-738.
15. Martin-Wintle, M. S., Shepherdson, D., Zhang, G. (2015). Free mate choice enhances conservation breeding in the endangered giant panda. *Nature communications*, 6(1), 1-7.
16. Ren, W., Yang, G., Wei, F. (2002). A simulation model for population viability analysis of giant panda in Mabian Dafengding nature reserve. *Acta Theriologica Sinica*, 22(4): 264-269.
17. Shaffer, M. L. (1981). Minimum population sizes for species conservation. *BioScience*, 31(2), 131-134.
18. Seddon, P. J., Griffiths, C. J., Soorae, P. S. & Armstrong, D. P. (2014). Reversing defaunation: restoring species in a changing world. *Science*, 345, 406412.
19. Tian, H. Modeling of Giant Panda Ecosystem based on Population dynamic P system. 2018. *Southwest Jiaotong University, Master thesis*.
20. Tian H., Zhang G., Rong H., Mario J. Pérez-Jiménez. (2018). Population model of giant panda ecosystem based on population dynamics P system. *Journal of Computer Applications*, 0-0.
21. Tong, Y. Modeling of giant panda population characteristics based on multi-environment membrane systems. 2019. *Southwest Jiaotong University, Master thesis*.
22. X. Wu, G. Li, B. Bi. (2017). A study on population viability analysis and estimation of minimum viable population in triplophysa venustus. *Acta Hydrobiologica Sinica*, 41(3), 543-551.
23. Xu, H.F., Lu, J.H. (1996). Minimum viable population-a basic theory of conservation biology. *Chinese Journal of Ecology*, 15(2): 25-30.
24. Yang, Z., Gu, X., Nie, Y. (2018). Reintroduction of the giant panda into the wild: A good start suggests a bright future. *Biological Conservation*, 217, 181-186.
25. Ye, Z. (1989). The diseases and prevention of 50 wild giant pandas. *Chinese Journal of Veterinary Medicine*, (02):30-31.
26. Zhang, D., Yu, B., Yu, J., et al. (2015). Scheme Design and Main Result Analysis of the Fourth National Survey on Giant Pandas. *Forest Resources Management*, 1: 11-16.
27. Zhang, G., Pérez-Jiménez, Mario J., Marian Gheorghe. (2017). Real-life applications with membrane computing, *Springer*.
28. Zhang, J., Hu, J., Wu, H. (2002). A Analysis on Population Viability for Giant Pandain Tangjiahe. *Acta Ecologica Sinica*, 22(7): 990-998.
29. Zhu, L., Wen, W., Zhang, H., Hu, J. (2008). Population viability analysis of gaint pandas in the Xiaoxiangling Mountains. *Journal of China West Normal University*, 29(2): 112-116.
30. Zhu, L.F., Zhang, S.N., Gu, X.D., Wei, F.W., (2011). Significant genetic boundaries and spatial dynamics of giant pandas occupying fragmented habitat across Southwest China. *Molecular Ecology*, 20, 11221132.
31. <http://www.panda.org.cn/china/news/news/2013-01-19/198.html>

A Web-based Visual Simulator for Spiking Neural P Systems

Annysia Glynis S. Dupaya, Anica Clarice Antonella P. Galano, Francis George C. Cabarle*, Ren Tristan De La Cruz, Ivan Cedric H. Macababayao, Korsie J. Ballesteros, Prometheus Peter L. Lazo

Algorithms and Complexity, Dept. of Computer Science, University of the Philippines, Diliman, Philippines

Abstract. Under the active area of natural computing, Spiking Neural P systems (SN P systems) are computational models that take from the idea of neural communication by sending spikes through the synapses connecting the neurons. As more research is done in this area, several studies are focused on creating simulators to aid in the creation, experimentation and understanding of SN P systems. Most simulators are ran through text-based simulations with no visualization of the actual system. A web-based simulator was created to bridge the need for a tool that provides a visualization of SN P systems for building and running computations. While limitations were found in the amount of memory used when running an SN P system, results and testing show promise in the use of web-based technologies.

Keywords: Membrane Computing · Spiking Neural P Systems · Visual Simulator

1 Introduction

Membrane computing is an area of natural computing that studies how computing ideas/models/paradigms in cells become systematic in nature and how these can be applied to modern computing [30]. Spiking neural P systems (SN P systems) in particular are inspired by how neurons typically communicate with each other through the sending of spikes between the synapses that connect one neuron to another [20]. The field of SN P systems, and membrane computing in general, is an ever-evolving area of computer science that requires more research as it changes the way people perceive how computations are traditionally made [26]. As part of the latest evolution of neural networks [16], much research is still being explored in what problems SN P systems can solve.

Figure 1.1 shows a simple SN P system, system Π_{3k3} , that sends spikes to its environment after $3k + 3$ time steps [20]. To have a better grasp of what a system does, one would need to use a table or describe what the system does at a time step.

* corresponding author: fccabarle@up.edu.ph

Another way to further understand and allow for the experimentation of more SN P systems, simulations have been made with various levels of abstraction [35]. The purpose for creating these simulations can vary, such as testing an SN P system using particular hardware, checking the validity of an algorithm made for an SN P system [15], or simply learning how an SN P system functions. While tables and descriptions do prove that the system does spike after $3k + 3$ time steps, the method of visualization is quite static and hard to understand at first glance. If the visualization was more dynamic with a display of spike changes and animations showing when a neuron spikes, the system in Figure 1.1 could be even more understandable, as shown in Figure 1.2 for reference.

Graphic User Interface (GUI) types of simulators are visual in nature and is usually intended for users who are not familiar with the specific details on how a system runs and who just want to see the output. It should be noted that the main challenge with GUI simulators in particular is to represent the interface and the output of its simulator in such a way that it becomes easily understandable even to people without such an in depth background in either SN P systems or membrane computing in general.

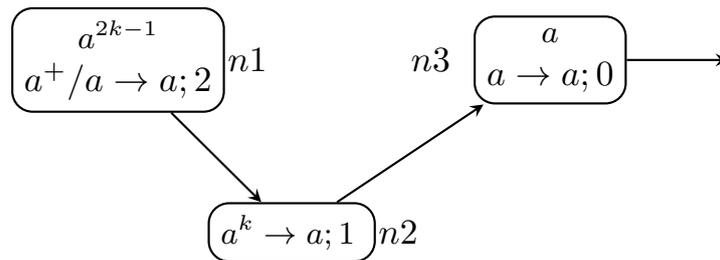


Fig. 1.1. The SN P system Π_{3k3}

WebSnapse is a web-based visual simulator for Spiking Neural P systems. The goal of this software is to increase accessibility to the study of SN P systems through the creation of a web-based simulator with a graphical user interface. It also aims to expand upon a previous SN P system visual simulator, Snapse [10].

Section 2 will be going over the key concepts involved in the creation of WebSnapse. These include the fundamentals of spiking neural P systems, the definition of web-based simulators, and the benefits of using these simulators. This will be followed by Section 3, which looks at the main visualization tool used as reference, Snapse, and compares it to other related tools. Section 4 goes over the basic functionalities of WebSnapse, the process for its development, and its comparison with Snapse and other simulators. Section 5 will elaborate on the capabilities of the software through case studies. This will then be summarized with conclusions and recommendations in Section 6.

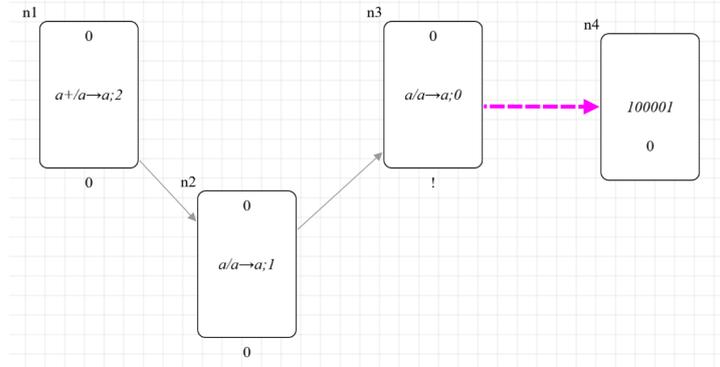


Fig. 1.2. Visualizing Π_{3k3} at time = 6

2 Preliminaries

The aim of this section is to discuss key concepts on the implementation of an SN P systems simulator on the web. In order, this section will discuss what SN P systems are, how they are discretely represented in a simulator, what web-based simulators are, and what their benefits are in comparison to simulations running on a local device.

2.1 Spiking Neural P Systems

A branch of membrane computing [26] and considered as part of the third generation of neural networks [16], spiking neural P systems (SN P systems) are defined as a set of neurons that communicate with each other through spikes, and that follow a set of rules [20]. These neurons all follow a global clock, and the encoding of information based on the time a spike arrives as well as the configuration of each component is the system is what decides its final output.

The model of spiking neural P systems, as covered by [25], is formally defined as the following:

Definition 1. A computing extended spiking neural P system, of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \dots, \sigma_m, syn, in, out), \text{ where :} \quad (1)$$

1. $O = \{a\}$ is the singleton alphabet (a is called spike);
2. $\sigma_1, \dots, \sigma_m$ are neurons of the form $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, where:
 - (a) $n_i \geq 0$ is the initial number of spikes contained in σ_i ,
 - (b) R_i is a finite set of rules of the following two forms:
 - i. $E/a^c \rightarrow a^p; d$, where E is a regular expression over a and $c \geq p \geq 1, d \geq 0$;

- ii. $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a^p; d$ of type (1) from R_i , we have $a^s \notin L(E)$;
- 3. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $i \neq j$ for all $(i, j) \in syn$, $1 \leq i, j \leq m$ (synapses between neurons);
- 4. $in, out \in \{1, 2, \dots, m\}$ indicate the input and the output neurons, respectively.

Rules of type (1) are known as firing or spiking rules, while the rules of type (2) are forgetting rules. If an SN P system has firing rules where $p = 1$ for all rules - in other words, each neuron only emits one spike - then it is said to be a standard or non-extended SN P system. The firing rules as described in (1) state that as long as the number of neurons meet what is stated in the regular expression E , r spikes are consumed and p spikes are fired and sent to the connected neurons after a delay d .

The forgetting rules simply state that if a neuron contains exactly s spikes, then the neuron is emptied of the containing spikes.

When neurons fire, they all follow a global clock, thereby synchronizing all of the neurons to work in parallel. The sequence in which the spikes enter the environment is referred to as a *spike train*. Unlike other examples of neural networks [16], the spikes in an SN P system are all the same. As such, it is when the spikes appear based on the rules within a neuron and its initial configuration that matter.

SN P systems have many capabilities. Even standard SN P systems turn out to be Turing complete [20]. Including systems that use extended rules, they can be found to emulate boolean circuits and sort a vector of natural numbers [21]. The parallel nature of these systems also prove their ability to solve known NP-complete problems [27] such as the Subset Sum and SAT problems in constant time [22], and can even be found to solve the SAT problem in strictly four steps [8] given exponentially calculated, precomputed resources. In more practical areas of Computer Science, SN P systems have been designed to use the RSA cryptographic algorithm [14]. This was demonstrated to reduce the complexity and computational power typically needed in asymmetric cryptographic systems.

Simulations show the theoretical capabilities of SN P systems, however, they are hindered by current hardware. While [8] and [22] do discuss solving SAT in constant time, this is only possible with systems capable of holding large amounts of memory that is not feasible with current technology. Parallelism is also an issue [5], although less than before, due to parallel computing platforms such as *CUDA*. But even then, results vary depending on the power of the specific Graphical Processing Unit (GPU) being used.

2.2 Matrix Representation of SN P systems

One notable way of representing SN P systems is through matrices. This representation makes it easy to assess the next configuration of an SN P system while knowing the previous configuration [37]. While the precise definition can

be found in the cited source, some of the primary elements can be listed as the following vectors and matrices:

- *Configuration Vectors* contain the amount of spikes present in each neuron at some time step.
- *Spiking Vectors* shows whether the rule of a neuron has been chosen and applied or not at some time step.
- *Spiking Transition Matrix* is a matrix containing a $n \times m$ matrix, where n is the number of neurons and m is the total number of rules. Each element of a matrix at some time step says whether or not a rule within a neuron has consumed some number of spikes, if a rule is applied and produces some number of spikes to a neuron, or if a rule is not applied to a neuron.

Variations of this format have been made to account for extended systems. For example, a matrix representation to account for the delays of rules and the status of a neuron (whether open or closed) was shown in [6]. Recent research analyzes further how other classes of SN P systems associated to a set of routing problems can be represented through matrices [15]. Improvements were also made in terms of space efficiency by using alternate methods to represent sparse matrices [2].

Using a matrix to represent a system will aid in simplifying its representation and enhance how fast the system runs in a parallel simulator and some concepts can be carried into visual simulators as well.

2.3 Web-based Simulations

The issue of hardware and accessibility can be alleviated through the use of web-based simulation, which is the idea of using resources offered by the World Wide Web in the area of simulation [11]. The concept is as old as the Internet itself, offering many variations to its developers: simulating locally, remotely, or hybrids thereof.

Local simulation is where both the simulation engine as well as the visualization or animation engine run from the client's side. The client also acts as the server distributing the simulation to the browser. Remote simulations are where both visualization and simulation engines are run from a server. Although overload can occur due to a server being tasked to run both engines, the benefit of remote simulations is that any computer can access the simulation as its results are simply being sent from a server doing all the computations. Lastly, hybrid simulations take the benefits of both, wherein the visualization engine runs on the client side and both simulations and computations are run on a server.

The web-based visual simulator from [11] shows the benefits of using the web to develop a simulator such as the multitude of open-sourced frameworks available, whether they be in JavaScript or other languages. The effect of this adds to the highly accessible nature of web-based simulations. While there are other pros to using them – such as cross-platform capability, collaboration, and integration – [11] its main drawback depends on both the Internet connection between the client and server as well as the stability of the servers that the application is being run from [10].

3 Related Works

Snapse [10] is a visual simulator built on Unity that is characterized as a GUI based on the classification given in [35], with the objective of aiding the efficiency of research on SN P systems and simplifying the representation of models for fields outside of computer science. The tool represents SN P systems graphically using rectangular nodes containing spikes and rules for the neurons and directed edges for the synapses. It also visualizes the opening and closing of neurons, and provides animations for the releasing of spikes. It has the ability to simulate SN P systems with delays and extended rules while generating a bitstring as its output. It also allows users to save configuration files written in human-readable format inspired by P-Lingua [31] and the matrix representation presented in [37].

A notable feature of Snapse is its application of non-determinism, either through a guided mode, where the user chooses which applicable rule a neuron is to follow, or in a pseudorandom manner. Snapse also provides a Choice History window where the rules chosen during points of non-determinism are recorded. In Section 4.5 we discuss some of the disadvantages of using Snapse in comparison to our visual simulator, WebSnapse.

The following tools are references for Snapse and are used for comparison.

P-Lingua [31] is a programming language for Membrane Computing that provides the standard in defining P systems. As part of this project, a Java library pLinguaCore was produced as a framework for simulating P systems. It has been extended to include definitions for SN P systems and to simulate computations of such models [23]. P-Lingua partly inspired the syntax for the configuration files used in Snapse.

MeCoSim (Membrane Computing Simulator) [29] is a software application integrated with P-Lingua that allows users to design GUIs and generate custom simulators for different types of models based on P systems. It allows for the validation of computational models and lets end-users unfamiliar with computers and membrane computing to simulate virtual experiments through tabular input with the addition of displaying graphs for verification purposes.

UPSimulator [17] is a general purpose simulator for P systems extending from P-Lingua. It uses its own object-oriented language, UPLanguage to define and represent cell-like, tissue-like and neural-like P systems. Its goal is to be a flexible and extendable framework in order to adapt to new models that will be introduced in the future. It is however limited to being a text-based simulator with no graphic visualization for P systems.

Snoopy [19] is a software tool that simulates Petri net classes in a graphical and unified modelling environment for the visualization and verification of models in systems and synthetic biology. JFLAP (Java Formal Languages and Automata Package) [32] is an interactive software written in Java consisting of graphical tools used to aid users in learning and experimenting with basic concepts in formal languages and automata theory. Both are visual tools with design and features that inspired Snapse such as the graphical representation of the models, animations as information travels between nodes, and lastly the handling of non-determinism. For this study, it is important to note as reference

that while Snoopy’s software is not web-based, the animation of Snoopy files is supported on web browsers through XSL transformation to SVG and JavaScript [33].

4 WebSnapse

The following section shows the features of the web-based visual simulator for SN P systems, WebSnapse. This covers its basic functionalities, how the simulator computes each configuration of the system, and how the software was developed. The code can be viewed and downloaded at <https://github.com/chinadupaya/WebSnapse>. The live version can be used at <https://chinadupaya.github.io/WebSnapse>.

4.1 Basic Functionalities

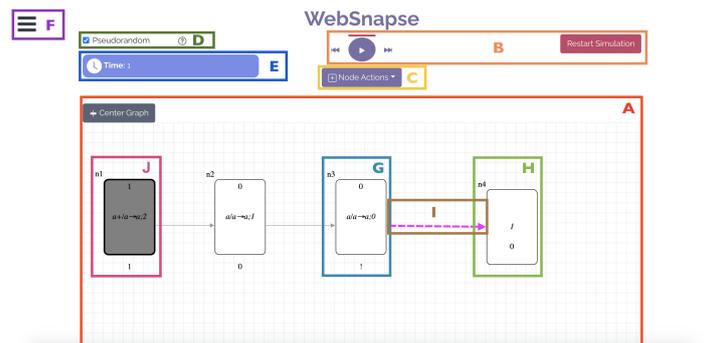


Fig. 4.1. Basic WebSnapse interface during simulation of $\Pi_{3 \times 3}$ from Figure 1.2 at time $t = 1$ with annotated labels

Basic Interface Figure 4.1 shows the basic interface of WebSnapse, consisting of the following components labelled as such:

- **A:** The Workspace - This is where the graphical representation of the system renders. Here the user can design their SN P system, connect neurons via directional synapses, zoom in and out, pan around the canvas, and view the simulation as it runs the system.
- **B:** Simulation Options - The user can run the simulation with step forward, step back, and automatic play buttons. Restart allows the user to start from the beginning. More information on how the play buttons function can be found in Section 4.3.
- **C:** Node Actions - These are the options to create, edit, and delete neurons.

- **D**: Pseudorandom Option - This checkbox dictates whether the simulation runs in pseudorandom mode when checked or guided mode otherwise. Both modes are explained in Section 4.1.
- **E**: Time Display - This element displays the current time step during simulation.
- **F**: Menu Actions - This menu contains the other available functions of WebSnapse namely, to save and load XML files (expounded on Section 4.1), to view choice history, to download sample SN P systems, and to replay the WebSnapse tutorial.

Before finalizing the current features and interface of WebSnapse, a prototype was created using the technologies being used for the application as discussed in Section 4.2. This prototype [34], shows how animations for SN P systems can be executed. An early version of the simulation and converter functions mentioned in Section 4.3 was used as well.

Neurons WebSnapse divides neurons into two types:

1. Regular Neurons - As highlighted by label G in Figure 4.1, they contain the ID, label, rules, and starting spike number. During simulation, the number below the neuron represents the current delay before the neuron sends a spike. This changes to an exclamation point when the neuron fires.
2. Environment Nodes - As highlighted by label H in Figure 4.1, they contain the ID and a bitstring symbolizing the presence or absence of a spike being sent to it at a time step. It should be noted that environment nodes are not part of the formal definition of SN P systems, but rather are only used for the purpose of implementation to represent the environment.

Type (1) neurons can be created, edited, and deleted in WebSnapse. Upon creation of a neuron, users can decide a custom label for the neuron, the initial number of spikes, and the rules of the neuron. Both types can be connected by synapses. It should be noted that type (1) neurons can connect to type (2) neurons, but the reverse cannot occur since type (2) neurons are only meant to receive spikes. They are meant to signify the outside environment that receives the output of system.

WebSnapse provides several visual indicators to aid in simulation. As annotated in Figure 4.1, label I shows the animation that represents spiking from a source neuron to a destination neuron. Label J shows a closed neuron presented with a gray background and thicker outline.

Rules are read through the *RegExp* object of JavaScript. Once all entered rules are read as valid, the application saves the rules into the proper neuron object it should be associated with. Rules are of the form $E/a \rightarrow p; d$ where:

- E is a regular expression of the format $(a^i)a^j*$, where $i, j \geq 0$ but both cannot be equal to 0. All acceptable rules are bound by this format, which is enough to create a Turing complete machine, but it is recommended that future works increase the allowed rules in the system.

- a is the number of spikes to be consumed by the neuron (ex. a for one spike, aa for two spikes).
- p is the number of spikes to be produced by the neuron which can use the previous form of a , aa , $aa..a$ or 0 if it is a forgetting rule.
- d is a non-negative integer representing the delay.

XML Support Systems created in WebSnapse can be imported from and exported to an XML file. WebSnapse takes the object containing the system’s original configuration and converts it into XML syntax for a more readable format. Users can then use any text editor to make changes to the elements on the file, such as creating more neurons or editing what nodes are connected. The edited XML file can then be loaded back into WebSnapse, provided there are no errors in the formatting. The new system can be viewed from there.

The XML code in Listing 1.1 is a general representation of an SN P system uploaded to WebSnapse.

Listing 1.1. XML representation for System Π_{3k3}

```

<content>
  <neuronName>
    <id>{string}</id>
    <position>
      <x>{int}</x>
      <y>{int}</y>
    </position>
    <rules>{string: rule1 rule2... ruleX}</rules>
    <startingSpikes>{int}{default: 0}</startingSpikes>
    <delay>{int} {default: 0}</delay>
    <spikes>{int}</spikes>
    <isOutput>{boolean}</isOutput>
    <out>{string}</out>
  </neuronName>
</content>

```

As shown in Listing 1.1, the whole system is encapsulated within ‘content’ tags. A neuron object is closed by their ‘id’ as tags. A neuron’s type is differentiated by the ‘isOutput’ tag containing ‘true’ for environment nodes and ‘false’ if otherwise. If a neuron is an environment node, an extra tag ‘bitstring’ will be present that contains its output.

The rules are a slightly different format from how they appear in the software. Instead of typing ‘– >’ for a right arrow, the greater than character ‘>’ is replaced with ‘>’. This is to prevent parsing errors from occurring when WebSnapse scans the uploaded XML file as the closing brackets of tags also end with a greater than symbol.

At the current version of WebSnapse, not all errors within an XML file have been accounted for. Errors in an XML file parsed by the software could lead to a white screen. The user can clear local storage to start a new session with WebSnapse or fix the error returned by the program.

Pseudorandom and Guided Mode Since the rules of SN P systems are chosen non-deterministically, non-determinism is emulated in a pseudorandom manner. WebSnapse, similar to its predecessor, offers a guided mode and pseudorandom mode for choosing which rule should be applied to a neuron during points of non-determinism. When guided mode is active and two or more rules can be applied to a neuron at a time step, a modal dialog - a small window that appears in front of the main interface - pops up to let the user choose which rule to apply. When pseudorandom mode is active, the application chooses which rule to apply.

Choice History Table WebSnapse provides the option for users to view the rules applied by each of the regular neurons per time step during the simulation. This can be found in the ‘Choice History’ tab located within the menu actions labelled F in Figure 4.1). It also provides the current bitstring per time step for environment nodes. The choice history table serves as a single view summary of the simulation.

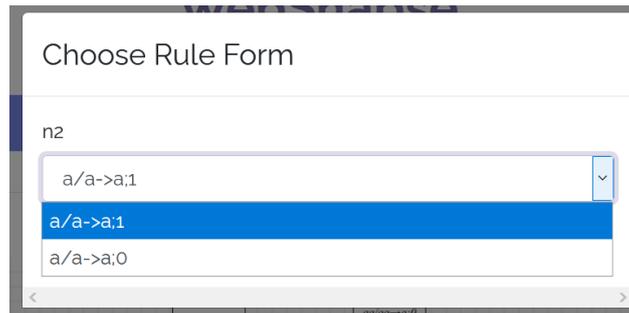


Fig. 4.2. Sample guided mode modal for Π_1 from Figure 5.1

A sample choice history table is shown in Figure 4.3 featuring the Π_{3k3} SN P system described in Section 1. The first column provides the simulation’s time steps. The following columns contain the neurons in the system as the header, with their corresponding rule per time step if applicable, and none if otherwise. For the last column, since $n4$ is an environment node, the bitstring printed per time step is displayed instead. From here, we can see that $n4$ received spikes at time = 0 and time = 5. The distance between these time steps is interpreted as distance = 6 because we count from 0. Users may find it convenient to use this table when focused on observing the output.

Additional Functionalities WebSnapse has the following additional features to aid in the creation and simulation of systems:

Time	n1	n2	n3	n4
0	a/a→a.2	No applicable rule.	a/a→a.0	1
1	a/a→a.2	No applicable rule.	No applicable rule.	10
2	a/a→a.2	No applicable rule.	No applicable rule.	100
3	No applicable rule.	a/a→a.1	No applicable rule.	1000
4	No applicable rule.	a/a→a.1	No applicable rule.	10000
5	No applicable rule.	No applicable rule.	a/a→a.0	100001
6	No applicable rule.	No applicable rule.	No applicable rule.	1000010

Fig. 4.3. Choice history table for Π_{3k3} where $k=1$.

- *Tutorial* - As shown in Figure 4.4, WebSnapse provides a tour around its interface and instructions for creating systems. Upon completion, the tutorial does not appear again.
- *Autosave* - In the case unintentional reloads or exits, WebSnapse sends a warning alert and also provides an autosave feature to allow the user to return to their work upon reopening. While in use, the application saves the current system to the browser's local storage after every few edits and at the start of simulation.
- *Downloadable Samples* - Users have the option to download sample files within the menu actions. The available SN P systems are listed in Figure 4.5.

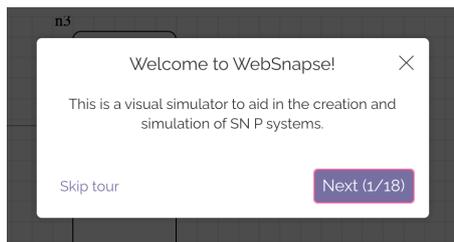


Fig. 4.4. WebSnapse tour that starts when first opening the web application

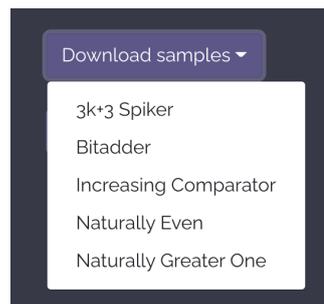


Fig. 4.5. Sample SN P systems that can be downloaded from WebSnapse

4.2 Technologies and Development

WebSnapse was build with the React Javascript library as the base for its components. Since the website does not have any dynamic content and simply displays the simulation interface for the user, WebSnapse can be hosted on Github Pages. The GUI of the simulation was created using the open-sourced graph library, Cytoscape.js (referred to as simply Cytoscape in this paper) [13]. Notable features include, distinguishing between graph types, traversal, gesture support, and animations.

During the development of WebSnapse, we encountered some issues when rendering an SN P system using Cytoscape. One is the inability to output superscript characters on any node labels within Cytoscape’s canvas element. In JavaScript, when the ‘sup()’ function is used to display a string in superscript, the string is placed in a ‘<sup>’ HTML element. Cytoscape is then unable to render the superscript character in the text and displays the HTML code as a string instead. Extension libraries can be used but the dragging and panning actions to the workspace become slowed and the formatting of rules become distorted while changing node positions as shown in Figure 4.6. Because of its effect on general performance, we decided against using it and this is why the number of spikes is represented in numerical form and necessary symbols are not superscript when printed.

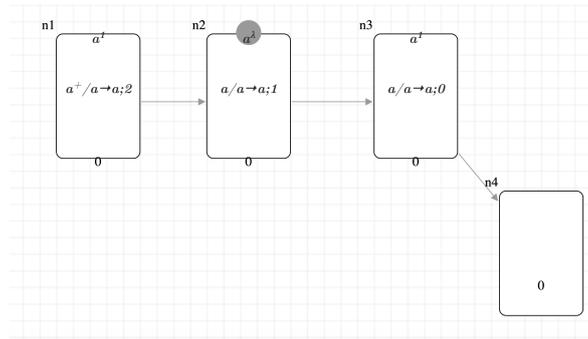


Fig. 4.6. WebSnapse implemented with cytoscape-node-html-label library extension highlighting formatting issues.

Each configuration in a certain time step is saved into an object variable in the React project. For Cytoscape to be able to parse and present the configuration, a converter function is created. This converter function accepts the current system from the React project and outputs it into another object format that is easier for Cytoscape to parse through. More details on this can be found in Section 4.3.

Cytoscape outputs data to be viewed by a user through nodes. Each node outputs some data of a neuron that has to be visible to the user. Out of the

eight elements within a neuron found in the XML file in Section 4.1, four are made visible to the user via nodes: the *id*, *rules*, *spikes*, and *delay*. Synapses are created through the *out* element. Environment nodes are only comprised of two elements: the *id*, and the *bitstring*.

Despite the current implementation using objects, the original goal was to make use of matrices similar to what was used in [37]. This goal was made to improve the performance of the simulation. However, given the way Cytoscape parses the data as an object, we found that continuously converting from matrices to objects and vice versa would add too many unnecessary steps to the program. Since WebSnapse aims to be a simple visual simulator, this would be extraneous.

4.3 Architecture

Figure 4.7 shows the general structure of the WebSnapse application. The *User* can be seen interacting with the *GUI*, which interacts with the *Controller*. The primary functions within the *Controller* are defined as follows:

1. *step()*: This function runs when the current SN P system is being played one time step forward. This function is in charge of deciding which rules will be active in a neuron, subtracting from the current number of spikes in a neuron and sending the correct amount of spikes to connected neurons. The resulting configuration from this function is saved to *Local Storage*. Algorithm 1 explains how the next step is generated.
2. *stepBack()*: This function runs when the user wishes to move back a time step. It accesses *Local Storage* to retrieve the configuration of the system from the previous time step. It should be noted that the current time step will be lost and the *step()* function will have to run again.
3. *convert()*: This function is in charge of converting the neuron objects to a more understandable format for Cytoscape. Every new time step or configuration of the system that the *User* needs to view visually runs through this function first.

Local Storage saves all previous steps the system has gone through to be accessed by the *stepBack()* function. Here, the configuration of all the neurons in the system are stored and cannot be changed as long as the simulation has not been reset. The template for the neurons themselves are simple; they only keep the current number of spikes, the currently active rule, and all the rules within a neuron. In summary, this is a completely client side simulation. All data is saved in the local storage of the users web-browser, so all information is stored within the users machine.

To prove the correctness of Algorithm 1, the input C_k must be analyzed. C_k is composed of a finite number of neurons, all of which contain a finite number of elements. The algorithm is composed of three loops that process this finite number of neurons and is then set to halt with the updated values of the system. This produces the next configuration C_{k+1} .

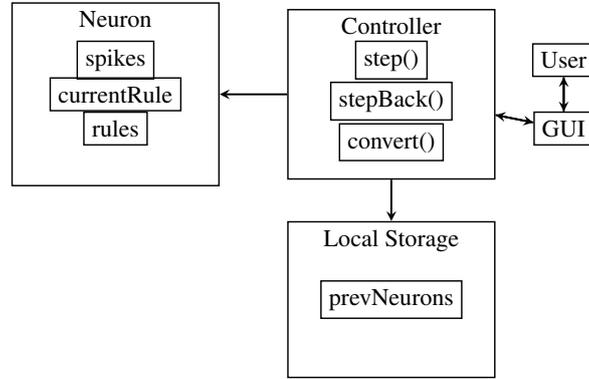


Fig. 4.7. The WebSnapse simulation architecture.

Algorithm 1: Next State Generator

Input: The current configuration C_k
Output: The next configuration C_{k+1}

```

1 for neuron in neurons do
2   if neuron does not have active rule then
3     program chooses possible rules in a neuron
4     if pseudorandom mode is active then
5       program picks an active rule randomly
6     end
7   end
8 end
9 if guided mode is active then
10  prompt user to pick an active rule for neurons
11 end
12 for neuron in neurons do
13   if neuron is not environment node AND has active rule then
14     if delay ≥ 0 then
15       delay ← delay - 1
16     else
17       consume and record spikes to send to connected neurons
18     end
19   else
20     if environment node is not receiving spikes then
21       add '0' to environment node bitstring;
22     end
23   end
24 end
25 end
26 send spikes to be received by each neuron. '1' if the neuron is an environment
   node.

```

Algorithm 1 has three loops that run depending on the given n neurons. The first from lines 1-7 to check for active rules, the second from lines 12-25 to execute the chosen rules, and line 26 is a for loop that sends the correct number of spikes to be received by each neuron. This shows a complexity of $O(n)$ for a change from one state to the next.

4.4 Testing

General Testing To test the correctness of the simulator, we assessed with already made SN P systems. Aside from the simple example of the $3k + 3$ spiker Π_{3k3} [20], other systems are covered and tested for their correct output. These can be found in Section 5 as well as in the Appendices. They can also be downloaded and tested using the links provided at the beginning of Section 4.

The testing library *react-testing-library* was used to check if there were any errors in the code and to easily examine if the program was reacting properly to user actions. We found that that most user actions were working as designed such as forms submitting correctly and the main component that contains the applications rendering the correct text. Smaller components such as the side menu and the tour that introduces users to WebSnapse were not able to be tested. It was also observed that the library would find errors in the rendering of the canvas generated by the Cytoscape component despite there not being any observable ones. The unstable nature of UI-driven tests and libraries are suspected to not be compatible with components that use canvases like Cytoscape.

Stress Testing Stress testing was also done with several types of systems. The goal was to see how many visual elements - the nodes mentioned in Section 4.2 - can be rendered in Cytoscape. We also wanted to observe if performance drops due to the simulation algorithm. WebSnapse was stress tested locally on a 2019 MacBook Pro with 8GB of RAM and a 1.4 GHz Intel Core i5 processor running macOS. The web browser used was Google Chrome. Python scripts were used to generate the different systems tested by inputting the desired number of neurons and creating the XML file as the output.

The first type of system is a simple ‘One Spike Chain’ Π_{c1} . A sample of the created system is shown in Figure 4.8. It shows neurons connected in a unidirectional manner. Each neuron, except for the environment node, has the same rule $a/a \rightarrow ; 0$, with the first neuron in the system containing one spike. For the test to be completed, the spike in the starting neuron must reach the end of the system. This happens at the time step one less than the total number of neurons.

Π_{c1} was generated with neuron counts starting at 50 and ending at 850, with intervals of 50. A 200-neuron simple chain was found to crash at 185 time steps without completing its simulation. Due to this, we set out to find the greatest number of neurons that would still be able to reach the end of the simulation. Until 192 neurons, simulations were able to complete and send spikes to the environment node. After this point, the browser’s local storage begins to exceed

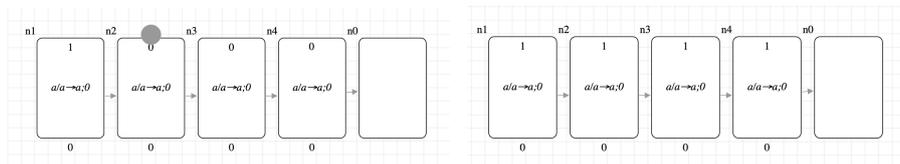


Fig. 4.8. Π_{c1} with 5 neurons at time = 0. **Fig. 4.9.** Π_{c2} with 5 neurons at time = 0.

12.5 megabytes. The 193-neuron system was able to complete its simulation, however, WebSnapse would proceed to crash. We found that as the number of neurons increases, more bytes were consumed per time step and the sooner the limit for local storage is reached, therefore less time steps are simulated.

At 450 neurons, the software experienced slight lag while panning and zooming in and out around the canvas where the system was being rendered. This lag during canvas related actions was made even more noticeable at 650 neurons and more. Since no lag was seen when the simulation ran or if the canvas was zoomed in with a lesser number of elements seen, the lag was most likely caused by Cytoscape having to render many elements. This is not due to any computations being made by the simulation since the number of neurons and rules is still relatively small from a computational standpoint.

The next type of system tested was the ‘All Spike Chain’ Π_{c2} . It is similar to the Π_{c1} type of system however, instead of the first neuron beginning with one spike, all neurons would start with a single spike as shown in Figure 4.9. The recurring rule is also different. Specifically, it is $a^+/a \rightarrow a; 0$ to facilitate continuous spiking. The purpose of testing this type of system was to see if the number of time steps that can be completed was only dependent on total neuron count, or if the number of neurons spiking at a time would serve as a factor. This type of system was also generated with neuron counts starting at 50 and ending at 850, with an interval of 50.

Π_{c2} was found to fail much sooner than Π_{c1} . At 192 neurons, the all spike chain system Π_{c2} crashed at 176 time steps, while the one spike chain system Π_{c1} was still able to finish its simulation at 191 time steps. Results found that storing the rule (in *rules* of a neuron object) $a/a \rightarrow a; 0$ only takes 16 bytes, while $a^+/a \rightarrow a; 0$ takes 18 bytes due to being one character longer. Along with this, the record of which rule is running - stored as *currentRule* and *chosenRule* - consumes 36 bytes per neuron. So while in the previous system, only one neuron would hold the extra 32 bytes, this system would have multiple neurons containing the extra *currentRule* and *chosenRule* attributes. Figure 4.10 shows as the number of neurons increase, the extra storage used affects how many time steps a system is able to complete. The figure also visualizes the limit of simulated time steps as the number of neurons reach certain points.

Another type of system used for stress testing was a fully connected graph from [4], the ‘Benchmark Complete Graph’ Π_{bc} , which can also be viewed in Figure 4.11. Each neuron has two spiking rules, namely $(a^2)^*/a \rightarrow a$ and $(a^2)^*/a^2 \rightarrow$

Number of neurons vs. simulated time steps per chain system

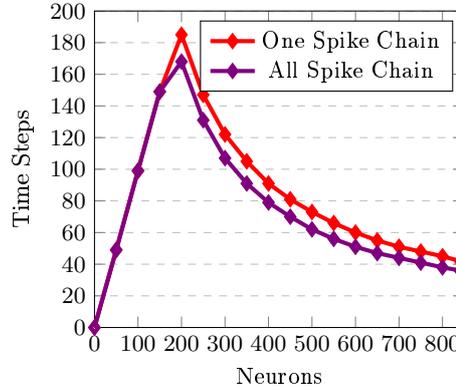


Fig. 4.10. Comparison between chain SN P systems with one spike Π_{c1} versus each neuron containing a spike Π_{c2} .

a^2 . They dictate that whenever the number of spikes in a neuron is a multiple of 2 then each neuron has to non-deterministically choose between the two rules. Π_{bc} was originally used as a tool for stress testing for parallel implementation of an SN P simulator in a GPU. Aside from testing a system with all neurons sending multiple spikes through all synapses continuously, Π_{bc} was also used to see how long the system can mimic parallelism before the processing of the simulation becomes noticeable.

Similarly, another type of system referred to as ‘Simple Complete Graph’ Π_{sc} was created with only one rule $a^*/a \rightarrow a; 0$ and can be seen in Figure 4.12. This system was created as a comparison to Π_{bc} in order to see if points of non-determinism play a role in the performance of the software. The systems Π_{bc} and Π_{sc} were tested using Pseudorandom Mode so that the tests are able to run automatically.

We were able to create several variations of the systems, with 4, 8, 16 and 32 neurons per system. As the number of neurons increases in the complete graph, the number of synapses coming from a neuron increases as well. The longer the systems run, the less time steps it takes for Cytoscape to start lagging. We experienced spiking animations no longer running for some synapses. When all 32 neurons would send spikes at the same time, the rendering performance of the system suffers due to multiple synapses being animated.

Similar to the chain systems, once the amount of local storage consumed begins to approach 12.5 megabytes, the complete graph systems starts to crash. Figure 4.13 summarizes how many neurons there are in a system and how many time steps it takes to crash. Despite having the same number of neurons and synapses, Π_{sc} was able to consistently run for a longer amount of time than Π_{bc} . Having less rules, and therefore less bytes used per neuron allows Π_{sc} to consume less storage per time step. Another observation is that the difference between

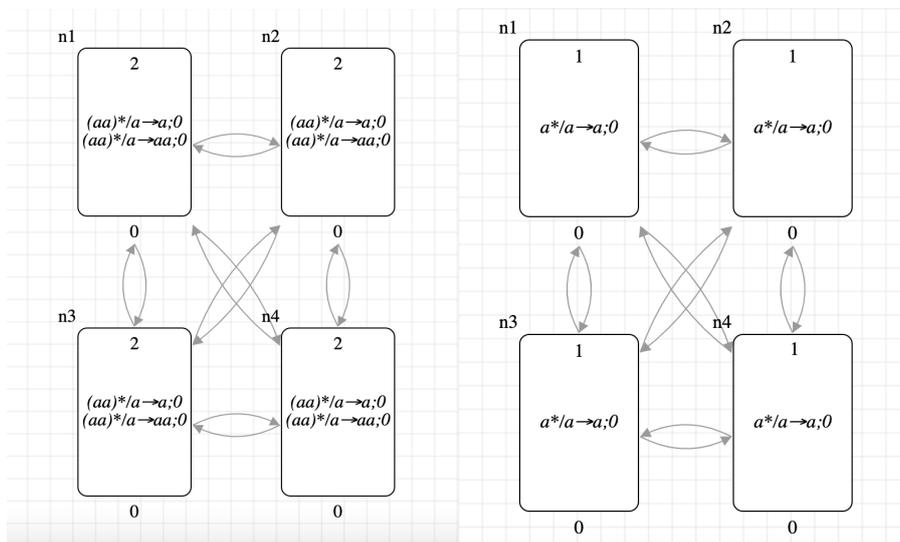


Fig. 4.11. Π_{bc} with 4 neurons.

Fig. 4.12. Π_{sc} with 4 neurons.

the number of time steps completed per system decreases as the number of neurons increases. This is also due to the reason mentioned previously regarding there simply being more elements and thus more storage consumed such as what occurred with the chain systems in Figure 4.10.

Overall, the number of time steps that a simulation can run is greatly limited to the amount of local storage available, which will differ based on the web browser the user is on. Since the current configuration or state of the whole system is saved into local storage per time step during simulation, the number of neurons will greatly impact the amount of data stored. However, it isn't only the number of neurons affecting it. The character length consumed by the rules, number of spikes, and whether a neuron has an applicable rule in play contribute to it as well. It was also observed that the actual computation behind the simulation runs normally despite the complexity of a system. The rendering of the visual elements is the aspect that is affected.

4.5 Comparison with Other Tools

An important objective of WebSnapse is to reach feature parity with Snapse. Based on Table 1, we were successful in recreating the core features of Snapse onto a new web application.

The distinction of WebSnapse using web-based simulations is important to consider in creating a visual SN P system simulator focused on accessibility. As a web application hardware is not a major issue since WebSnapse has cross-platform access, regardless of operating system and device. All it requires is the

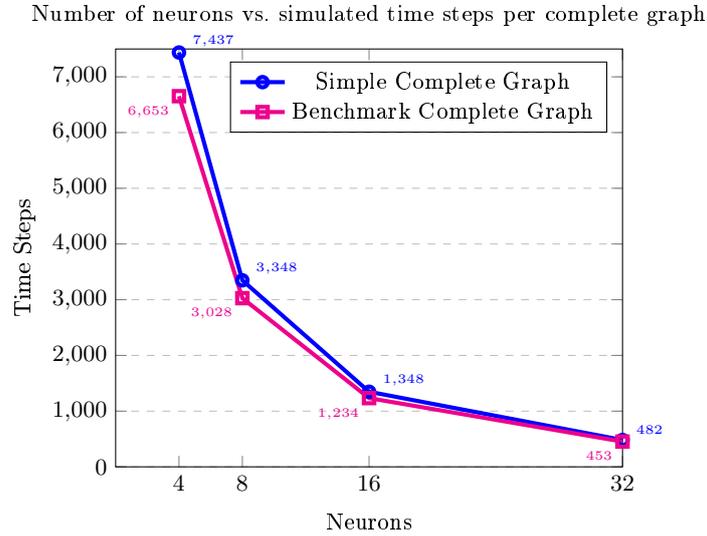


Fig. 4.13. Comparison between complete SN P systems with 2 rules to choose for non-determinism versus only one rule for each neuron.

use of any web browser. As a desktop application, Snapse provides different versions for two computer operating systems, namely Windows and Linux. Unity, the engine used, has build specifications depending on the target platform. This may have limited the development of versions for other operating systems. Deployment for a web application is done easily, as it is rolled out straight to the host server and available once installed. Given its single code base, it is easy to maintain WebSnapse and make changes for improvements. Its flexibility and scalability also expands the capacity for upgrades. While users would need a connection to the Internet in order to access the web application, there are also options that provide an offline experience. Electron [24] is a framework that allows desktop applications to be made from web technologies.

There are several other differences in the execution of other features. For a visual simulator, the graphical representation of SN P systems is a major aspect to consider. While both Snapse and WebSnapse display the spikes, rules and delay count, they have differences in the presentation of these details. In Snapse, a neuron is represented by a rectangular object with separate side-by-side sections for spikes and rules. WebSnapse attempts to follow closely the conventional visual representation found in [20] to aid users in generating figures. Another visual detail to consider is the animation to represent the action of spiking. Snapse displays this action of spiking through a yellow circle that travels through the synapse from a source neuron to a destination neuron. This is likely to be missed if the distance between neurons is short. WebSnapse uses dashed

Table 1. Comparison of WebSnapse with Other Tools

	WebSnapse	Snapse	P-Lingua	JFLAP	Snoopy	MeCoSim	UPSimulator
Simulate SN P Systems	✓	✓	✓			✓	✓
General Solution to P Systems			✓			✓	✓
Graphical Presentation	✓	✓		✓	✓	✓	
Graphical Design	✓	✓		✓	✓		
Pseudorandom Mode	✓	✓	✓	✓		✓	✓
Guided Mode	✓	✓					
Animation	✓	✓			✓		
Configuration Files	✓	✓	✓		✓	✓	✓
Web-based Simulation	✓				✓		

lines that move along the highlighted directed edge. This animation remains on display even when the simulation is paused at a certain time step.

For running simulations, both Snapse and WebSnapse give the user the option to step forward, step back, and play automatically in time intervals. However, Snapse doesn't provide the choice to restart the simulation from its original configuration. This means the user would need to click the step back button multiple times until the first time step is reached. WebSnapse gives the user the option to restart the simulation from its initial configuration of spikes, empty bitstring outputs, and resets the time to zero. While WebSnapse went through stress testing as discussed in Section 4.4, we are unable to compare the limitations in the size and the simulation of systems since Snapse had not undergone any similar stress testing.

One functionality WebSnapse adapted from Snapse is the idea of a choice history table to highlight the chosen rule during points of non-determinism. This feature is only available for systems with neurons that have more than one applicable rule. WebSnapse's choice history table shows the rules used per neuron at each time step played in the simulation regardless of the presence of non-determinism. It also displays the current bitstring for environment nodes.

For saving and loading configuration files, Snapse uses its own file format with the file extension '.snapse'. WebSnapse uses XML as shown in Section 4.1. This takes into account possible integration with P-Lingua [31], as it can generate XML. With this, both tools provide a way to preemptively save the work of the user. Snapse autosaves the current system being edited into the user's computer as a separate file. The system it saves is at its current configuration of spikes and bitstring outputs. As a web application, WebSnapse autosaves the original state of the current system being edited into the browser's local storage and preloads it in the next session when WebSnapse is opened.

WebSnapse encompasses the most features in comparison to other tools as shown in Table 1. Both graphical presentation and graphical design are newer features for tools related to P systems. This is due to earlier simulators focusing on performance in terms of the scope of systems that can be run, and not the visualization. The tools unrelated to P systems have graphical features that are

also incorporated into WebSnapse such as JFLAP's use of a drag and drop gestures for their elements and Snoopy's animated token passing. Snoopy provides support for browser-based simulation, however the user would have to unpack a specific compressed file into the web server root folder, make several edits to the Snoopy files and possibly rename the files [33]. These steps may be inconvenient as compared to a straightforward web application.

5 Case Studies

To further understand the capabilities of the software, several SN P systems will be covered to explain its core features. These tests were used as simple tests for correctness in WebSnapse as mentioned in section 4.4. Throughout the paper, system Π_{3k3} from [20] was used in explaining SN P system and examining the different functionalities of WebSnapse. To show how WebSnapse tackles non-determinism, a naturally even number generator [20] will be observed. More practical applications of SN P systems will then be covered, namely addition with a bit adder system [18], and sorting [7].

5.1 Naturally Even Number Generator

To show the non-deterministic capabilities of the software, we use a system that uses non-determinism to generate some naturally even number, referring to the system as Π_1 . Its formal definition can be found in [20].

Figure 5.3 shows the system ending after 104 time steps. The distance between the two spikes is then 102, which is the generated even number. Notice that the pseudorandom checkbox is not ticked. With the system now set to guided mode, this allows the user to choose which rule should run in the case that a neuron can apply two or more rules at its current state. In this system, it is neuron $n2$. Setting the pseudorandom checkbox to true would let the application choose which rule to apply while the system is being simulated. Figure 5.2 shows how a user can choose which rule to apply in guided mode.

It may be difficult to observe the distance between spikes given a large bit-string output. For ease of checking in either guided or pseudorandom mode, the user can also view the rules chosen through the choice history table. The final row in Figure 5.4 shows how choosing rule $a/a \rightarrow a; 1$ at 101 time steps led to the end of the simulation. At 102 time steps, the partial result of the environment node gains one bit or spike as shown in Figure 5.5.

5.2 Bit Adder

Arithmetic operations are a necessity in a computing system. The system on Figure A.1, referred to as Π_2 , is based on [18] and adds two natural numbers. The system has three main neurons, with the leftmost two being the *input neurons* and the subsequent neuron being the addition neuron. The environment node shows the summed result in binary form. The two input neurons were set up in

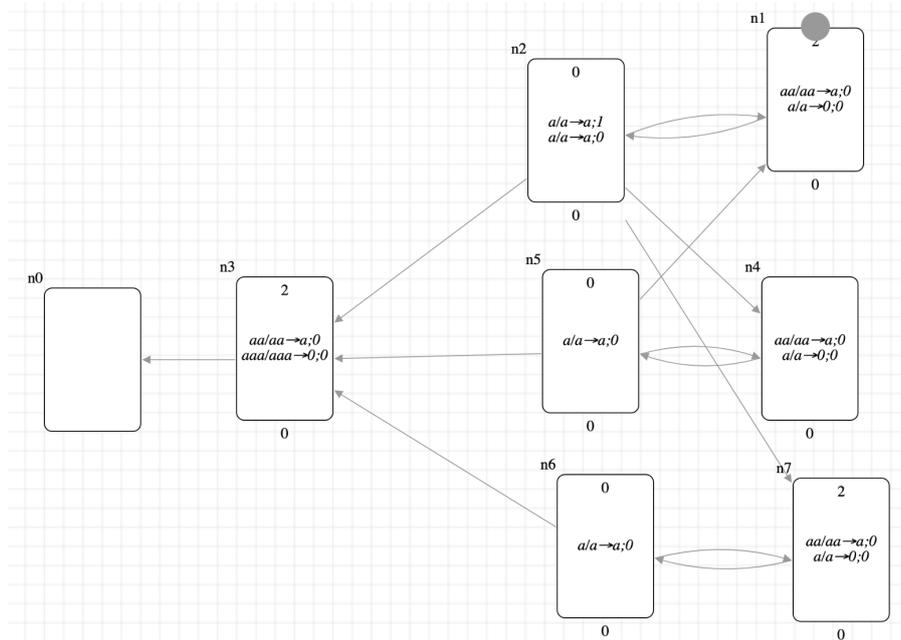


Fig. 5.1. Π_1 at time = 0.

a way that they send spikes in the order of their binary form (2_{10} as 10_2 , and 3_{10} as 11_2). This results in the environment node of $10_2 + 11_2 = 0101_2$ or 5_{10} .

5.3 Increasing Comparator

Another important function of a computing system is comparing and sorting values. Figure 5.6 shows an increasing comparator [7] for the numbers 5 and 8. The first layer of neurons contains the values to be compared, codified as the number of spikes. At each step they send spikes to both $n2$ and $n5$. As long as both $n2$ and $n5$ receive spikes, only $n2$ sends spikes to $n4$ and $n6$, the environment nodes. The minimum is then obtained once one neuron runs out of spikes to send. Since only one spike is being sent at a time, only $n5$ will send spikes while $n2$ applies its forgetting rule to the spikes it receives. The remaining spikes are then sent to the lower environment node $n6$.

The final values of the environment nodes can be viewed in Figure 5.7. The figure shows that the topmost environment node $n4$ was able to receive five spikes, but since the input neuron $n3$ ran out of spikes, the environment node $n6$ would receive the rest of the spikes. The system ends after 9 time steps.

This system can be expanded to sort more values and create bitonic sorters. One such system of size $n = 4$ can be found in the Appendix as Figure A.4.

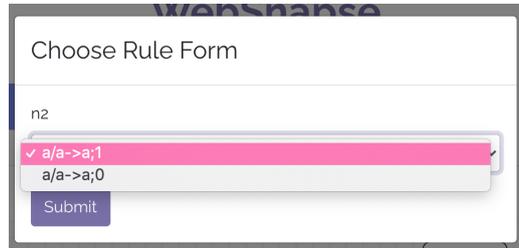


Fig. 5.2. User chooses rule $a/a \rightarrow a; 1$ through guided mode modal during simulation of system in Figure 5.1.

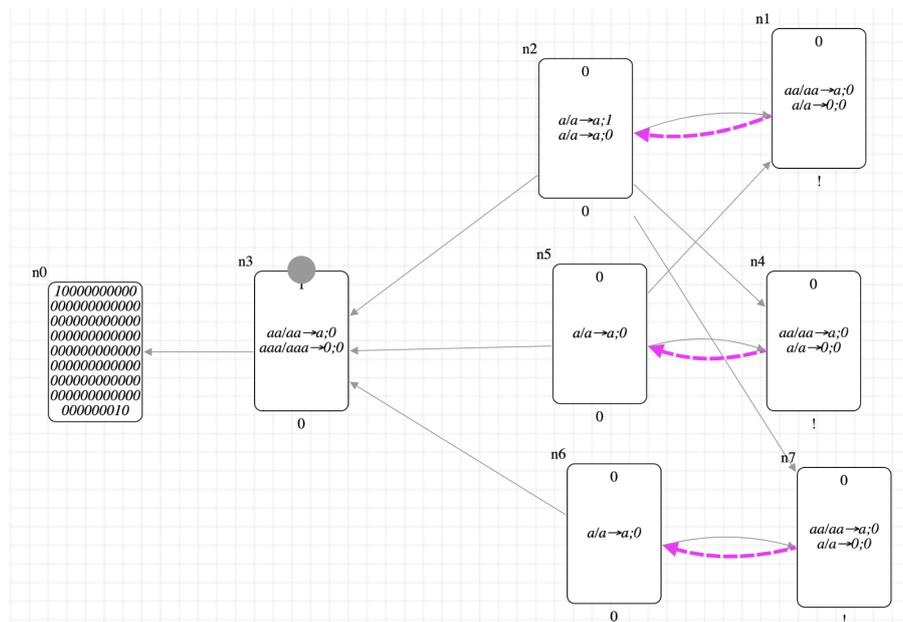


Fig. 5.3. Π_1 ends simulation at time = 104

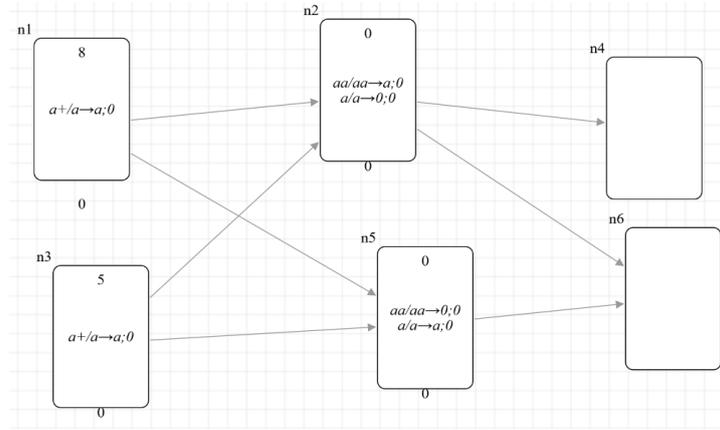


Fig. 5.6. Initial configuration of II_3 comparing the values 8 and 5 at time = 0.

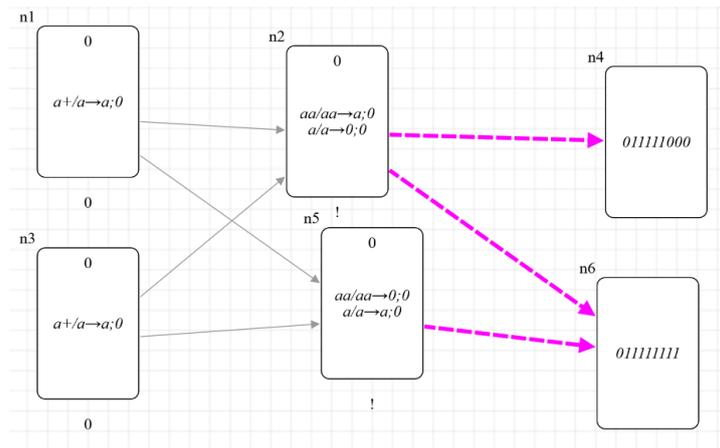


Fig. 5.7. The end of II_3 's simulation at time = 9.

6 Final Remarks

WebSnapse is a visual simulator that aims to match the capabilities of its predecessor Snapse while adding more functionalities to the original architecture. Its main focus is its availability as a web-based package for easier development and accessibility.

We were overall successful in reaching feature parity with Snapse, although there are differences in execution in terms of file formats used, and the visual representation of systems. During development, we also found some limitations when using Cytoscape in terms of state-specific styling of neurons and rendering issues while navigating the canvas with too many neurons on screen. While

WebSnapse focuses on being a visual simulator, its development makes evident the trade-off of being able to visualize systems versus the capability for bigger or more complex systems. This is from finding that the complexity of the systems created is dependent on the local storage and cache available on the web browser.

The user experience can be improved upon in several ways. The results from Section 5.1 suggest the possibility of allowing users to create patterns or configurations for what rules they would like to run during guided mode. This could be convenient when wanting to study specific outputs of a system, as it avoids continuously incurring the select rule modal dialog. In addition, providing a computation tree to view the number of spikes in a neuron per time step would also allow the user to have a more detailed view of their system after it has finished running. To address the difficulty in reading the length of the bitstring in the environment node, the spike strain can also be represented as the distance between each spike in decimal notation. Lastly, For testing the whole program along with Cytoscape, Section 4.4 shows that possibly using both end-to-end UI-driven tests with *react-testing-library* as well as headless browser testing libraries, such as *Puppeteer* or *Selenium*, would also help in creating a system that's certain to be more stable. To combat connection issues or for users interested in working offline, WebSnapse can be wrapped into a desktop application using Electron as mentioned in Section 4.5. Future work can also allow for users to make the SN P systems interact with each other through both input and environment nodes [36, 1].

The performance of the simulations themselves can be improved upon through modifications in WebSnapse's architecture. Section 4.1 shows the regular expressions currently allowed. This is limited due to the parsing capabilities of JavaScript's *RegExp* object in regards to balanced parentheses. Expanding the scope of regular expressions and creating a custom rule input parser allows for more variations of SN P systems to be accounted for. One example is an SN P system with structural plasticity (SNPSP system) where rules would include adding and deleting connections between neurons [12, 9]. Another is an SN P system with stochastic application of rules (\star SN P system) where during points of non-determinism, the selection of firing rules goes through a stochastic process [28]. Similar to the idea of a pseudorandom and guided mode, it is possible to allow users to provide probabilities for the rules or to have them determined in a stochastic manner. Possible integration with a GPU simulator running on a web browser [3] should also be considered.

Section 4.1 shows that there are still bugs that may occur when uploading an XML file with errors. Adding more sophisticated parsing upon loading a system can help the user identify the errors easily. A software similar to *kPWorkbench* may be useful in identifying the correctness of the system such as how it was used in [15].

The results found in Section 4.4 show that the amount of local storage available in the system is a bottleneck to how long a system can run. One solution is to allow computations to be made on a server that can store more data and allow for more parallel computations. This also lessens the resources to be used

on the client's side that may lead to better performance. Another alternative to this is optimizing the way configurations are currently being stored in local storage. This can be done by only saving the changes in state instead of the whole neuron object.

Acknowledgements

K.J. Ballesteros, I.C.H. Macababayao, and R.T.A. De La Cruz acknowledge support from ERDT scholarships of the DOST-SEI, Philippines. F.G.C. Cabarle acknowledges support from ERDT of DOST-SEI, and the Dean Ruben A. Garcia PCA from the University of the Philippines Diliman.

Appendix

A.1 Case Studies

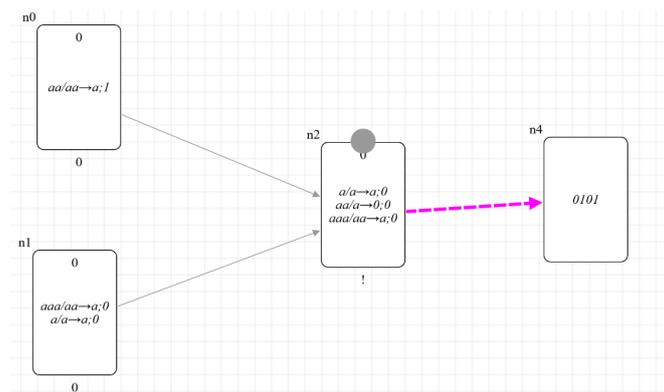


Fig. A.1. Π_2 from Section 5.2 adding natural numbers 2 and 3.

Time	n4	no	n1	n2
0	0	aa/aa→a1	aaa/aa→a.0	No applicable rule.
1	01	aa/aa→a1	a/a→a.0	a/a→a.0
2	010	No applicable rule.	No applicable rule.	aa/a→0.0
3	0101	No applicable rule.	No applicable rule.	a/a→a.0

Fig. A.2. Π_1 Choice History table from Section 5.1.

Time	n2	n4	n3	n5	n6
0	No applicable rule.	0	a/a→a.0	No applicable rule.	0
1	aa/aa→a.0	01	a/a→a.0	aa/aa→0.0	01
2	aa/aa→a.0	011	a/a→a.0	aa/aa→0.0	011
3	aa/aa→a.0	0111	a/a→a.0	aa/aa→0.0	0111
4	aa/aa→a.0	01111	a/a→a.0	aa/aa→0.0	01111
5	aa/aa→a.0	011111	No applicable rule.	aa/aa→0.0	011111
6	a/a→0.0	0111110	No applicable rule.	a/a→a.0	0111111
7	a/a→0.0	01111100	No applicable rule.	a/a→a.0	01111111
8	a/a→0.0	011111000	No applicable rule.	a/a→a.0	011111111

Fig. A.3. A portion of the Π_2 Choice History table.

A.2 Bitonic Sorter

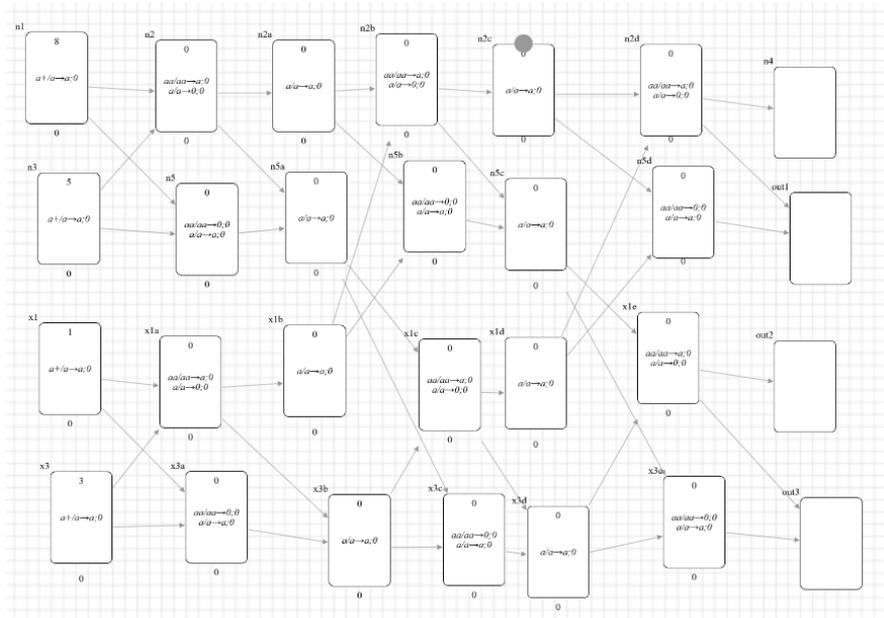


Fig. A.4. 4 neuron bitonic sorter with 8, 5, 1, and 3 from [7] based on Π_3 from Section 5.3.

A.3 Naturally Greater One

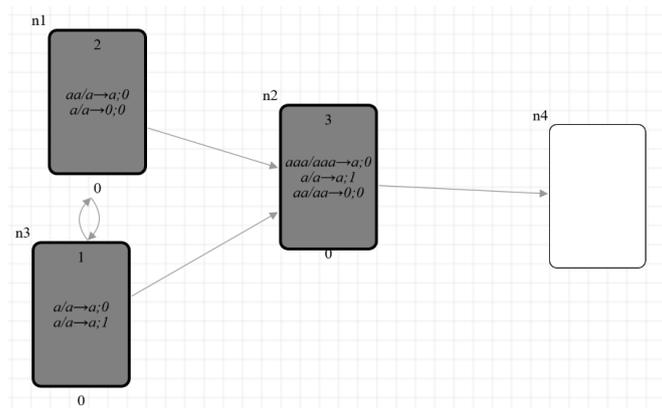


Fig. A.5. System generating a number greater than one from [20]. Closed neurons have a gray background and a thicker border.

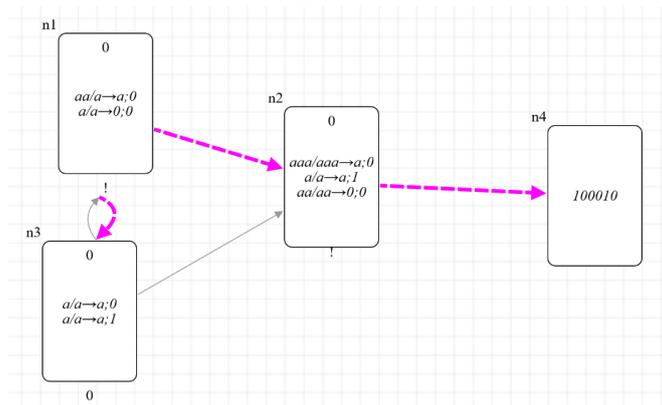


Fig. A.6. With 4 time steps between that 2 spikes, the output of the system is 4.

Choose Rule Form

n3-nm2MDsgNh

a/a->a:0

a/a->a:0

a/a->a:1

Fig. A.7. Guided Mode modal at points of non-determinism for neuron n_3

References

1. Adorna, H.N.: Computing with SN P systems with I/O mode. *Journal of Membrane Computing* **2**(4), 230–245 (2020)
2. Martínez del Amor, M.Á., Orellana Martín, D., Cabarle, F.G.C., Pérez Jiménez, M.d.J., Adorna, H.N.: Sparse-matrix representation of spiking neural P systems for gpus. *BWMC 2017: 15th Brainstorming Week on Membrane Computing* (2017), p 161-170 (2017)
3. Arian Allenson M. Valdez, Filbert Wee, F.G.C.C., del Amor, M.A.M.: Gpu simulations of spiking neural p systems on modern web browsers. (accepted) *International Conference on Membrane Computing (ICMC2021)*, Chengdu, China and Debrecen, Hungary, August 24-28, 2021 (2021)
4. Cabarle, F.G.C., Adorna, H.N., Martínez del Amor, M.Á., Pérez Jiménez, M.d.J.: Improving GPU simulations of spiking neural P systems. *Romanian Journal of Information Science and Technology*, 15 (1), 5-20. (2012)
5. Carandang, J.P., Cabarle, F.G.C., Adorna, H.N., Hernandez, N.H.S., Martínez-del Amor, M.Á.: Handling non-determinism in spiking neural P systems: Algorithms and simulations. *Fundamenta Informaticae* **164**(2-3), 139–155 (2019)
6. Carandang, J.P., Villaflores, J.M.B., Cabarle, F.G.C., Adorna, H.N., Martinez-del Amor, M.A.: CuSNP: Spiking neural P systems simulators in cuda. *Romanian Journal of Information Science and Technology* **20**(1), 57–70 (2017)
7. Ceterchi, R., Tomescu, A.I.: Spiking neural P systems—a natural model for sorting networks. *Proceedings of the Sixth Brainstorming Week on Membrane Computing*, 93-105. Sevilla, ETS de Ingeniería Informática, 4-8 de Febrero, 2008 (2008)
8. Chen, H., Ionescu, M., Ishdorj, T.O.: On the efficiency of spiking neural P systems. *Proceedings of the Fourth Brainstorming Week on Membrane Computing, Vol. I*, 195-206. Sevilla, ETS de Ingeniería Informática, 30 de Enero-3 de Febrero, 2006 (2006)
9. de la Cruz, R.T.A., Cabarle, F.G.C., Macababayao, I.C.H., Adorna, H.N., Zeng, X.: Homogeneous spiking neural P systems with structural plasticity. *Journal of Membrane Computing* **3**(1), 10–21 (2021)
10. Fernandez, A.D.C., Fresco, R.M., Cabarle, F.G.C., de la Cruz, R.T.A., Macababayao, I.C.H., Ballesteros, K.J., Adorna, H.N.: Snpase: A Visual Tool for Spiking Neural P Systems. *Processes* **9**(1), 72 (2021)
11. Fonseca, Í.A., Gaspar, H.M.: A Prime On Web-Based Simulation. In: *ECMS (European Council for Modelling and Simulation)*. pp. 23–29 (2019)

12. Francis George C. Cabarle, Henry N. Adorna, M.J.P.J., Song, T.: Spiking neural P systems with structural plasticity. *Neural Computing and Applications* **26**, 1905–1917 (2015)
13. Franz, M., Lopes, C.T., Huck, G., Dong, Y., Sumer, O., Bader, G.D.: Cytoscape.js: a graph theory library for visualisation and analysis. *Bioinformatics* **32**(2), 309–311 (2016)
14. Ganbaatar, G., Nyamdorj, D., Cichon, G., Ishdorj, T.O.: Implementation of RSA cryptographic algorithm using SN P systems based on HP/LP neurons. *Journal of Membrane Computing* pp. 1–13 (2021)
15. Gheorghe, M., Lefticaru, R., Konur, S., Niculescu, I.M., Adorna, H.N.: Spiking neural P systems: matrix representation and formal verification. *Journal of Membrane Computing* **3**(2), 133–148 (2021)
16. Ghosh-Dastidar, S., Adeli, H.: Third generation neural networks: Spiking neural networks. In: *Advances in Computational Intelligence*, pp. 167–178. Springer (2009)
17. Guo, P., Quan, C., Ye, L.: UPSimulator: A general P system simulator. *Knowledge-Based Systems* **170**, 20 – 25 (2019). <https://doi.org/https://doi.org/10.1016/j.knosys.2019.01.013>, <http://www.sciencedirect.com/science/article/pii/S0950705119300115>
18. Gutiérrez Naranjo, M.Á., Leporati, A.: Performing arithmetic operations with spiking neural P systems. *Proceedings of the Seventh Brainstorming Week on Membrane Computing*, vol. I, 181-198. Sevilla, ETS de Ingeniería Informática, 2-6 de Febrero, 2009 (2009)
19. Heiner, M., Herajy, M., Liu, F., Rohr, C., Schwarick, M.: Snoopy - A Unifying Petri Net Tool. In: Haddad, S., Pomello, L. (eds.) *Application and Theory of Petri Nets*. pp. 398–407. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
20. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems. *Fundamenta informaticae* **71**(2, 3), 279–308 (2006)
21. Ionescu, M., Sburlan, D.: Some applications of spiking neural P systems. *Computing and Informatics* **27**(3+), 515–528 (2012)
22. Leporati, A., Mauri, G., Zandron, C., Păun, G., Pérez-Jiménez, M.J.: Uniform solutions to SAT and Subset Sum by spiking neural P systems. *Natural computing* **8**(4), 681 (2009)
23. Macías-Ramos, L.F., Pérez-Hurtado, I., García-Quismondo, M., Valencia-Cabrera, L., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A p-lingua based simulator for spiking neural P systems. In: *International Conference on Membrane Computing*. pp. 257–281. Springer (2011)
24. OpenJS Foundation: *Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS*. <https://www.electronjs.org/> (2021)
25. Păun, G.: Spiking neural P systems. A tutorial. *Bull. Eur. Assoc. Theor. Comput. Sci.* **91**, 145–159 (2007)
26. Păun, G.: From cells to (silicon) computers, and back. In: *New computational paradigms*, pp. 343–371. Springer (2008)
27. Păun, G., Pérez-Jiménez, M.J.: Membrane computing: brief introduction, recent results and applications. *Biosystems* **85**(1), 11–22 (2006)
28. Prometheus Peter Lazo, Francis George Cabarle, H.A., Yap, J.M.: A return to stochasticity and probability in spiking neural P systems. *Journal of Membrane Computing* **3**(2), 149–161 (2021)
29. PĂrez-Hurtado, I., Valencia-Cabrera, L., PĂrez-JimĂñez, M.J., Colomer, M.A., Riscos-NĂñez, A.: MeCoSim: A general purpose software tool for simulating biological phenomena by means of P systems. In: *2010 IEEE Fifth International*

- Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA). pp. 637–643 (2010). <https://doi.org/10.1109/BICTA.2010.5645199>
30. PáĀun, G.: Computing with Membranes. *Journal of Computer and System Sciences* **61**(1), 108 – 143 (2000). <https://doi.org/https://doi.org/10.1006/jcss.1999.1693>, <http://www.sciencedirect.com/science/article/pii/S0022000099916938>
 31. Research Group on Natural Computing: *The P-Lingua Website*. http://www.p-lingua.org/wiki/index.php/Main_Page (2005)
 32. Roger, S.: *JFLAP*. <http://www.jflap.org/> (2009)
 33. Schwarze, M.: Web-based Petri net animation (in German). Diploma thesis, FH Lausitz, Dep. of CS (November 2009)
 34. Valdez, A.A.: (2020), <https://secretmapper.github.io/Snapse/>
 35. Valencia-Cabrera, L., Pérez-Hurtado, I., Martínez-del Amor, M.Á.: Simulation challenges in membrane computing. *Journal of Membrane Computing* pp. 1–11 (2020)
 36. Verlan, S., Freund, R., Alhazov, A., Ivanov, S., Pan, L.: A formal framework for spiking neural P systems. *Journal of Membrane Computing* **2**(4), 355–368 (2020)
 37. Zeng, X., Adorna, H., Martínez-del Amor, M.Á., Pan, L., Pérez-Jiménez, M.J.: Matrix representation of spiking neural P systems. In: *International conference on membrane computing*. pp. 377–391. Springer (2010)

Parallel Computing With Water

Alec Henderson¹, Radu Nicolescu¹, Michael J. Dinneen¹, TN Chan²,
Hendrik Happe³, and Thomas Hinze³

¹ School of Computer Science,

University of Auckland, Auckland, New Zealand

² Compucon New Zealand, Auckland, New Zealand

³ Department of Bioinformatics,
Friedrich Schiller University of Jena, Jena, Germany

Abstract. We further the work on a recently proposed membrane computing model which utilises decentralised water tanks interconnected by pipes with water flow controlled by valves. We demonstrate that such systems can construct ‘efficiently’: 1) A programmable sequential, random-access machine (RAM), which we then extend to construct: 2) a programmable exclusive read exclusive write (EREW) parallel random-access machine (PRAM).

Keywords: Water-based computing · Membrane systems · RAM machine · Fluidics · PRAM

1 Introduction

P systems, first proposed by Gheorghe Pun in [1] are a parallel and distributed model of computation inspired by biological membranes. P systems are decentralised and typically evolve based on the content of a membrane. A water-based system was proposed by [2] which contained many properties similar to those of membrane computing and was therefore described as a P system. The system was decentralised, and the content of each tank would evolve in parallel, similar to how P systems evolve.

In this work, we demonstrate that the water system proposed in [2] and extended in [3] can construct a PRAM machine. This model has been shown to be Turing complete, inherently proving that a RAM model can be built. However, this construction does not consider the complexity of such a construction. In this work, we demonstrate that both a programmable RAM machine and a programmable PRAM machine can be constructed ‘efficiently’.

2 Background

In this section, we start by defining saturated arithmetic. We then define a RAM machine and show how the Euclidean algorithm can be implemented in RAM

instruction code. Ending the section with the definition of the water computing model, which we utilise throughout the remainder of the paper.

2.1 Saturated arithmetic

As discussed in [4] saturation arithmetic restricts operations to a fixed range. If a result exceeds the upper or lower limit, the result is that limit. For example, if the range was $[0,10]$ then, $5 * 5 = 10$ and $10 - 20 = 0$. If the upper and lower limits are $+\infty$ and $-\infty$, then it is standard arithmetic. For simplicity, we denote saturating addition as \oplus and saturating subtraction as \ominus . Throughout the remainder of the text, without loss of generality, we assume all saturated arithmetic except for control tanks to have lower bound 0, and upper bound $+\infty$ where, control tanks have 0 and 1.

2.2 RAM

There have been many models of computation proposed, with the 'sequential machines' typically being the deterministic Turing machine and the random access memory (RAM) machine [5]. In this work, we focus on the RAM machine as they usually have more practical uses than the traditional Turing model.

A RAM machine consists of a finite program of m lines and a sequence of registers r_1, r_2, \dots, r_n (perhaps an infinite sequence). The program of the RAM will consist of a sequence of operation codes and parameters. Based on the definition in [6], a RAM has the following operations:

1. $r_i \leftarrow C$: assign a constant value C to register i .
2. $r_i \leftarrow r_j \oplus r_k$: add the value of two registers j and k and assign to register i .
3. $r_i \leftarrow r_j \ominus r_k$: subtract from register j the value stored in k and assign to register i .
4. $r_i \leftarrow r_{r_j}$: get the value y from register j , then get the value from register y and assign to register i
5. $r_{r_i} \leftarrow r_j$: get the value y from register j , then get the value x from register i and assign y to register x .
6. TRA m $r_i > 0$: go to program line m (control transferred to line m of the program) if r_i greater than 0, otherwise go to the next line.

Where we use the item number as the op code as seen in Table 1.

We assume that the output will be the values stored in the registers once the program has halted. A program halts when it has gone to a line number in the program which is not defined. For example, consider the following Euclidean algorithm pseudocode for positive integers:

Table 1. Operations and there corresponding opcodes.

Operation	Opcode
$r_i \leftarrow C$	1 $i C$
$r_i \leftarrow r_j \oplus r_k$	2 $i j k$
$r_i \leftarrow r_j \ominus r_k$	3 $i j k$
$r_i \leftarrow r_{r_j}$	4 $i j$
$r_{r_i} \leftarrow r_j$	5 $i j$
TRA $m r_i > 0$	6 $m i$

```

1 function gcd( $a, b$ )
2   while ( $a \neq b$ )
3     if ( $a > b$ ) then
4        $a \leftarrow a \ominus b$ 
5     else
6        $b \leftarrow b \ominus a$ 
7   return  $a$ 

```

This pseudocode translates into the RAM code presented in Table 2.

Table 2. Euclidean algorithm implemented for RAM machine.

Line number	Operation	Opcode	Comment
1	$r_3 \leftarrow r_1 \ominus r_2$	3 3 1 2	
2	TRA 6 $r_3 > 0$	6 6 3	Go to line 6 if $a > b$
3	$r_3 \leftarrow r_2 \ominus r_1$	3 3 2 1	
4	TRA 8 $r_3 > 0$	6 8 3	Go to line 8 if $b > a$
5	TRA 10 $r_1 > 0$	6 10 1	Halt $a = b$ result is in r_1
6	$r_1 \leftarrow r_1 \ominus r_2$	3 1 1 2	$a \leftarrow a \ominus b$
7	TRA 1 $r_1 > 0$	6 1 1	Start the loop again
8	$r_2 \leftarrow r_2 \ominus r_1$	3 2 2 1	$b \leftarrow b \ominus a$
9	TRA 1 $r_2 > 0$	6 2 1	Start the loop again

We note that our RAM model has a fixed size program with m lines. This is able to implement RAM programs of length less or equal m . Noting that the machine halts when it gets to line $m + 1$. Hence, if a program had less than m lines an additional line could be added to jump to line $m + 1$.

As the RAM model we have discussed is inherently sequential one can extend it to be parallel. A parallel RAM machine (PRAM) is one of the most well known parallel computing models. As defined in [7, 8] a PRAM consists of a set of processors p_1, p_2, \dots and a set of shared registers r_1, r_2, \dots . Each processor can be viewed as an individual RAM with its own program and sequence of registers.

Processors can only communicate via the shared memory via read or write operations. The processors execute in a synchronous manner, with each step taking the same amount of time. Typically there are three models of PRAM discussed in the literature to model the read and write of the shared memory they are:

- Exclusive read exclusive write (EREW): only one processor can read or write to each shared register at a time.
- Concurrent read exclusive write (CREW): Any number of processors can read from the same shared register at the same time but only one can write to a each shared memory location.
- Concurrent read concurrent write (CRCW): Any number of processors can read and write the same shared register at the same time.

The CRCW PRAM is typically further defined based on different ways of handling concurrent writes these include [9]:

- Collision: A collision symbol is written. This does not give details about which processors caused the collision or what they were attempting to write.
- Common: Successful write only if all processors writing to the same location are writing the same value.
- Arbitrary: Only one arbitrary attempt is successful but, this choice will result in the same final result.
- Priority: The processor with the lowest IDs write is successful.

2.3 Water model

The water-based system originally proposed in [2] and extended in [3] works by having a set of tanks interconnected with pipes which flow is controlled by valves. This model contains no central control as well as no loops (water flows in one direction). Formally the model is defined as:

$$\Pi = (T, T', F, E, R, L, C, V, S, P)$$

With its components:

- T finite set of tank identifiers.
- $T' \subset T$ finite set of control tank identifiers.
- $F \subset T'$ set of control tanks that when full indicate termination of the system.
- $E \in T \setminus T'$ the unique infinite sink of the system.
- $R \in T \setminus T'$ the unique infinite source in the system.
- $L : T \rightarrow \mathbb{N}_+$ The level at which the tanks are built, the lower the number the conceptually higher the tank. Water can only flow from a tank with a lower number to one with a higher number. $L(R) = 0$ and $L(E) = \infty$.

- $C : T \rightarrow \mathbb{N}_+ \cup \{\infty\}$ capacity of the tanks. Where we assume that value tanks are able to be unbounded. Of course, for practical cases, all value tanks will need to have finite capacity. Control tanks all have capacity 1 (they act as Boolean values).
- V finite set of valve identifiers.
- $S : V \rightarrow (T = \mathbb{N}_+ \cup T \neq \mathbb{N}_+)$ An expression from a valve identifier to check whether or not a tank has a certain volume.
- $P \subset T \times T \times \mathcal{P}(V)$ ($\mathcal{P}(V)$ denotes the power set over V) finite set of pipes where water flows from the first element to the second. A pipe (i, j, v) must have $L(i) < L(j)$, meaning water only flows in one direction ('down').

The system evolves in discrete time steps where, at each step one unit of water flows down each pipe if all valves on the pipe are open and water is available. If at any point a tank has less water than the number of pipes with all there valves open, then the result will be dependant on the assumed run-time (physical implementation). Reasonable run times include:

- Non-deterministically decide which pipes get a unit of water.
- A priority based ordering of pipes.

As this does not occur throughout this paper we do not discuss this further.

Typically, this system is modelled by either a set of equations or diagrams. Control tanks as presented in [3] are used to denote when each functions input/output is ready. That is to say for each input/output the value is ready to be used once its corresponding control tank is full. This design allows the water-based functions to be modularised for a more in-depth discussion see [3].

2.4 Examples

For self-containment, we present four examples of water-based functions, for a more detailed description of these functions see [3]. These functions are then used in the construction of a RAM. We display tanks as open rectangles with pipes going from one tank to the next as a line between them. If a pipe goes to the infinite sink, we denote this by an arrow at the end of the line. If a pipe comes from the infinite source, we have the line start with a small rectangle. Valves are displayed as lines crossing a pipe with the Boolean condition next to it.

- Increment $f(x) = x \oplus 1$ (cf. Figure 1): The increment operator drains the two input tanks (value and control) into the result. Once the inputs are empty the output control tank is filled.
- Addition $f(x, y) = x \oplus y$ (cf. Figure 2): The addition operator works similar to the increment operator but, instead of the control tank draining to the

result the value y does. Once all inputs are empty the output control tank is filled.

- Subtraction $f(x, y) = x \ominus y$ (cf. Figure 3): The subtraction operator drains from both x and y into the infinite sink until y is empty. Once y is empty any remaining water drains from x into the result tank. Once all inputs are empty the output control tank is filled.

It is important to note that when the Increment, Addition, and the Subtraction operators are executed the inputs are drained/lost. This means that if we want to execute a function and keep the inputs, we need an in-place copy:

- Inplace copy $f(x) = x$ (cf. Figure 4): Input tank x drains into result tank x_1 . Whilst x drains tank a fills from the infinite source. Once x is empty a stops filling and the control tank is drained into the infinite sink. Once the control tank is drained, the content of a (which stores the value of the original input) is drained into the original input tank x . Once a is empty again, the output control is filled.

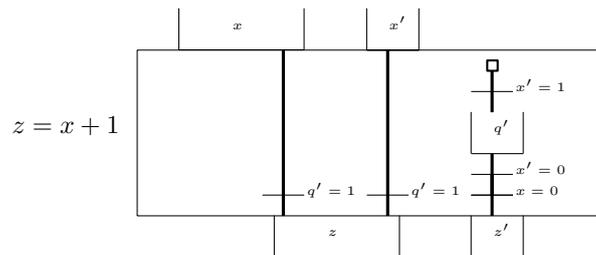


Fig. 1. A diagram representing the increment operator function $z = x \oplus 1$.

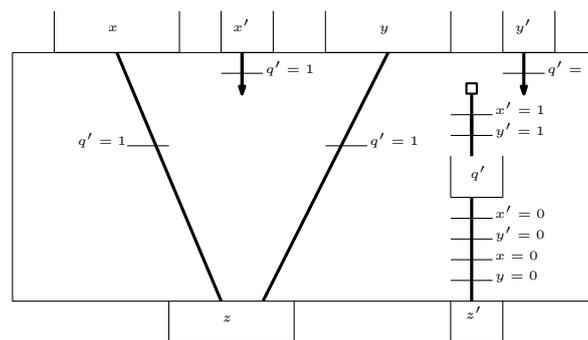


Fig. 2. A diagram representing the controlled saturating addition operator $z = x \oplus y$.

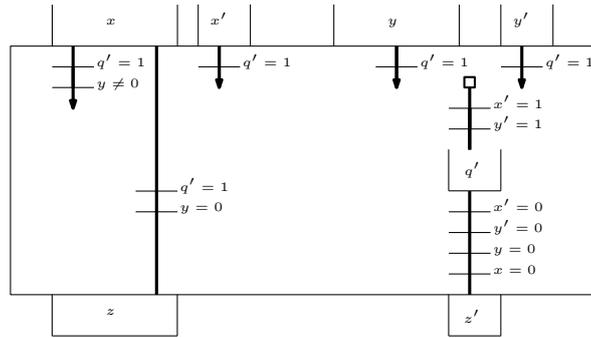


Fig. 3. A diagram representing the controlled saturating subtraction operator $z = x \ominus y$.

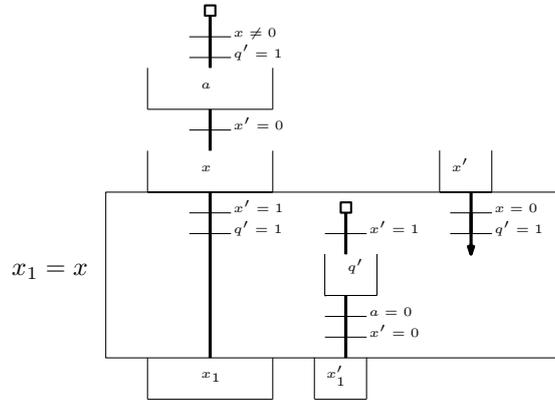


Fig. 4. A diagram representing the inplace copy function $i(x) = x$.

3 Constructing a programmable RAM machine

Constructing a programmable RAM machine using water can be broken into phases. The first phase is to read the line number that we are up to. Using the op code for that line, run a function which executes that function. After that has been done, check whether the line number exceeds the program line count; if it does then the RAM halts, otherwise it will do the process again. To make it easier to follow, we shall also break up the construction into modules which can then be pieced together to form the entire RAM.

3.1 Execute line L of the program

Here we assume that the program being executed will be stored in water tanks $p_{1,1}, p_{1,2}, p_{1,3}, p_{1,4}, p_{2,1}, p_{2,2}, p_{2,3}, p_{2,4}, \dots, p_{m,1}, p_{m,2}, p_{m,3}, p_{m,4}$. Where $p_{i,j}$ denotes the i th line with parameter j , noting that we assume that each line of code will have one op code followed by three parameters; the third tank contents will be ignored for operations of two parameters.

To start the program at line L and continue executing until we reach line $m + 1$ we utilise the tank system presented in Figure 6. For brevity we have presented an arbitrary program line $p_{o,j}$ but this can be expanded for all program lines by changing the valve for $p_{i,k}$ to $L = i$ for the i th line. For example, the tanks presented in Figure 6 for the Euclidean algorithm would initially contain the volumes presented in Table 3.

Noting that to run a function we use the tank system presented in Figure 7. The tank system presented in Figure 7 is not repeated, only one instance exists for a RAM machine. The line inputs $p_{i,j}$ all drain into the inputs p_1, p_2, p_3, p_4 . Using traditional programming the operation can be explained to be the function **RAM** presented in Figure 5.

1	function RAM (p, L): // $p = [p_{1,1}, p_{1,2}, p_{1,3}, p_{1,4}, p_{2,1}, p_{2,2}, \dots, p_{m,4}]$
2	$p_1 \leftarrow p[4 * (L - 1)]; p_2 \leftarrow p[4 * (L - 1) + 1]$
3	$p_3 \leftarrow p[4 * (L - 1) + 2]; p_4 \leftarrow p[4 * (L - 1) + 3]$
4	$s \leftarrow \mathbf{run_op}(p_1, p_2, p_3, p_4, L)$
5	if $s \neq m + 1$:
6	RAM (p, s)
7	halt

Fig. 5. Code to describe the outer loop of our RAM system.

Table 3. Initial volumes of water for tanks presented in Figure 6 for Euclidean algorithm. Where we denote a tank i stores volume j by $i(j)$

i	$p_{i,1}$	$p'_{i,1}$	$p_{i,2}$	$p'_{i,2}$	$p_{i,3}$	$p'_{i,3}$	$p_{i,4}$	$p'_{i,4}$
1	3	0	3	1	0	0	2	0
2	6	0	6	0	3	0	0	0
3	3	0	3	0	2	0	1	0
4	6	0	8	0	3	0	0	0
5	6	0	10	0	1	0	0	0
6	3	0	1	0	1	0	2	0
7	6	0	1	0	1	0	0	0
8	3	0	2	0	2	0	1	0
9	6	0	2	0	1	0	0	0

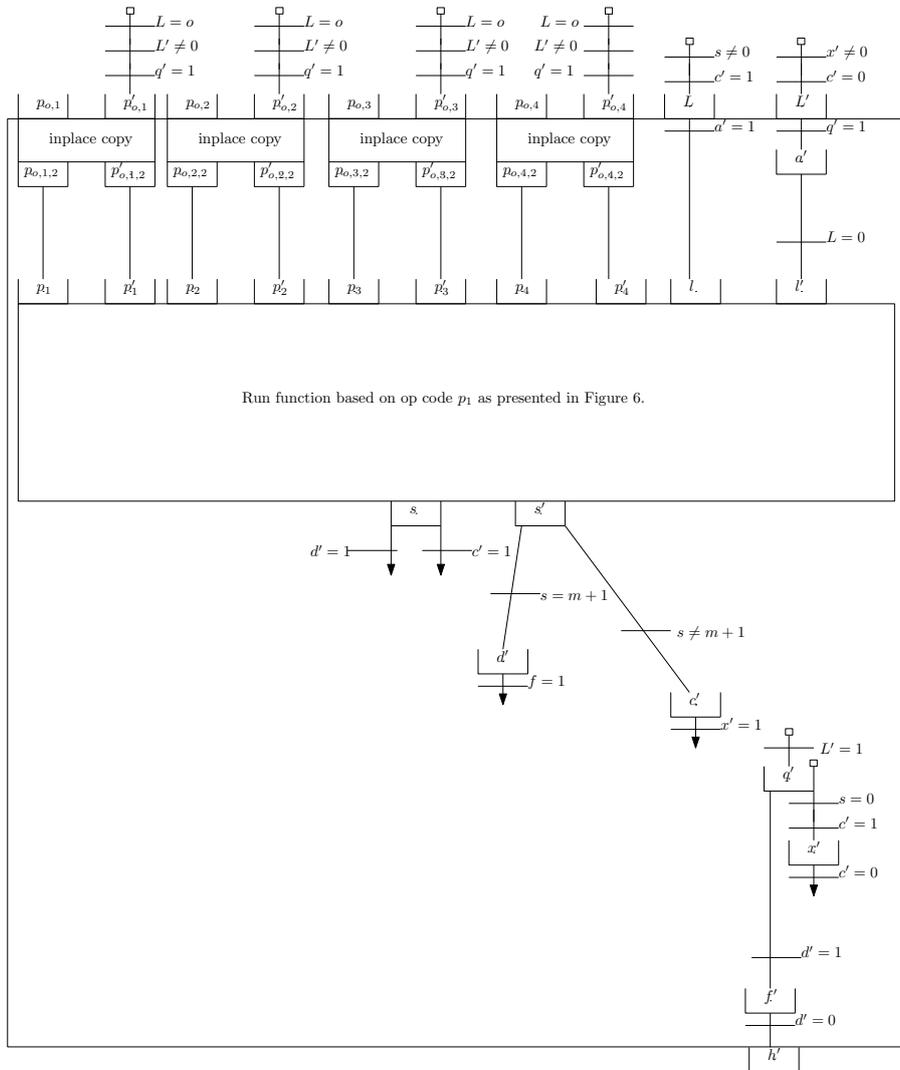


Fig. 6. A diagram representing the an outer loop of a RAM. Where the input L is the line to start the programs execution (typically line 1). After the operation has completed tank s will contain the next line to execute. Tank s will then be drained and L filled with that contents. Once the program has ended tank h' will be full. Noting that we have shown only for one arbitrary program line o .

3.2 Executing an operation code

Once the line number is read and operation number selected it is passed into the inner function presented in Figure 7. This will utilise the op code stored in p_1 and decide which operation to execute. For simplicity, we have presented an arbitrary operation X , this can be expanded for all operations by taking $X = 1, 2, 3, 4, 5, 6$. This operation can alternatively be summarised by the function `run_op` presented in Figure 8.

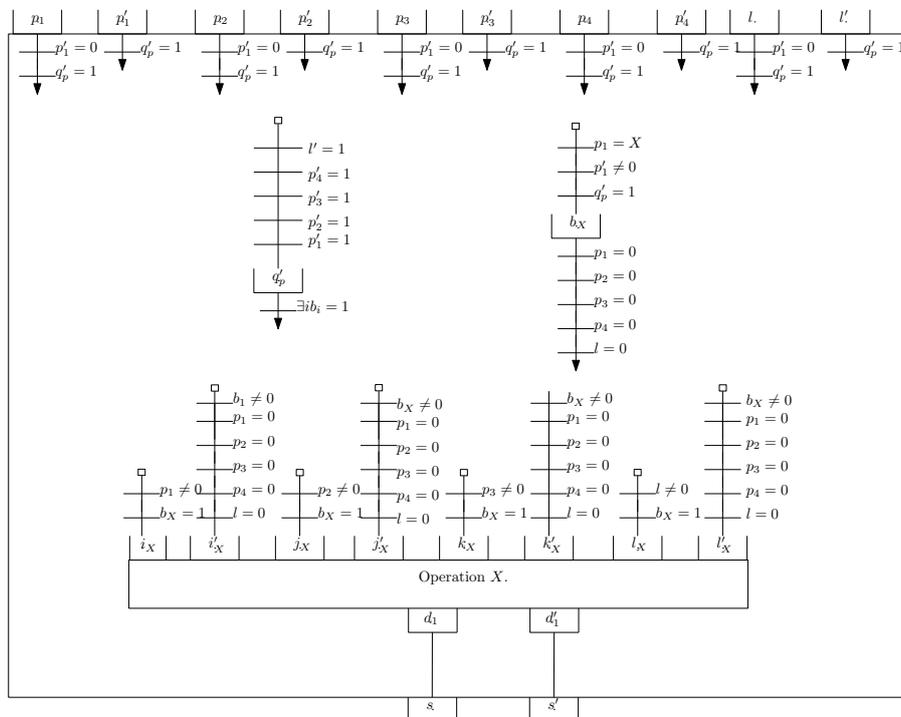


Fig. 7. A diagram representing which of the 6 functions will be executed. Where for simplicity we have shown for an arbitrary opcode X where $X \in \{1, 2, 3, 4, 5, 6\}$. Selected operations are presented: Operation 1 in Figure 10, Operation 2 in Figure 12, Operation 4 in Figure 13 and Operation 6 in Figure 14. Noting that $\exists i b_i = 1$ is a shorthand to describe six pipes from q'_p each with one valve as shown in Figure 9.

As described earlier, a RAM can be constructed using six basic operations. However, to simplify these operations we utilise the fact that these operations can be viewed as a sequence of read and writes from registers.

```

1 function run_op( $p_1, p_2, p_3, p_4, l$ ):
2   if  $p_1 = 1$ 
3     return const( $p_2, p_3, l$ )
4   else if  $p_1 = 2$ 
5     return add( $p_2, p_3, p_4, l$ )
6   else if  $p_1 = 3$ 
7     return sub( $p_2, p_3, p_4, l$ )
8   else if  $p_1 = 4$ 
9     return indr( $p_2, p_3, l$ )
10  else if  $p_1 = 5$ 
11    return indw( $p_2, p_3, l$ )
12  else if  $p_1 = 6$ 
13    return tra( $p_2, p_3, l$ )

```

Fig. 8. Code to describe the outer loop of our RAM system.

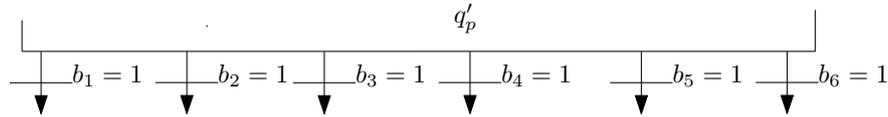


Fig. 9. Diagram to show the expanded version of the shorthand $\exists i b_i = 1$ for tank q'_p .

Utilising read and write operators, we construct the base operations. For brevity, we only show functions 1, 2, 4, and 6; functions 3 and 5 can be straightforwardly derived from functions 2 and 4, respectively.

Although operation one is near identical to the write operation, it is important to note that the operations need to also return the new line number after completing, hence we use the increment function. Operation 1 can be viewed in Figure 10. A programmatic view of operation one is presented in Figure 11

Operation 2 can be viewed in Figure 12 where we have left out the line number increment. The line increment can be done just as we did in the first operation.

Operation 4 can be viewed in Figure 13.

Operation 6 can be viewed in Figure 14.

We implement the read and write functions in Figure 15 and Figure 16 respectively. We note that these two functions are reading and writing to the same set of registers, so the registers r_1, \dots, r_n are shared between the figures.

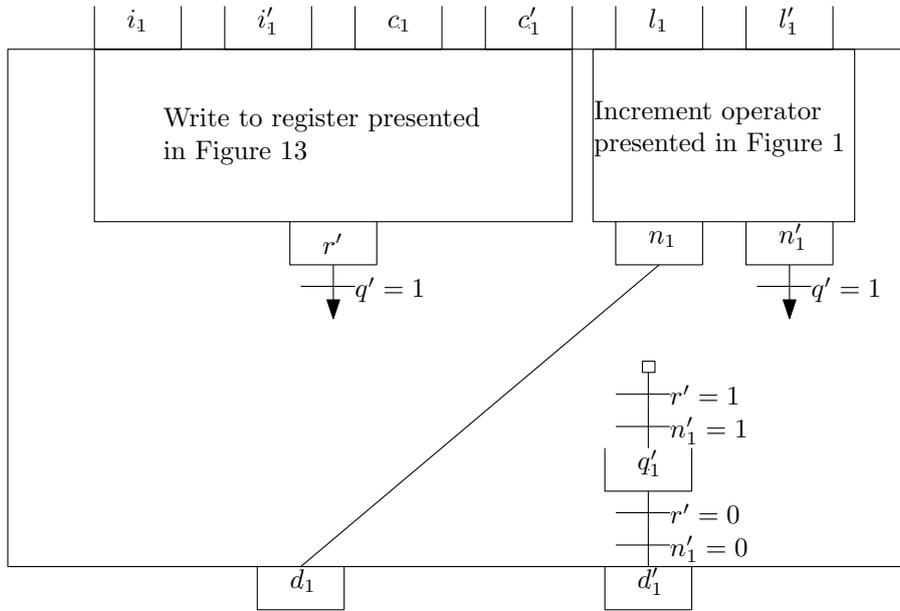


Fig. 10. A diagram representing operation 1: $r_i \leftarrow C$. Returns $l \oplus 1$ where l is the program line number.

```

function const( $i$ ,  $c$ ,  $l$ ):
     $r_i \leftarrow c$ 
    return  $l + 1$ 
    
```

Fig. 11. Code to describe the constant function.

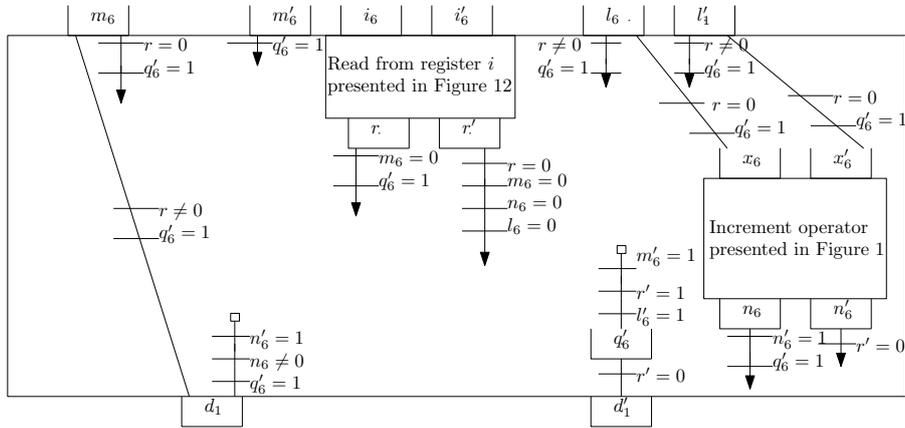


Fig. 14. A diagram representing operation 6: TRA $m r_i > 0$.

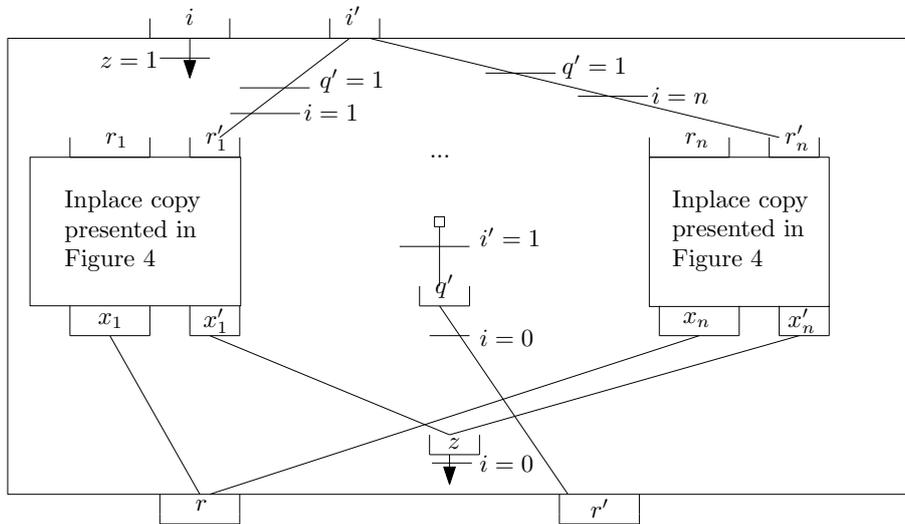


Fig. 15. A diagram representing read from register i .

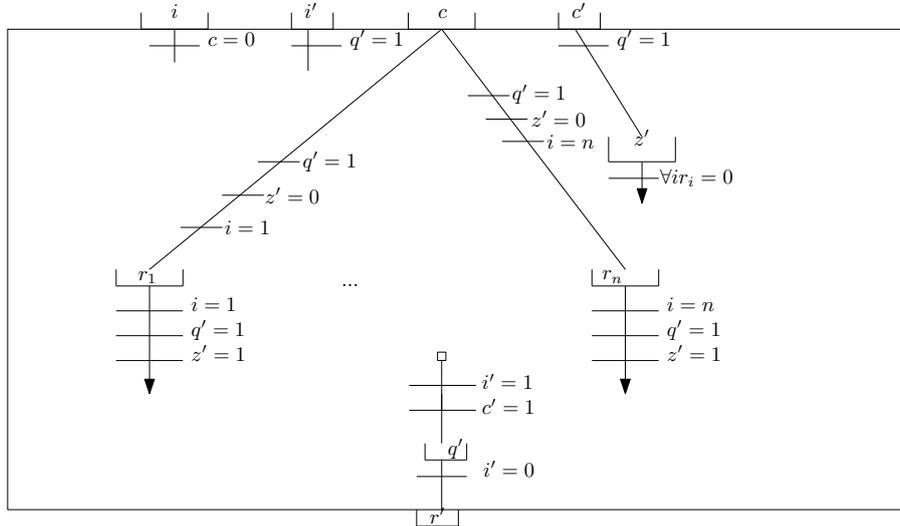


Fig. 16. A diagram representing write to register i the value $c : r_i \leftarrow c$. Where the \forall is a shorthand to mean n valves with each valve being $r_i = 0$.

4 Extending to PRAM

Extending to a EREW PRAM we note the following:

1. The only way for processors to communicate is via read/writes to the shared memory.
2. The output of the PRAM will be the content stored on the shared memory once all processors halt.
3. At each time step if any two processors try to read and or write from the same shared memory location the result will be undefined.
4. Each operation will be run synchronously, i.e. at every time step each processor fully complete one program line.
5. We assume that each processor has its own finite program and a set of local registers.

In this section, we extend our previously constructed RAM model to be a single processor. We note currently our RAM model defined earlier does not have operations to read/write to shared memory. However, more importantly it does not satisfy the synchronous behaviour required for operations. In this section, we first describe adding shared reads and writes to our previously discussed model. We then describe adding a synchronous lock to ensure each processor evaluates each line of its program at the same time.

Denoting shared registers ρ we define the following additional operators:

- $r_i \leftarrow \rho_j$: Read from shared memory and store in a local register.
- $\rho_i \leftarrow r_i$: Read from a local register and write to a shared register.

To define these new functions, we note that they are a simpler version of the indirect operation presented in Figure 13. With only one read but, instead of the read from a local register r we define a read from shared memory ρ .

We extend the reading and writing of the local registers presented in Figure 15 and Figure 16. To extend it we have each processor $1, 2, \dots, p$ have its own input, output, and control tanks for the shared read and writes. The diagram presented in Figure 17 shows how an arbitrary register X reads from shared memory.

The extension to the read can similarly be used for the write which we omit for brevity. It is important to note that for an EREW PRAM, at any one time step, only one of the processors may read or write to a register. If multiple reads, writes, or a combination are done on the same shared memory the computation will have an unexpected result.

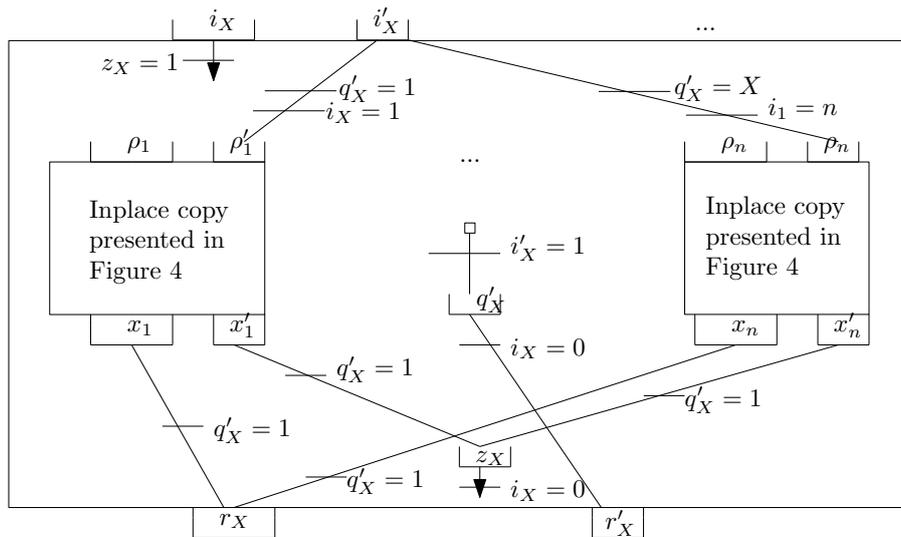


Fig. 17. Parallel read for arbitrary processor X .

To ensure that each line of code is run synchronously each processor must wait until all other processors have completed their current line. To achieve this, we alter the outer program presented in Figure 6 to that presented in Figure 18 and Figure 19.

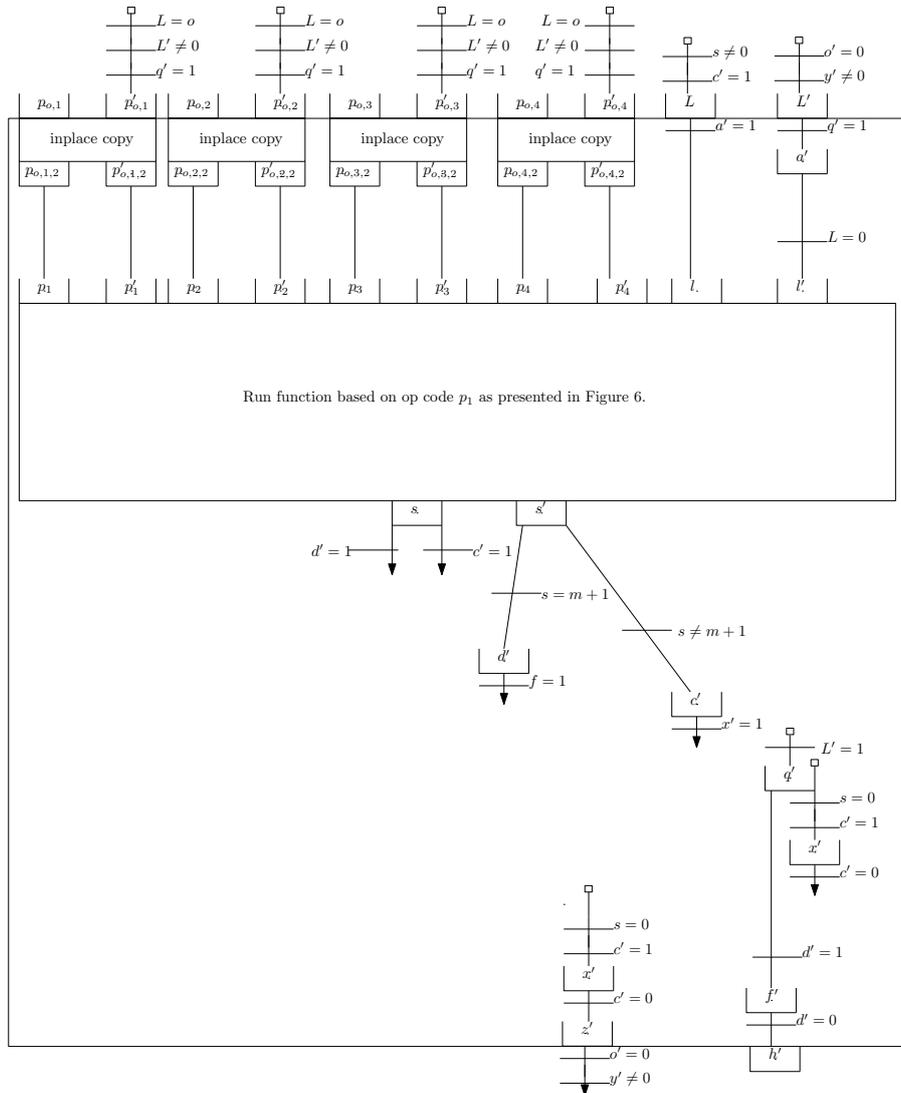


Fig. 18. Parallel outer program that will only loop based on the synchronisation presented in Figure 19

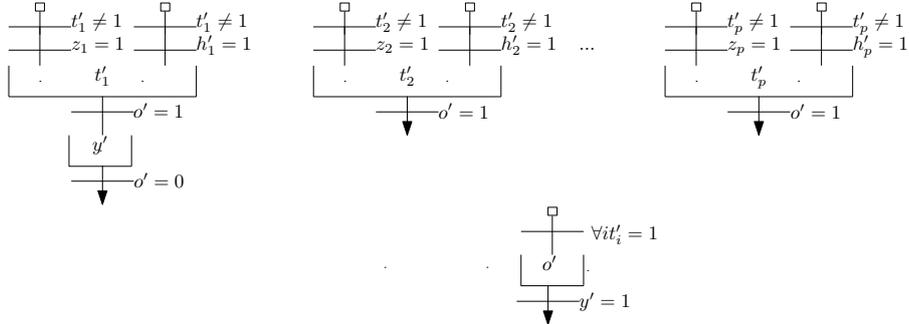


Fig. 19. The synchronisation scheme which ensures that only after all other processors have done an operation may they go to the next line. Noting that we use the shorthand $\forall it'_i = 1$ which expands to the valves presented in Figure 20

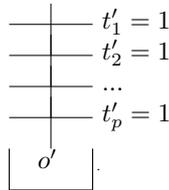


Fig. 20. Expanded version of $\forall it'_i = 1$.

5 Conclusions and future work

Using the previously defined water system [3] we have constructed: 1) a programmable RAM machine and 2) a programmable EREW PRAM machine. This demonstrates the non centrally controlled model is inherently parallel and can model one of the most well known parallel computing models. However, we have only modelled the least powerful of the PRAM models, leaving future work to look at allowing concurrent reads and/or writes. Of course with this modelling many well-known results of traditional parallel computing transfer to this model.

Another open problem is the cost-based minimisation of the number of valves and pipes. Future work could look at physical realisations of this system and determining which is more ‘expensive’ based on a well defined measurement of cost. Physical realisations of the system could also have many other benefits such as for education.

References

1. Gh. Păun, “Computing with membranes,” *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.

2. T. Hinze, H. Happe, A. Henderson, and R. Nicolescu, “Membrane computing with water,” *Journal of Membrane Computing*, vol. 2, no. 2, pp. 121–136, 2020.
3. A. Henderson, R. Nicolescu, M. J. Dinneen, T. Chan, H. Happe, and T. Hinze, “Turing completeness of water computing,” report CDMTCS-554, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand, July 2021.
4. F. Zappa and S. Esculapio, *Microcontrollers. Hardware and Firmware for 8-bit and 32-bit devices*. LIGHTNING SOURCE Incorporated, 2017.
5. I. Parberry, “Parallel speedup of sequential machines: A defense of parallel computation thesis,” *SIGACT News*, vol. 18, p. 5467, Mar. 1986.
6. S. A. Cook and R. A. Reckhow, “Time bounded random access machines,” *Journal of Computer and System Sciences*, vol. 7, no. 4, pp. 354–375, 1973.
7. E. Gafni, J. Naor, and P. Ragde, “On separating the EREW and CREW PRAM models,” *Theoretical Computer Science*, vol. 68, no. 3, pp. 343–346, 1989.
8. P. B. Gibbons, “A more practical PRAM model,” in *Proceedings of the first annual ACM symposium on Parallel algorithms and architectures*, pp. 158–168, 1989.
9. F. E. Fich, P. Ragde, and A. Wigderson, “Relations between concurrent-write models of parallel computation,” *SIAM Journal on Computing*, vol. 17, no. 3, pp. 606–627, 1988.

Sparse matrix representation of Spiking Neural P systems on GPUs

Javier Hernández-Tello¹, Miguel Á. Martínez-del-Amor¹,
David Orellana-Martín¹, and Francis George C. Cabarle²

¹ Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
jhtello@us.es, mdelamor@us.es, dorellana@us.es

² Algorithms and Complexity Laboratory
Department of Computer Science
University of the Philippines Diliman
1101 Quezon City, Philippines;
fccabarle@up.edu.ph

Abstract. Spiking Neural P (SNP) systems can be simulated by means of a matrix and vector representation. In fact, this is the key of parallel simulators for SNP systems, such as cuSNP. However, since matrices can be sparse, the performance of the simulation can be negatively affected. Recently, key concepts for compressing sparse matrices for SNP systems have been published. In this short paper, we show our first steps towards an implementation on GPUs using CUDA and its preliminary results.

Keywords: Membrane Computing · Sparse matrices · Spiking Neural P systems · Parallel Simulation · GPU computing

1 Introduction

Most sequential simulators of Spiking Neural P (SNP) systems [5] make use of *ad-hoc* representations, specifically defined for a certain variant [11]. The parallel simulation of SNP systems [13,8] have been performed by means of a matrix representation that was introduced in [12]. The GPU-based simulator for SNP systems called cuSNP implements a simulation algorithm that applies these vector-matrix operations [2,3,1]. However, this matrix representation can be sparse, i.e. having a majority of zero values. The main cause is that the directed graph of SNP systems are not usually fully connected.

Sparse vector-matrix operations are well known and natural in high performance computing solutions [4]. For this reason, a recent published work in [9] introduces the a compressed matrix representation for several SNP system variants: standard, with budding and division, and with plasticity. This previous work demonstrates that for SNP systems with dynamic structures, the plasticity variant fits better with the compressed sparse matrix representation.

However, these new representations and algorithms have not been implemented on parallel technologies such as GPUs [7]. GPUs are parallel devices with thousands of parallel cores that has been used to simulate P systems for years [8,13]. In this short paper, we show our preliminary results, analyse them and settle the path for future work. We use the results to stand out that there is room for improvements and more research, since compressed representation of SNP systems will help to better deploy models.

The paper is structured as follows: Section 2 gives a short description of the design of compressed matrix representation of SNP systems; Section 3 summarizes the implementation details of the new simulator based on compressed matrices; Section 4 briefly shows the preliminary results of the simulator; Section 5 discusses conclusions and future work.

2 SpMV operations for SNP systems

We assume the reader is familiar with Spiking Neural P (SNP) systems. More information such as the syntax and semantics of SNP systems, including regular expressions, what constitutes a valid spiking vector can be found in [5,6]. We will also briefly describe the matrix representation in order to enable a linear-algebra-based simulator, but more details can be found in [12,9].

Although the baseline matrix representation only involves SNP systems without delays and static structure, many extensions have followed such as for enabling delays or supporting non-determinism [2,3]. The matrix representation employed in parallel simulators for a SNP systems without delays is as follows. For a SNP system of degree (n, m) (n rules and m neurons), we define the following 3 data structures:

- *Configuration vector*: C_k is the vector containing all spikes in every neuron on the k^{th} computation step/time, where C_0 denotes the initial configuration. It contains m elements.
- *Spiking vector*: S_k says if a rule is going to fire at the transition step k (having value 1) or not (having value 0). Given the non-determinism nature of SNP systems, it would be possible to have a set of valid spiking vectors. However, for the computation of the next configuration vector, only a spiking vector is used. It contains n elements.
- *Spiking transition matrix*: M_π is a matrix of $n \cdot m$, where rows represent rules and columns represent neurons, and each element says how each rule affects each neuron: consuming (negative value) or producing (positive value) and how many spikes, or 0 otherwise.

Hence, to compute the transition k , it is enough to select a spiking vector S_k and calculate: $C_k = S_k \cdot M_\pi + C_{k-1}$.

M_π can be compressed because it might be very sparse. This can reduce the memory footprint of simulators and accelerate the simulation. In [9], three versions are proposed:

- *Sparse*: this implementation has no compression, and it is defined as above.
- *ELL*: this implementation is based on the ELL compression of sparse matrices [7,4]. It first calculates the transpose of the matrix (this improves the data coalescing in GPUs), and then:
 - The number of rows is the maximum amount of non-zero values in a row of M_{II} , denoted by z . It can be shown that $z = \text{MaxOutDegree} + 1$, where MaxOutDegree is the maximum output degree in the neurons of the SNP system. In general, a column devoted for a rule $E/a^c \rightarrow a^p$ contains values $+p$ for every neuron connected with the source neuron (i.e. where it belongs to), and a value $-c$ for consuming the spikes in that source neuron.
 - The values inside columns can be sorted, so that the consumption of spikes ($-c$ values) are placed at the first row. In this way, all threads can start with the same task, consuming spikes. Moreover, the loop along the columns can be ended prematurely, once 0 values are encountered.
 - Every position is a pair where the first element is a neuron label, and the second is the amount of spikes ($+p$ or $-c$).
- *Optimized*: the transition matrix can be split in order to avoid, for each rule, replicating the generation of spikes ($+p$) for all synapses. Thus, the synapses can be stored separately from the rule information, as follows:
 - *Rule information*. By using a CSR-like format [7,4], rules of the form $E/a^c \rightarrow a^p$ (also forgetting rules are included, assuming $p = 0$) can be represented by a couple of arrays that stores the regular expression associated to the rule, and the values c and p . An arrays of pointers (indexes) is employed to associate, for each neuron, the subset of rules that it has associated.
 - *Synapse matrix*, Sy_{π} . It has a column per neuron i , and a row for every neuron j such that $(i, j) \in Syn$ (there is a synapse). That is, every element of the matrix corresponds to a synapse or null. The later is necessary given that the number of rows equals to the maximum output degree in the neurons of the SNP system and padding is required.
 - *Spiking vector* is modified, containing only m positions, one per neuron, and stating which rule $0 \leq r \leq n$ is selected.

For more information and pseudocode of the algorithms, we refer the reader to [9].

3 Implementation

In this work, we present the first CUDA implementation of this compressed matrix representation. Next, we briefly provide some details on the initialization of the simulator and how to perform the computation. In this work we have also extended the ideas to support SNP systems with delays. The current state of the source code can be fetched at https://github.com/javihernant/sparse_snp.

3.1 Initialization

The following step has to be followed in order to initialize the simulator with the target model:

1. Set the initial configuration of the model; i.e. set the spikes each neuron will have in the beginning.
2. Set the rules for each neuron. In order to do this, the number of spikes to be consumed, the spikes to be produced, and a regular expression has to be provided. A delay may be as well introduced to the rule. The delay will consist of a number that represents how many transition steps will be executed until the producing spikes can reach the destination neurons. Current implementation provides the following types of regular expressions:
 - Type one (e^*). Rule can be activated at any possible condition.
 - Type two (e^+). Rule may activate if its corresponding neuron contains at least 1 spike.
 - Type three (e^n). Rule may activate if and only if its neuron contains exactly n spikes.
3. Add synapses of the SNP system. Once the initial configuration is set and all the rules of the model have been specified, the final step in the creation of the model is adding synapses. This information will be used to create the transition matrix, a vital part for the computation of the model. The simulator first initialize a sparse matrix, which is compressed afterwards for the corresponding implementation. This is done in this way to make the definition of the SNP system flexible for the user, otherwise the simulator would require parameters such as z beforehand to work with compressed representations only.

3.2 Simulation

The main loop of the simulation is carried out by the following components.

Spiking Vector In order to produce a computational step, the spiking vector has to be calculated. The spiking vector contains which rules have been selected to be fired. To calculate said vector, a CUDA kernel is implemented and it is launched with as many threads as neurons are in the model. The purpose of each thread will be to examine every rule of the corresponding neuron and determine if they can be enabled. Only one will get selected randomly for each neuron; however, in this preliminary version, it is a deterministic loop over the rules of each neuron, and the first applicable rule is chosen.

To check whether a rule can be enabled, the rule's regular expression need to comply with the current number of spikes in the neuron where the rule is contained. Thus, in case of a type one regular expression, the algorithm will check if the neuron contains at least 0 spikes. Or if a type two is used, then it will check that at least 1 spike are present. If type three is used, the neuron will be checked if it has exactly n spikes.

Stop criterion When a final configuration has been achieved, no further transitions will be made. To see if a configuration is final a stop criterion has been established, all rules have to be disabled (in the spiking vector) and all neurons open. All rules being disabled is not a guarantee of becoming active in later transitions. If there is any neuron closed, it can be assured that it contains a rule waiting to be fired.

Transition A Matrix-vector multiplication is calculated between the transition matrix and the spiking vector. This operation will be performed through a kernel that is launched with n threads, one for each neuron. The outcome is the next configuration vector.

As mentioned, to benefit from memory coalescing, the transition matrix is stored by columns. Each row now corresponds to a column of the original matrix. The algorithm will make sure only rules corresponding to open neurons are executed, the rest are ignored (even if enabled) until the delay has passed.

Update delays A delays vector has been implemented to keep count of the current delayed state of every neuron. Said vector contains n elements, one for each neuron’s delay counter. After a transition step is performed, all counters from closed neurons are decremented in one time.

4 Preliminary results

In order to test the first version of the simulator, we have based on a SNP system family without delays designed to sort natural numbers [6]. We fixed to 100 the amount of numbers to be sorted. This instance of the problem required $q = 3n = 300$ neurons, $m = n + n^2 = 10,100$ rules and $z = n = 100$ maximum out degree. According to [9], the size of the sparse representation is of order $3n^3 + 6n^2 + 5n + 1 = 3,060,501$, ELL is $2n^3 + 7n^2 + 11n + 1 = 2,071,101$, and optimized is $7n^2 + 13n + 1 = 71,301$. The GPU employed for the test were an RTX2080 (2944 cores, 8 GBytes GDDR5).

Kernel	Sparse	ELL	Optimized
Configuration vector	1350.60	21.265	18.992
Spiking vector	6.114	7.27	7.286

Table 1: Total execution time of kernels (in ms) for computing spiking and configuration vector, for each matrix representation. The employed model is an SNP system sorting 100 natural numbers.

Table 1 shows the execution time of the two main CUDA kernels (both computing the spiking vector and the configuration vector at each transition) on our RTX2080. First of all, for this model, we can see that using compressed

representations of matrices for simulating SNP systems is much better than using sparse representation. ELL is up to 63.5 times faster for configuration vector kernel, but Optimized is only 11% faster than ELL. For spiking vector, the computation is a bit slower (17%) in ELL and Optimized. Although the spiking vector is smaller in these versions (number of neurons instead of rules), the kernel is a bit affected.

5 Conclusions and Future Work

In this short paper, we have introduced our first steps towards a GPU implementation of compressed matrix representation for SNP systems. We have run our first version with a model designed to sort 100 natural numbers. We report up to 63x of speedup compared to sparse representation, but only 11% of speedup when using the optimized design.

We can conclude that the results can be further improved. Our future plan is to optimize the kernels, putting more efforts on improving the parallelism to better fit the GPU architecture. We are also working on running other examples with our simulators to better characterize them. Moreover, we are expanding the simulators to fully support SNP systems with delays and SNP systems with dynamic networks (budding, division and plasticity). Furthermore we plan to handle more types of regular expressions, e.g. $a^i(a^j)^*$ where $j \geq 0$ and $i \geq 1$, and $*$ can be replaced with $+$. More types of SNP systems can be also considered for this representation: SNP systems with weights, thresholds, rules on synapses, fuzzy reasoning SNP systems, dendrite P systems, etc.

Our main goal is also to provide a flexible framework to simulate SNP systems, by providing an API in common languages such as Python or C++, in order to programmatically define and simulate SNP systems. This is inspired from modern Deep Learning frameworks such as Keras and PyTorch. P-Lingua 5 will be also employed in order to define the SNP models from text files with a syntax close to the one used by model designers.

As a general remark, it can be concluded that spiking neural P system model is convenient for parallelization, since deciding whether a neuron spikes (and by which rule) does not depend on other neurons, while updating a configuration can be realized by scatter operations (e.g. atomic operations on GPUs) [10]. Nevertheless, synchronization is necessary between the transition steps, and between choosing the spiking vector and updating the configuration.

Acknowledgements

This work was supported by MABICAP research project: *FEDER/Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación/ _Proyecto (TIN2017-89842-P)*. D. Orellana-Martín also acknowledges Contratación de Personal Investigador Doctor. (Convocatoria 2019) 43 Contratos Capital Humano Línea 2. Paidi 2020, supported by the European Social Fund and Junta de Andalucía. F.G.C. Cabarle is supported in part by the ERDT program of the *DOST-SEI*,

Philippines, and the Dean Ruben A. Garcia PCA from UP Diliman. The authors also acknowledge the comments received by the reviewers.

References

1. Cabarle, F.G.C., Adorna, H.N., Martínez-del-Amor, M.A., Pérez-Jiménez, M.J.: Improving GPU simulations of spiking neural P systems. *Romanian Journal of Information Science and Technology* **15**(1), 5–20 (2012)
2. Carandang, J., Villaflores, J., Cabarle, F., Adorna, H., Martínez-del-Amor, M.: CuSNP: Spiking neural P systems simulators in CUDA. *Romanian Journal of Information Science and Technology* **20**, 57–70 (2017)
3. Carandang, J.P., Cabarle, F.G.C., Adorna, H.N., Hernandez, N.H.S., Martínez-del-Amor, M.Á.: Handling non-determinism in spiking neural P systems: Algorithms and simulations. *Fundamenta Informaticae* **164**(2-3), 139–155 (2019). <https://doi.org/10.3233/FI-2019-1759>
4. Fatahalian, K., Sugerman, J., Hanrahan, P.: Understanding the efficiency of GPU algorithms for matrix-matrix multiplication. In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. p. 133–137. *HWWS '04*, Association for Computing Machinery, New York, NY, USA (2004). <https://doi.org/10.1145/1058129.1058148>
5. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. *Fundamenta Informaticae* **71**(2,3), 279–308 (feb 2006)
6. Ionescu, M., Sburlan, D.: Some applications of spiking neural p systems. *Computing and Informatics* **27**, 515–528 (01 2008)
7. Kirk, D.B., Hwu, W.W.: *Programming massively parallel processors: a hands-on approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edn. (2016), <https://www.sciencedirect.com/science/book/9780128119860>
8. Martínez-del-Amor, M.A., García-Quismondo, M., Macías-Ramos, L.F., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Simulating P systems on GPU devices: a survey. *Fundamenta Informaticae* **136**(3), 269–284 (2015)
9. Martínez-del-Amor, M.A., Orellana-Martín, D., Pérez-Hurtado, I., Cabarle, F.G.C., Adorna, H.N.: Simulation of spiking neural P systems with sparse matrix-vector operations. *Processes* **9**(4) (2021). <https://doi.org/10.3390/pr9040690>
10. Orellana-Martín, D., Martínez-del-Amor, M.A., Valencia-Cabrera, L., Pérez-Hurtado, I., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Dendrite p systems toolbox: Representation, algorithms and simulators. *International Journal of Neural Systems* **31**(01), 2050071 (2021). <https://doi.org/10.1142/S0129065720500719>
11. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del-Amor, M.A., Pérez-Jiménez, M.J.: An interactive timeline of simulators in membrane computing. *Journal of Membrane Computing* **1**(3), 209–222 (2019). <https://doi.org/10.1007/s41965-019-00016-z>
12. Zeng, X., Adorna, H., Martínez-del-Amor, M.A., Pan, L., Pérez-Jiménez, M.J.: Matrix representation of spiking neural P systems. In: *Proceedings of the 11th International Conference on Membrane Computing*. vol. 6501, pp. 377–391 (08 2010). https://doi.org/10.1007/978-3-642-18123-8_29
13. Zhang, G., Shang, Z., Verlan, S., Martínez-del-Amor, M.A., Yuan, C., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: An overview of hardware implementation of membrane computing models. *ACM Comput. Surv.* **53**(4) (Aug 2020). <https://doi.org/10.1145/3402456>

Comparison Results On Parallel Contextual Array P Systems and Parallel Contextual Array Insertion Deletion P Systems

S. James Immanuel¹[0000-0003-0653-4882], S. Jayasankar²[0000-0001-9896-0779],
D. Gnanaraj Thomas³[0000-0001-6327-8446], T. Robinson⁴[0000-0001-8507-4196],
and Atulya K Nagar⁵[0000-0001-5549-6435]

¹ Department of Mathematics, Sri Sairam Institute of Technology, Chennai - 600044, India

james_imch@yahoo.co.in

² Department of Mathematics, Ramakrishna Mission Vivekananda College, Chennai - 600004, India

ksjayjay@gmail.com

³ Department of Applied Mathematics, Saveetha School of Engineering, SIMATS, Chennai - 602105, India

gnanarajthomas@gmail.com

⁴ Department of Mathematics, Madras Christian College, Chennai - 600 059, India

robinson@mcc.edu.in

⁵ Department of Mathematics and Computer Science, Liverpool Hope University, Liverpool L16 9JD UK

nagara@hope.ac.uk

Abstract. Two-dimensional languages (Picture languages) are natural generalisations of one-dimensional languages (string languages). There are various tools/systems to generate, recognize picture languages. Georghe Păun, inspired by the dynamics of a living cell, introduced a system called P system to recognize formal languages. Subsequently, various P systems were introduced in the literature of membrane computing by the scholars and researchers. James et al. introduced two P systems namely Parallel Contextual Array P Systems (PCAPS) employing contextual operations and Parallel Contextual Array Insertion Deletion P Systems (PCAIDPS) exploiting insertion and deletion operations together with contextual operations. In this paper, we compare the generative powers of PCAPS with that of PCAIDPS and prove that PCAIDPS has more generative power than PCAPS.

Keywords: P Systems, Internal Contextual, External Contextual, Parallel Contextual Array P Systems, Parallel Contextual Array Insertion Deletion P Systems

1 Introduction

In 1969 Marcus [24] came out with an entirely different class of grammars different from Chomsky grammars called contextual grammars. A contextual grammar produces a language by starting from a given finite set of strings and adding

repetitively, pairs of strings (called as contexts), associated to sets of words (called selectors) to the string already obtained. Both Freund et al and Helen Chandra et al. [10,5] extended these grammars to two-dimensional arrays respectively in their own style. Freund et al. generalised the concept of contextual grammars and adopted a new and simple approach in [10]. Both row and column contexts are allowed and contextual rules are finite in parallel contextual array grammars [5].

D. Haussler [13] was the first to conceive context-free insertion systems as a generalisation of concatenation. L. Kari [20] studied the role of insertion and deletion operations in formal language theory in 1991. Domaratzki and Okhotin [8] and Ito, Kari, Thierrin and Yu [19,21] investigated different variations of insertion and deletion systems.

A P system (membrane system) introduced by Păun [25] is a distributed theoretical model with maximal parallelism based on the membrane structure and function of the living cells. P systems have proved to be a rich theoretical framework to study many computational problems besides giving a new impetus to formal language theory. Various types of P systems were introduced in the literature and their properties, computing power, normal forms and basic decision problems were studied [25,26]. S. N. Krishna et al [22] introduced a new variant of P system with string objects having insertion-deletion rules to control the production of strings. A. Alhazov [1] et al have considered insertion-deletion P systems with priority over insertion and showed that these P systems with one-symbol together with context-free insertion rules were able to generate Parikh sets of all recursively enumerable languages (PsRE). The contextual way of handling string objects in P systems has been considered by Madhu et al. [23] and the contextual P systems are found to be more powerful than ordinary string contextual grammars and their variants. Ceterchi et al. [3] introduced array P systems of the isometric variety, extending the string rewriting P systems to arrays using context-free type of rules. In [9], a P system model called contextual array P system with array objects and array contextual rules has been introduced based on the contextual style of array generation considered in [10], and its generative power in the description of picture arrays was examined. In [6], P system models namely, external and internal array contextual P systems were introduced. Influenced by the works on contextual style of external and internal parallel contextual array grammars [5,28], James et al. [18] introduced a new P system model called external and internal parallel contextual array P systems and parallel contextual array P systems by shuffling contexts on trajectories and studied some properties of the family of languages generated by these P systems and compared their generative capacities with other array generating P systems. James et al also have introduced another new contextual array P system subsequently called parallel contextual array insertion deletion P system (PCAIDPS) in [17], based on internal parallel contextual array grammars in [5] and proved that the families of local (*LOC*) and recognizable picture languages (*REC*) [11,12] of Giammaresi and Resitivo and context-sensitive matrix languages (*CSML*) [27] of Siromoney et al. are properly contained in the family

of languages generated by the parallel contextual array insertion deletion P systems with 2 membranes $\mathcal{L}(PCAIDPS_2)$ [17]. Jayasankar et al.[15] proved that $\mathcal{L}(PCAIDPS_2)$ properly contains the family of array languages generated by (Context-free : Right-linear Indexed Right-linear) Siromoney matrix grammars $\mathcal{L}(CF : RIR)$. James et al have proved that the family of picture languages generated by Tabled Matrix Grammars (TMGs) of Siromoney is a proper subset of the family of picture languages generated by PCAIDPS [14].

As we are aware of the fact that formal language theorists are still trying to bring in a hierarchy among the family of two-dimensional picture languages, we make a humble attempt in this direction by comparing the generative powers of the P System models, Parallel Contextual Array P Systems [16] and Parallel Contextual Array Insertion Deletion P Systems [17] and prove that the family of picture languages generated by PCAPS is properly contained in the family of picture languages generated by PCAIDPS.

The paper is organised as follows: In section 2, some basic definitions pertaining to IPCAPS, EPCAPS and PCAIDPS together with examples are recalled. In section 3, definitions of PCAIDGs and PCAIDPSs are given along with interesting examples. In section 4, we prove the main result: $\mathcal{L}(PCAPS) \subsetneq \mathcal{L}(PCAIDPS)$. Future work is identified in section 5.

2 Preliminaries

In this section, we recall some definitions pertaining to two-dimensional picture languages.

Let V be a finite alphabet, V^* , the set of words over V including the empty word λ . $V^+ = V^* - \{\lambda\}$. An array consists of finitely many symbols from V

arranged in rows and columns and is written in the form, $A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$

$a_{11} \cdots a_{1n}$
or $\begin{matrix} \vdots & \ddots & \vdots \end{matrix}$ or in short $A = [a_{ij}]_{m \times n}$, $a_{ij} \in V$, $i = 1, 2, \dots, m$ and $j =$

$a_{m1} \cdots a_{mn}$
 $1, 2, \dots, n$. The set of all arrays over V is denoted by V^{**} which also includes the empty array Λ (zero rows and zero columns). $V^{++} = V^{**} - \{\Lambda\}$. The column

concatenation of $A = \begin{bmatrix} a_{11} & \cdots & a_{1p} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mp} \end{bmatrix}$ and $B = \begin{bmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nq} \end{bmatrix}$, defined only when

$m = n$, is given by $A \oplus B = \begin{bmatrix} a_{11} & \cdots & a_{1p} & b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mp} & b_{n1} & \cdots & b_{nq} \end{bmatrix}$.

As $1 \times n$ arrays can be easily interpreted as words of length n (and vice versa), we will then write their column concatenation by juxtaposition (as usual). Similarly, the row concatenation of A and B , defined only when $p = q$, is given

by $A \oplus B = \begin{bmatrix} a_{11} & \cdots & a_{1p} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mp} \\ b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nq} \end{bmatrix}$. The empty array acts as the identity for column

and row concatenation of arrays of arbitrary dimensions.

3 Parallel Contextual Array P System and Parallel Contextual Array Insertion Deletion P System

In this section, we recall some notions of parallel contextual array P system and parallel contextual array insertion deletion P system. For further details we refer to [16,17].

Definition 1. [16] *An internal parallel contextual array P system is a construct,*

$$\Pi = (V, T, \mu, C, R, M_1, M_2, \dots, M_h, (R_1, \varphi_1), (R_2, \varphi_2), \dots, (R_h, \varphi_h), \varphi_c, \varphi_r, i_0)$$

where,

V is the finite nonempty set of symbols called alphabet;

$T \subseteq V$ is the output alphabet;

μ is the membrane structure with h membranes or regions;

C is the finite subset of $V^{**}\$C V^{**}$ called column array contexts;

R is the finite subset of $V^{**}\$R V^{**}$ called row array contexts;

M_i is the finite sets of arrays over V called axioms, each associated with a region $i, 1 \leq i \leq h$ of μ ;

$\varphi_c : V^{**} \rightarrow 2^C$ is the choice mapping performing parallel column contextual operations;

$\varphi_r : V^{**} \rightarrow 2^R$ is the choice mapping performing parallel row contextual operations;

φ_i 's are either φ_c or φ_r ;

$$R_i = \emptyset \text{ (or) } \left\{ \left(\left\{ \varphi_c(A_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$C \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} / i = 1, 2, \dots, m-1 \right\}, \alpha \right) \right\} \text{ with}$$

$A_i = \begin{bmatrix} a_{ij} & \cdots & a_{ik} \\ a_{(i+1)j} & \cdots & a_{(i+1)k} \end{bmatrix}, 1 < j \leq k < n, \alpha \in \{\text{here, out, in}_t\}, u_i$ and u_{i+1} are of size $1 \times p, v_i$ and v_{i+1} are of size $1 \times q$ with $p, q \geq 1$

(or)

$$\left\{ \left(\left\{ \varphi_r(B_i) = [u_i \ u_{i+1}] \$R [v_i \ v_{i+1}] / i = 1, 2, \dots, n-1 \right\}, \alpha \right) \right\} \text{ with}$$

$B_i = \begin{bmatrix} a_{ji} & a_{j(i+1)} \\ \vdots & \vdots \\ a_{ki} & a_{k(i+1)} \end{bmatrix}, 1 < j \leq k < m, \alpha \in \{\text{here, out, in}_t\}, u_i$ and u_{i+1} are of size $p \times 1, v_i$ and v_{i+1} are of size $q \times 1$ with $p, q \geq 1$

i_0 is the output membrane

The direct derivation with respect to \amalg is a binary relation \Rightarrow on V^{**} and is defined as $X \Rightarrow_{in} Y$, where $X, Y \in V^{**}$ if and only if,
 $X = X_1 \oplus X_2 \oplus X_3, Y = X_1 \oplus L \oplus X_2 \oplus R \oplus X_3$ for some $X_1, X_2, X_3 \in V^{**}$, and L, R are contexts obtained by using the evolution rules R_i based on the parallel internal column contextual operations according to the choice mapping φ_c .

(or)

$X = X_1 \ominus X_2 \ominus X_3, Y = X_1 \ominus U \ominus X_2 \ominus D \ominus X_3$ for some $X_1, X_2, X_3 \in V^{**}$, and U, D are contexts obtained by using the evolution rules R_i based on the parallel internal row contextual operations according to the choice mapping φ_r .

The initial configuration of the system consists of the membrane structure with h membranes labelled $1, 2, \dots, h$, where h is an odd positive integer. We represent the structure μ as:

$[1[\frac{h-1}{2} \cdots [2]2 \cdots [\frac{h-1}{2}][h[h-1 \cdots [\frac{h+1}{2}]\frac{h+1}{2} \cdots]_{h-1}]_h]_1$, with the outermost membrane being the skin membrane labelled as 1, which also acts as our output membrane. Using the evolution rules R_i based on the choice mapping φ_i present in the region i we do the step by step computation. The array we obtain after each computation is considered to be of size $m \times n$. This array is placed in the membrane indicated by α . If we choose α to be 'here', it means that the resulting array remains in the same membrane. If we choose α to be 'out', it means that the resulting array is sent out of the current membrane and enters the immediate outer membrane. If that outer membrane happens to be the skin membrane, we say that the resulting array is present in the language generated by this P system. If we choose α to be 'in $_t$ ', it means that the resulting array is sent to the membrane labelled t . When there is no rule applicable to the choice array obtained after the last computation we say that the computation is successful and it halts. A successful computation depending on α may result in an array being sent out to the skin membrane. All the arrays with symbols over T collected in the skin membrane is the language generated by the internal parallel contextual array P system, denoted by $IPCALC(\amalg)$ or $L(\amalg)$.

The family of all array languages generated by internal parallel contextual array P systems \amalg with at most h membranes is denoted by $IPCAPC_h$.

Example 1. $\amalg = (V, T, \mu, C, R, M_1, M_2, M_3, (R_1, \varphi_1), (R_2, \varphi_2), (R_3, \varphi_3), \varphi_c, \varphi_r, 1)$

where,

$$\begin{aligned} V &= \{a, b\}; \\ T &= \{a, b\}; \\ \mu &= [1[2]2[3]3]_1; \\ C &= \left\{ \begin{bmatrix} a \\ b \end{bmatrix} \$c \begin{bmatrix} b \\ a \end{bmatrix}, \begin{bmatrix} a \\ a \end{bmatrix} \$c \begin{bmatrix} b \\ b \end{bmatrix}, \begin{bmatrix} b \\ b \end{bmatrix} \$c \begin{bmatrix} a \\ a \end{bmatrix} \right\}; \\ R &= \left\{ [a \ b] \$r [b \ a], [a \ a] \$r [b \ b], [b \ b] \$r [a \ a] \right\}; \\ M_1 &= \emptyset; \end{aligned}$$

$$\begin{aligned}
 M_2 &= \begin{pmatrix} a & a & b & b \\ a & a & b & b \\ b & b & a & a \\ b & b & a & a \end{pmatrix}; \\
 M_3 &= \emptyset; \\
 R_1 &= \emptyset; \\
 R_2 &= \left\{ \left(\left\{ \varphi_c(A_1) = \begin{bmatrix} a \\ b \end{bmatrix} \$c \begin{bmatrix} b \\ a \end{bmatrix}, \varphi_c(A_2) = \begin{bmatrix} a \\ a \end{bmatrix} \$c \begin{bmatrix} b \\ b \end{bmatrix}, \right. \right. \\
 &\quad \left. \left. \varphi_c(A_3) = \begin{bmatrix} b \\ b \end{bmatrix} \$c \begin{bmatrix} a \\ a \end{bmatrix} \right\}, in_3 \right\}; \\
 A_1 &= \begin{bmatrix} a & b \\ b & a \end{bmatrix}, A_2 = \begin{bmatrix} a & b \\ a & b \end{bmatrix}, A_3 = \begin{bmatrix} b & a \\ b & a \end{bmatrix} \\
 R_3 &= \left\{ \left(\left\{ \varphi_r(B_1) = [a \ b] \$r [b \ a], \varphi_r(B_2) = [a \ a] \$r [b \ b] \right\} \right. \right. \\
 &\quad \left. \left. , \varphi_r(B_3) = [b \ b] \$r [a \ a] \right\}, \alpha \right\}; \\
 B_1 &= \begin{bmatrix} a & b \\ b & a \end{bmatrix}, B_2 = \begin{bmatrix} a & a \\ b & b \end{bmatrix}, B_3 = \begin{bmatrix} b & b \\ a & a \end{bmatrix}, \alpha \in \{out, in_2\}
 \end{aligned}$$

Membrane labelled 1 i.e., the skin membrane is the output membrane.

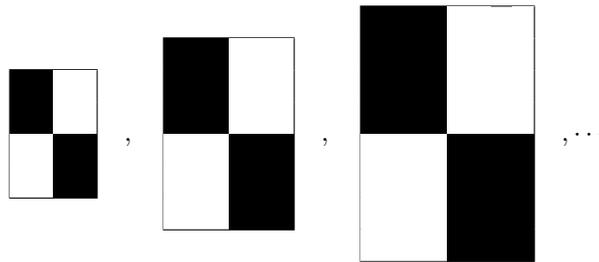
The language generated by this internal parallel contextual array P system is ,

$$L_1 = L(\Pi) = \left\{ \begin{matrix} (a^n b^n)_n \\ (b^n a^n)_n \end{matrix} / n \geq 2 \right\}$$

We note that if every element in an $m \times n$ array is a , then we write it as $(a^n)_m$ i.e.,

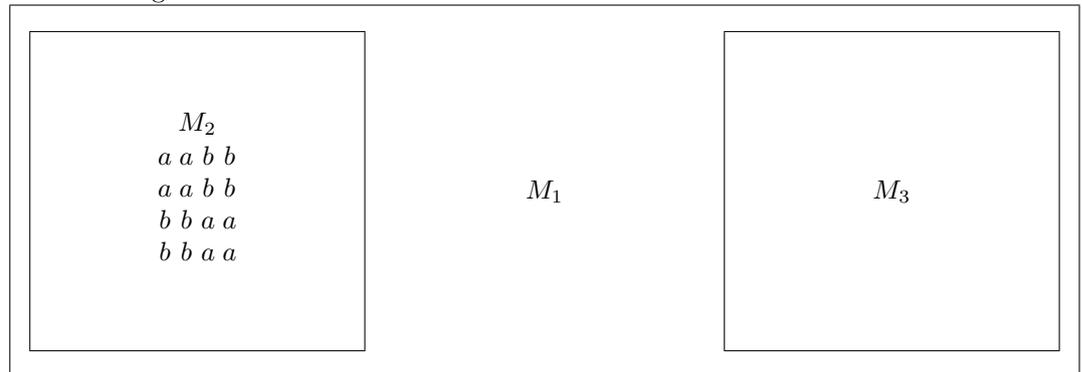
$$(a^n)_m = \left. \begin{matrix} \overbrace{a \cdots a}^{n \text{ columns}} \\ \vdots \cdots \vdots \\ a \cdots a \end{matrix} \right\} m \text{ rows}$$

By taking ‘ a ’ as black square and ‘ b ’ as white square, we obtain the set of pictures of the following forms,

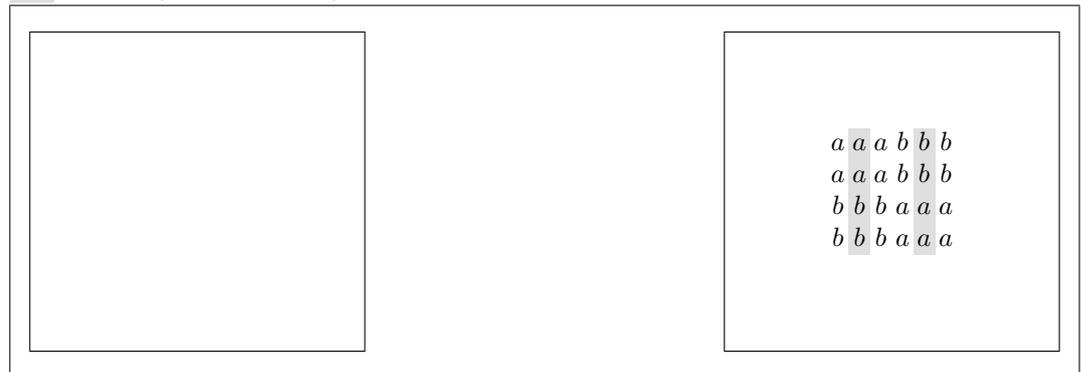


We give the working of this P system to generate the array $\begin{pmatrix} a^4 b^4 \\ b^4 a^4 \end{pmatrix}_4$

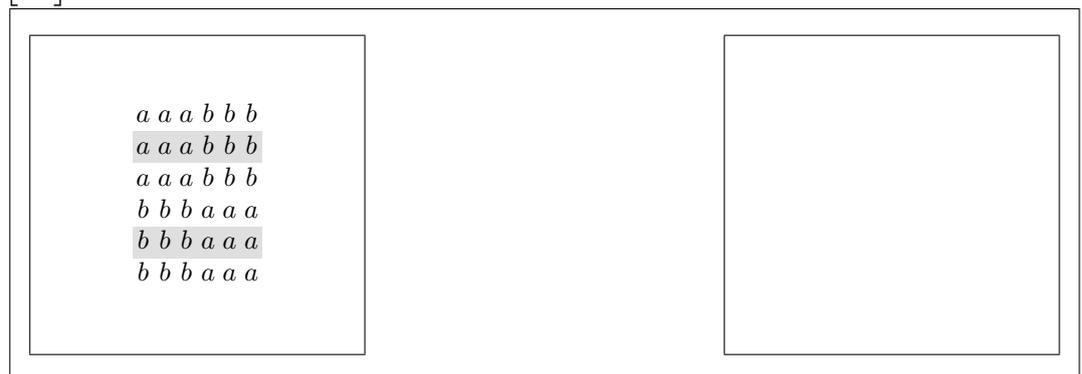
Initial Configuration:



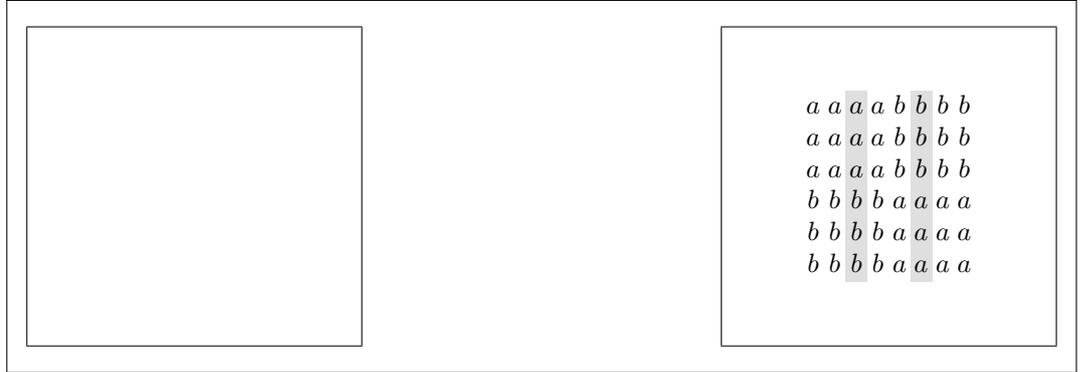
Rule to Apply: $\left(\left\{ \varphi_c \begin{bmatrix} a & b \\ b & a \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} \$_c \begin{bmatrix} b \\ a \end{bmatrix}, \varphi_c \begin{bmatrix} a & b \\ a & b \end{bmatrix} = \begin{bmatrix} a \\ a \end{bmatrix} \$_c \begin{bmatrix} b \\ b \end{bmatrix}, \varphi_c \begin{bmatrix} b & a \\ b & a \end{bmatrix} = \begin{bmatrix} b \\ b \end{bmatrix} \$_c \begin{bmatrix} a \\ a \end{bmatrix} \right\}, in_3 \right) \in R_2$
 - is the (row or column) context inserted



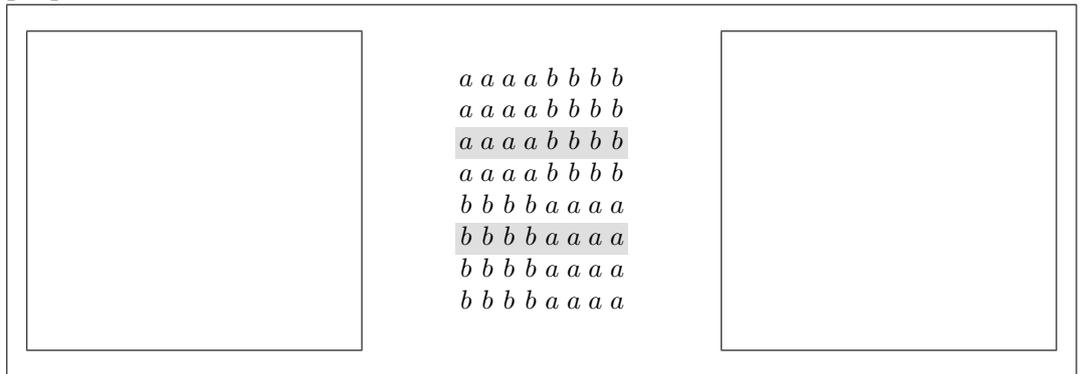
Rule to Apply: $\left(\left\{ \varphi_r \begin{bmatrix} a & b \\ b & a \end{bmatrix} = [a b] \$_r [b a], \varphi_r \begin{bmatrix} a & a \\ b & b \end{bmatrix} = [a a] \$_r [b b], \varphi_r \begin{bmatrix} b & b \\ a & a \end{bmatrix} = [b b] \$_r [a a] \right\}, in_2 \right) \in R_3$



$$\text{Rule to Apply: } \left(\left\{ \varphi_c \begin{bmatrix} a & b \\ b & a \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} \$c \begin{bmatrix} b \\ a \end{bmatrix}, \varphi_c \begin{bmatrix} a & b \\ a & b \end{bmatrix} = \begin{bmatrix} a \\ a \end{bmatrix} \$c \begin{bmatrix} b \\ b \end{bmatrix}, \varphi_c \begin{bmatrix} b & a \\ b & a \end{bmatrix} = \begin{bmatrix} b \\ b \end{bmatrix} \$c \begin{bmatrix} a \\ a \end{bmatrix} \right\}, in_3 \right) \in R_2$$



$$\text{Rule to Apply: } \left(\left\{ \varphi_r \begin{bmatrix} a & b \\ b & a \end{bmatrix} = [a & b] \$r [b & a], \varphi_r \begin{bmatrix} a & a \\ b & b \end{bmatrix} = [a & a] \$r [b & b], \varphi_r \begin{bmatrix} b & b \\ a & a \end{bmatrix} = [b & b] \$r [a & a] \right\}, out \right) \in R_3$$



Definition 2. [16] An external parallel contextual array **P** system is a construct,

$$\Pi = (V, T, \mu, C, R, M_1, M_2, \dots, M_h, (R_1, \varphi_1), (R_2, \varphi_2), \dots, (R_h, \varphi_h), \varphi_c, \varphi_r, i_0)$$

where,

V is the finite nonempty set of symbols called alphabet;

$T \subseteq V$ is the output alphabet;

μ is the membrane structure with h membranes or regions;

C is the finite subset of $V^{**} \$c V^{**}$ called column array contexts;

R is the finite subset of $V^{**} \$r V^{**}$ called row array contexts;

M_i is the finite set of arrays over V called axioms, each associated with a region $i, 1 \leq i \leq h$ of μ ;

$\varphi_c : V^{**} \rightarrow 2^C$ is the choice mapping performing parallel column contextual operations;

$\varphi_r : V^{**} \rightarrow 2^R$ is the choice mapping performing parallel row contextual operations;

φ_i 's are either φ_c or φ_r ;

$R_i = \emptyset$ (or) $\left\{ \left(\left\{ \varphi_c(A_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$_c \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} / i = 1, 2, \dots, m-1 \right\}, \alpha \right) \right\}$ with

$A_i = \begin{bmatrix} a_{i1} & \dots & a_{in} \\ a_{(i+1)1} & \dots & a_{(i+1)n} \end{bmatrix}$, $\alpha \in \{here, out, in_t\}$, u_i and u_{i+1} are of size $1 \times p$, v_i and v_{i+1} are of size $1 \times q$ with $p, q \geq 1$

(or)

$\left\{ \left(\left\{ \varphi_r(B_i) = [u_i \ u_{i+1}] \$_r [v_i \ v_{i+1}] / i = 1, 2, \dots, n-1 \right\}, \alpha \right) \right\}$ with

$B_i = \begin{bmatrix} a_{1i} & a_{1(i+1)} \\ \vdots & \vdots \\ a_{mi} & a_{m(i+1)} \end{bmatrix}$, $\alpha \in \{here, out, in_t\}$, u_i and u_{i+1} are of size $p \times 1$, v_i and v_{i+1} are of size $q \times 1$ with $p, q \geq 1$

i_0 is the output membrane

The direct derivation with respect to Π is a binary relation \Rightarrow on V^{**} and is defined as $X \Rightarrow_{ex} Y$, where $X, Y \in V^{**}$ if and only if,

$Y = L \oplus X \oplus R$ where, L, R are contexts obtained by the parallel external column contextual operations according to the choice mapping.

(or)

$Y = U \oplus X \oplus D$ where, U, D are contexts obtained by the parallel external row contextual operations according to the choice mapping.

The working of the external parallel array contextual P system is the same as the internal parallel array contextual P system except that the contexts are obtained externally using the evolution rules provided based on the parallel external column or row contextual operations according to the choice mapping φ_c or φ_r . Similar to the internal parallel array contextual P system every successful computation depending on α may result in an array being sent out to the skin membrane. All the arrays with symbols over T collected in the skin membrane is the language generated by the external parallel contextual array P system, denoted by $EPCALC(\Pi)$ or $L(\Pi)$.

The family of all array languages generated by external parallel contextual array P systems with at most h membranes is denoted by $EPCAPC_h$.

Example 2. $\Pi = (V, T, \mu, C, R, M_1, M_2, M_3, (R_1, \varphi_1), (R_2, \varphi_2), (R_3, \varphi_3), \varphi_c, \varphi_r, 1)$

where,

$$V = \{a, b\};$$

$$T = \{a, b\};$$

$$\mu = [1[2]2[3]3]1;$$

$$\begin{aligned}
 C &= \left\{ \begin{bmatrix} b & a \\ a & a \end{bmatrix} \$c \begin{bmatrix} a & b \\ a & a \end{bmatrix}, \begin{bmatrix} a & a \\ b & a \end{bmatrix} \$c \begin{bmatrix} a & a \\ a & b \end{bmatrix} \right\}; \\
 R &= \left\{ \begin{bmatrix} b & a \\ a & a \end{bmatrix} \$r \begin{bmatrix} a & a \\ b & a \end{bmatrix}, \begin{bmatrix} a & b \\ a & a \end{bmatrix} \$r \begin{bmatrix} a & a \\ a & b \end{bmatrix} \right\}; \\
 M_1 &= \emptyset; \\
 M_2 &= \left\{ \begin{bmatrix} b & a & b \\ a & a & a \\ b & a & b \end{bmatrix} \right\}; \\
 M_3 &= \emptyset; \\
 R_1 &= \emptyset; \\
 R_2 &= \left\{ \left(\{ \varphi_c(A_1) = \Lambda \$c \Lambda, \varphi_c(A_2) = \Lambda \$c \Lambda, \}, in_3 \right), \right. \\
 &\quad \left. \left(\left\{ \varphi_c(A_1) = \begin{bmatrix} b & a \\ a & a \end{bmatrix} \$c \begin{bmatrix} a & b \\ a & a \end{bmatrix}, \varphi_c(A_2) = \begin{bmatrix} a & a \\ b & a \end{bmatrix} \$c \begin{bmatrix} a & a \\ a & b \end{bmatrix} \right\}, \alpha \right) \right\}; \\
 A_1 &= \left[\begin{bmatrix} (b & a)^n & b \\ a & a & a \end{bmatrix} \right], A_2 = \left[\begin{bmatrix} (a & a)^n & a \\ b & a & b \end{bmatrix} \right], n \geq 1, \alpha \in \{here, out, in_3\} \\
 R_3 &= \left\{ \left(\{ \varphi_r(B_1) = \begin{bmatrix} b & a \\ a & a \end{bmatrix} \$r \begin{bmatrix} a & a \\ b & a \end{bmatrix}, \varphi_r(B_2) = \begin{bmatrix} a & b \\ a & a \end{bmatrix} \$c \begin{bmatrix} a & a \\ a & b \end{bmatrix} \right\}, \alpha \right\}; \\
 B_1 &= \left[\begin{bmatrix} (b & a) \\ (a & a) \\ b & a \end{bmatrix}_m \right], B_2 = \left[\begin{bmatrix} (a & b) \\ (a & a) \\ a & b \end{bmatrix}_m \right], m \geq 1, \alpha \in \{here, out, in_3\}
 \end{aligned}$$

Membrane labelled 1 i.e., the skin membrane is the output membrane.

The language generated by this external parallel contextual array P system is,

$$L_2 = L(\Pi) = \left\{ \begin{bmatrix} (b & a)^{2n-1} & (b) \\ (a & a)_{2m-1} & (a)_{2m-1} \\ (b & a)^{2n-1} & b \end{bmatrix} / n, m \geq 1 \right\}$$

Definition 3. [17] A parallel contextual array insertion deletion P system with h membranes ($PCAIDPS_h$) is a construct,

$$\Pi = (V, T, \mu, C, R, (M_1, I_1, D_1), \dots, (M_h, I_h, D_h), \varphi_c^i, \varphi_r^i, \varphi_c^d, \varphi_r^d, i_0)$$

where,

- V is the finite nonempty set of symbols called alphabet;
- $T \subseteq V$ is the output alphabet;
- μ is the membrane structure with h membranes or regions;
- C is the finite subset of V^{**} called set of column array contexts;
- R is the finite subset of V^{**} called set of row array contexts;
- M_i is the finite set of arrays over V called as axioms associated with the region μ_i of μ ;

- $\varphi_c^i : V^{**} \times V^{**} \rightarrow 2^C$ is the partial mapping performing parallel column contextual insertion operations;
- $\varphi_r^i : V^{**} \times V^{**} \rightarrow 2^R$ is the partial mapping performing parallel row contextual insertion operations;
- $\varphi_c^d : V^{**} \times V^{**} \rightarrow 2^C$ is the partial mapping performing parallel column contextual deletion operations;
- $\varphi_r^d : V^{**} \times V^{**} \rightarrow 2^R$ is the partial mapping performing parallel row contextual deletion operations;

– $I_i = \emptyset$ (or) $\left\{ \left(\left\{ \varphi_c^i(A_i, B_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \mid i = 1, 2, \dots, m-1 \right\}, \alpha \right) \right\}$ where
 $A_i = \begin{bmatrix} a_{ij} & \cdots & a_{i(k-1)} \\ a_{(i+1)j} & \cdots & a_{(i+1)(k-1)} \end{bmatrix}, B_i = \begin{bmatrix} a_{ik} & \cdots & a_{i(l-1)} \\ a_{(i+1)k} & \cdots & a_{(i+1)(l-1)} \end{bmatrix}, 1 \leq j \leq k < l \leq n+1$ (or)
 $1 \leq j < k \leq l \leq n+1, \alpha \in \{here, out, in_t\}, u_i$ and u_{i+1} are arrays of size $1 \times p$ with $p \geq 1$.

(or)
 $\left\{ \left(\left\{ \varphi_r^i(C_i, E_i) = [u_i \ u_{i+1}] \mid i = 1, 2, \dots, n-1 \right\}, \alpha \right) \right\}$ where
 $C_i = \begin{bmatrix} a_{ji} & a_{j(i+1)} \\ \vdots & \vdots \\ a_{(k-1)i} & a_{(k-1)(i+1)} \end{bmatrix}, E_i = \begin{bmatrix} a_{ki} & a_{k(i+1)} \\ \vdots & \vdots \\ a_{(l-1)i} & a_{(l-1)(i+1)} \end{bmatrix}, 1 \leq j \leq k < l \leq m+1$

(or) $1 \leq j < k \leq l \leq m+1, \alpha \in \{here, out, in_t\}$, where 'here' stands for the current membrane where the actions are being performed, 'out' stands for immediate outer membrane of the current membrane and 'in_t' stands for the specified membrane t to which the controls are to be transferred to. Here u_i and u_{i+1} are arrays of size $p \times 1$ with $p \geq 1$.

– $D_i = \emptyset$ (or) $\left\{ \left(\left\{ \varphi_c^d(A_i, B_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \mid i = 1, 2, \dots, m-1 \right\}, \alpha \right) \right\}$ where
 $A_i = \begin{bmatrix} a_{ij} & \cdots & a_{i(k-1)} \\ a_{(i+1)j} & \cdots & a_{(i+1)(k-1)} \end{bmatrix}, B_i = \begin{bmatrix} a_{i(k+p)} & \cdots & a_{i(l-1)} \\ a_{(i+1)(k+p)} & \cdots & a_{(i+1)(l-1)} \end{bmatrix}, 1 \leq j \leq k < l \leq n+1, \alpha \in \{here, out, in_t\}, u_i$ and u_{i+1} are arrays of size $1 \times p$ with $p \geq 1$.

(or)
 $\left\{ \left(\left\{ \varphi_r^d(C_i, E_i) = [u_i \ u_{i+1}] \mid i = 1, 2, \dots, n-1 \right\}, \alpha \right) \right\}$ where
 $C_i = \begin{bmatrix} a_{ji} & a_{j(i+1)} \\ \vdots & \vdots \\ a_{(k-1)i} & a_{(k-1)(i+1)} \end{bmatrix}, E_i = \begin{bmatrix} a_{(k+p)i} & a_{(k+p)(i+1)} \\ \vdots & \vdots \\ a_{(l-1)i} & a_{(l-1)(i+1)} \end{bmatrix}, 1 \leq j \leq k < l \leq m+1,$

$\alpha \in \{here, out, in_t\}, u_i$ and u_{i+1} are arrays of size $p \times 1$ with $p \geq 1$.

– i_0 is the output membrane.

The array language generated by Π is denoted by $L(\Pi)$ and the family of array languages generated by $PCAIDPS$ with h membranes is denoted by $\mathcal{L}(PCAIDPS_h)$. If $\Pi = (V, T, \mu, C, R, (M_1, I_1), \dots, (M_h, I_h), \varphi_c^i, \varphi_r^i, i_0)$, then the P system is denoted by $PCAIPSh$. The family of array languages generated by $PCAIPSh$ with h membranes is denoted by $\mathcal{L}(PCAIPSh)$.

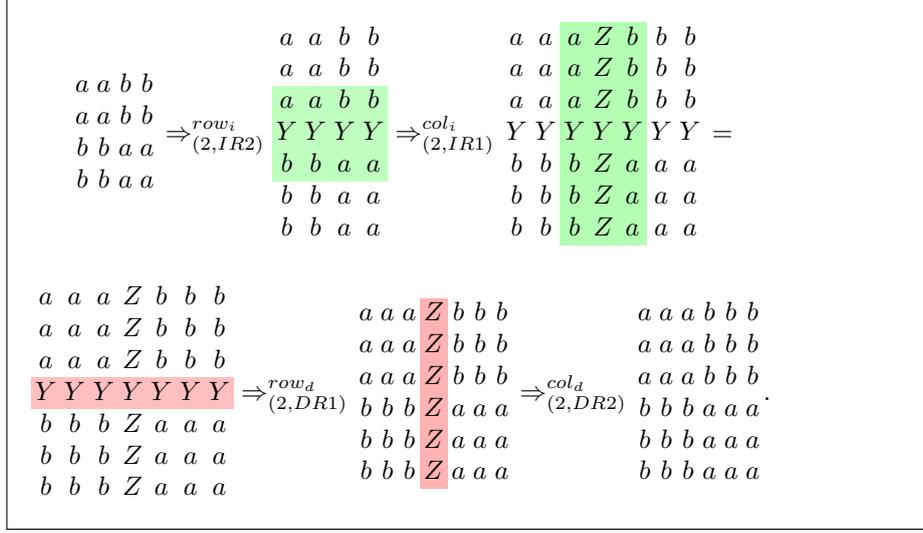
We give below two $PCAIDPS_2, \Pi_1$ and Π_2 to generate the array language L_1 given in Example 1 and the language L_2 given in Example 2 respectively.

Example 3. $\Pi_1 = \left(V, T, \mu, C, R, (M_1, I_1, D_1), (M_2, I_2, D_2), \varphi_c^i, \varphi_r^i, \varphi_c^d, \varphi_r^d, i_0 \right)$, where

- $V = \{b, a, Y, Z\}$;
- $T = \{b, a\}$;
- $\mu = [2[1]_1]_2$;
- $C = \left\{ \begin{array}{l} a b \ a b \ b a \\ a b \ b a \ b a \end{array} \right\}$;
- $R = \left\{ \begin{array}{l} a a \ a b \ b b \\ b b \ b a \ a a \end{array} \right\}$;
- $M_1 = \emptyset$; $M_2 = \left\{ \begin{array}{l} a a \ b b \\ a a \ b b \\ b b \ a a \\ b b \ a a \end{array} \right\}$;
- $I_1 = \emptyset$;
- $I_2 = \{IR1, IR2\}$ with
 - $IR1 = \left\{ \varphi_c^i \left[\begin{array}{l} a a \ , \ b b \\ a a \ , \ b b \end{array} \right] = \left\{ \begin{array}{l} a \ Z \ b \\ a \ Z \ b \end{array} \right\}, \varphi_c^i \left[\begin{array}{l} a a \ , \ b b \\ b b \ , \ a a \end{array} \right] = \left\{ \begin{array}{l} a \ Z \ b \\ b \ Z \ a \end{array} \right\}, \right.$
 $\left. \varphi_c^i \left[\begin{array}{l} b b \ , \ a a \\ b b \ , \ a a \end{array} \right] = \left\{ \begin{array}{l} b \ Z \ a \\ b \ Z \ a \end{array} \right\}, here \right\}$.
 - $IR2 = \left\{ \varphi_r^i \left[\begin{array}{l} a a \ , \ b b \\ a a \ , \ b b \end{array} \right] = \left\{ \begin{array}{l} a \ a \\ Y \ Y \\ b \ b \end{array} \right\}, \varphi_r^i \left[\begin{array}{l} a b \ , \ b a \\ a b \ , \ b a \end{array} \right] = \left\{ \begin{array}{l} a \ b \\ Y \ Y \\ a \ b \end{array} \right\}, \right.$
 $\left. \varphi_r^i \left[\begin{array}{l} b b \ , \ a a \\ b b \ , \ a a \end{array} \right] = \left\{ \begin{array}{l} b \ b \\ Y \ Y \\ a \ a \end{array} \right\}, here \right\}$.
- $D_1 = \emptyset$;
- $D_2 = \{DR1, DR2\}$ with
 - $DR1 = \left\{ \varphi_r^d \left[\begin{array}{l} a a \ , \ b b \\ a a \ , \ b b \end{array} \right] = \left\{ \begin{array}{l} Y \ Y \\ Y \ Y \end{array} \right\}, \varphi_r^d \left[\begin{array}{l} a Z \ , \ b Z \\ a Z \ , \ b Z \end{array} \right] = \left\{ \begin{array}{l} Y \ Y \\ Y \ Y \end{array} \right\}, \right.$
 $\left. \varphi_r^d \left[\begin{array}{l} Z b \ , \ Z a \\ Z b \ , \ Z a \end{array} \right] = \left\{ \begin{array}{l} Y \ Y \\ Y \ Y \end{array} \right\}, \varphi_r^d \left[\begin{array}{l} b b \ , \ a a \\ b b \ , \ a a \end{array} \right] = \left\{ \begin{array}{l} Y \ Y \\ Y \ Y \end{array} \right\}, here \right\}$.
 - $DR2 = \left\{ \varphi_c^d \left[\begin{array}{l} a \ , \ b \\ a \ , \ b \end{array} \right] = \left\{ \begin{array}{l} Z \\ Z \end{array} \right\}, \varphi_c^d \left[\begin{array}{l} a \ , \ b \\ b \ , \ a \end{array} \right] = \left\{ \begin{array}{l} Z \\ Z \end{array} \right\}, \varphi_c^d \left[\begin{array}{l} b \ , \ a \\ b \ , \ a \end{array} \right] = \left\{ \begin{array}{l} Z \\ Z \end{array} \right\}, out \right\}$.
- i_0 is the membrane 1.

A sample derivation of a picture of size 6×6 in L_1 is given as follows:

	- is the (row or column) context inserted
	- is the (row or column) to be deleted context

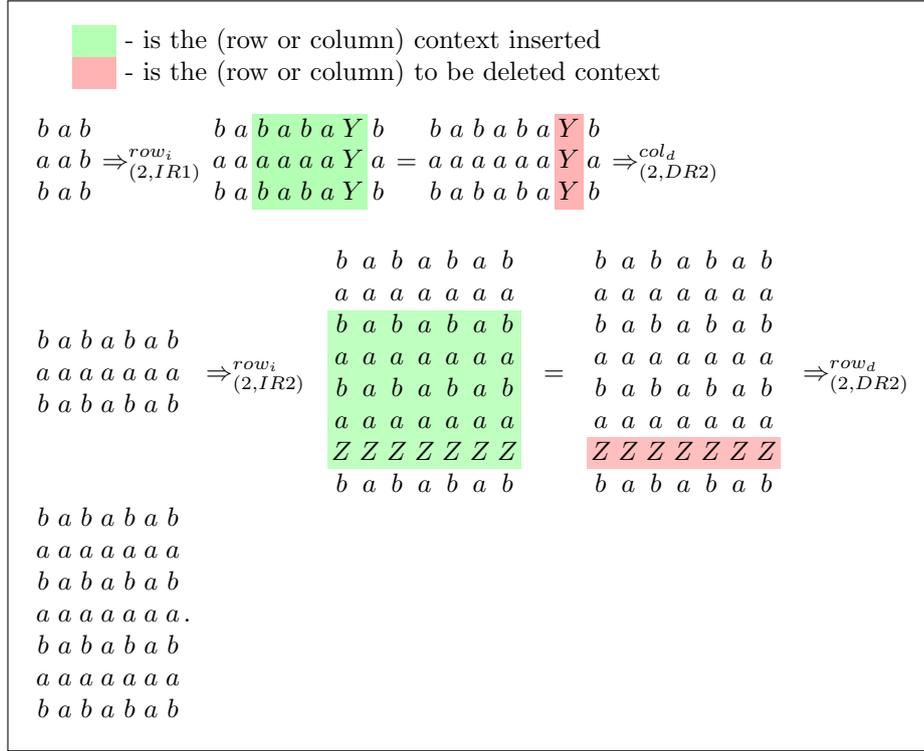


Example 4. $\Pi_2 = \left(V, T, \mu, C, R, (M_1, I_1, D_1), (M_2, I_2, D_2), \varphi_c^i, \varphi_r^i, \varphi_c^d, \varphi_r^d, i_0 \right)$, where

- $V = \{b, a, Y, Z\}$;
- $T = \{b, a\}$;
- $\mu = [2[1]1]2$;
- $C = \left\{ \begin{matrix} b a a a b a b a b a Y a a a a Y \\ a a ' b a ' a ' b ' a a a a Y ' b a b a Y \end{matrix} \right\}$;
- $R = \left\{ \begin{matrix} & & b a a b \\ & & a a a a \\ b a , a b & , b a , a b , b a , a b \\ a a , a a & , & a a a a \\ & & Z Z Z Z \end{matrix} \right\}$;
- $M_1 = \emptyset$; $M_2 = \left\{ \begin{matrix} b a b \\ a a a \\ b a b \end{matrix} \right\}$;
- $I_1 = \emptyset$;
- $I_2 = \{IR1, IR2, IR3, IR4\}$ with
 - $IR1 = \left\{ \left\{ \varphi_c^i \begin{bmatrix} b a & b \\ a a & a \end{bmatrix} = \begin{bmatrix} b a b a Y \\ a a a a Y \end{bmatrix} \right\}, \right.$
 $\left. \varphi_c^i \begin{bmatrix} a a & a \\ b a & b \end{bmatrix} = \left\{ \begin{bmatrix} a a a a Y \\ b a b a Y \end{bmatrix} \right\}, here \right\}$.
 - $IR2 = \left\{ \left\{ \varphi_r^i \begin{bmatrix} b a & b a \\ a a & a \end{bmatrix} = \begin{bmatrix} b a \\ a a \\ b a \\ a a \\ Z Z \end{bmatrix} \right\}, \varphi_r^i \begin{bmatrix} a b & a b \\ a a & a \end{bmatrix} = \begin{bmatrix} a b \\ a a \\ a b \\ a a \\ Z Z \end{bmatrix} \right\}, here \right\}$.

- $D_1 = \emptyset$;
- $D_2 = \{DR1, DR2\}$ with
 - $DR1 = \left\{ \left\{ \varphi_r^d \begin{bmatrix} b a \\ a a \end{bmatrix}, b a \right\} = \left\{ Z Z \right\}, \varphi_r^d \begin{bmatrix} a b \\ a a \end{bmatrix}, a b \right\} = \left\{ Z Z \right\} \right\}, \alpha$.
 - $DR2 = \left\{ \left\{ \varphi_c^d \begin{bmatrix} b a \\ a a \end{bmatrix}, b \right\} = \left\{ Y \right\}, \varphi_c^d \begin{bmatrix} a a \\ b a \end{bmatrix}, a \right\} = \left\{ Y \right\} \right\}, \alpha$
- , where $\alpha \in \{here, out\}$.
- i_0 is the membrane 1.

A sample derivation of a picture of size 7×7 in L_2 is given as follows:



4 Results

Theorem 1. $IPCAPC_h \subseteq \mathcal{L}(PCAIPS_h)$

Proof. For every internal parallel contextual array P system with h membranes generating a language L, we construct a corresponding parallel contextual array insertion P system generating the same language L.

Consider $\prod_{int} \in IPCAPC_h$ generating a picture language L as

$$\prod_{int} = \left(V, T, \mu, C, R, M_1, M_2, \dots, M_h, (R_1, \phi_1), (R_2, \phi_2), \dots, (R_h, \phi_h), \phi_c, \phi_r, i_0 \right).$$

We construct a PCAIPS_h, $\Pi = \left(V, T, \mu, C', R', (M_1, I_1), \dots, (M_g, I_g), \phi_c^i, \phi_r^i, i_0 \right)$,
 where $g = 2h$ if $R_i \neq \emptyset$, $g = 2h - 1$ if $R_i = \emptyset$.
 $\forall \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$_c \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in C, \exists \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}, \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in C'$. Similarly, $\forall [u_i \ u_{i+1}] \$_r [v_i \ v_{i+1}] \in R, \exists [u_i \ u_{i+1}], [v_i \ v_{i+1}] \in R'$.
 If $R_1 = \emptyset$ then $I_1 = \emptyset$.
 If $R_i = \left\{ \left(\left\{ \phi_c(A_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$_c \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} / i = 1, 2, \dots, m - 1 \right\}, \alpha_i \right) \right\}$ with $A_i = \begin{bmatrix} a_{ij} & \dots & a_{ik} \\ a_{(i+1)j} & \dots & a_{(i+1)k} \end{bmatrix}$, $1 < j \leq n, \alpha_i \in \{here, out, in_t\}$, u_i and u_{i+1} are of size $1 \times p$, v_i and v_{i+1} are of size $1 \times q$ with $p, q \geq 1$, then
 $I_i = \left\{ \left(\left\{ \phi_c^i(C_i, A_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} / i = 1, 2, \dots, m - 1 \right\}, in_{(i+h)} \right) \right\}$ where $C_i = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$
 with $c_1, c_2 \in T \cup \{\varepsilon\}$ and
 $I_j = \left\{ \left(\left\{ \phi_c^i(A_i, C_i) = \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} / i = 1, 2, \dots, m - 1 \right\}, in_{(i+h)} \alpha_i \right) \right\}$ where $j = i + h$ if $R_i \neq \emptyset$ else $j = i + h - 1$.
 If $R_i = \left\{ \left(\left\{ \phi_r(B_i) = [u_i \ u_{i+1}] \$_r [v_i \ v_{i+1}] / i = 1, 2, \dots, m - 1 \right\}, \alpha_i \right) \right\}$ with
 $B_i = \begin{bmatrix} a_{ji} & a_{ik} \\ \vdots & \vdots \\ a_{ki} & a_{k(i+1)} \end{bmatrix}$, $1 < j \leq k < m, \alpha_i \in \{here, out, in_t\}$, u_i and u_{i+1} are of size $p \times 1$, v_i and v_{i+1} are of size $q \times 1$ with $p, q \geq 1$, then
 $I_i = \left\{ \left(\left\{ \phi_r^i(E_i, B_i) = [u_i \ u_{i+1}] / i = 1, 2, \dots, n - 1 \right\}, in_{(i+h)} \right) \right\}$ where $E_i = [e_1 \ e_2]$
 with $e_1, e_2 \in T \cup \{\varepsilon\}$ and
 $I_j = \left\{ \left(\left\{ \phi_r^i(B_i, E_i) = [v_i \ v_{i+1}] / i = 1, 2, \dots, n - 1 \right\}, in_{(i+h)} \alpha_i \right) \right\}$ where $j = i + h$ if $R_i \neq \emptyset$ else $j = i + h - 1$.

Working Procedure : For every row (or column) insertion rule R_i of the membrane M_i of IPCAPS there corresponds a pair of row (or column) insertion rules R'_i and R''_i in the respective membranes M'_i and M''_i of PCAIPS. Once this process is over, the resulting picture in PCAIPS is sent to the output membrane by the rules present in the penultimate membrane.

Theorem 2. $EPCAPC_h \subseteq \mathcal{L}(PCAIPS_h)$

Proof. For every external parallel contextual array P system with h membranes generating a picture language L , we construct a corresponding parallel contextual array insertion P system generating the same language L .

Consider $\Pi_{ext} \in EPCAPC_h$ generating a picture language L as

$$\Pi_{ext} = \left(V, T, \mu, C, R, M_1, M_2, \dots, M_h, (R_1, \phi_1), (R_2, \phi_2), \dots, (R_h, \phi_h), \phi_c, \phi_r, i_0 \right).$$

We construct a PCAIPS_h, $\Pi = \left(V, T, \mu, C', R', (M_1, I_1), \dots, (M_g, I_g), \phi_c^i, \phi_r^i, i_0 \right)$,
 where $g = 2h$ if $R_i \neq \emptyset$, $g = 2h - 1$ if $R_i = \emptyset$.

For every $\begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$c \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in C, \exists \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}, \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in C'$. Similarly, $\forall [u_i \ u_{i+1}] \$r [v_i \ v_{i+1}] \in R, \exists [u_i \ u_{i+1}], [v_i \ v_{i+1}] \in R'$. Also, $D_i = \emptyset \ \forall i$. If $R_1 = \emptyset$ then $I_1 = \emptyset$.
 If $R_i = \left\{ \left(\left\{ \phi_c(A_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$c \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} / i = 1, 2, \dots, m-1 \right\}, \alpha_i \right) \right\}$ with $A_i = \begin{bmatrix} a_{i1} & \dots & a_{in} \\ a_{(i+1)1} & \dots & a_{(i+1)n} \end{bmatrix}, \alpha_i \in \{here, out, in_t\}$, u_i and u_{i+1} are of size $1 \times p$, v_i and v_{i+1} are of size $1 \times q$ with $p, q \geq 1$, then
 $I_i = \left\{ \left(\left\{ \phi_c^i(C_i, A_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} / i = 1, 2, \dots, m-1 \right\}, in_{(i+h)} \right) \right\}$ where $C_i = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$ with $c_1, c_2 \in T \cup \{\varepsilon\}$ and
 $I_j = \left\{ \left(\left\{ \phi_c^i(A_i, C_i) = \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} / i = 1, 2, \dots, m-1 \right\}, \alpha_i \right) \right\}$ where $j = i + h$ if $R_i \neq \emptyset$ else $j = i + h - 1$.
 If $R_i = \left\{ \left(\left\{ \phi_r(B_i) = [u_i \ u_{i+1}] \$r [v_i \ v_{i+1}] / i = 1, 2, \dots, n-1 \right\}, \alpha_i \right) \right\}$ with
 $B_i = \begin{bmatrix} a_{1i} & a_{1(i+1)} \\ \vdots & \vdots \\ a_{mi} & a_{m(i+1)} \end{bmatrix}, \alpha_i \in \{here, out, in_t\}$, u_i and u_{i+1} are of size $p \times 1$, v_i and v_{i+1} are of size $q \times 1$ with $p, q \geq 1$, then
 $I_i = \left\{ \left(\left\{ \phi_r^i(E_i, B_i) = [u_i \ u_{i+1}] / i = 1, 2, \dots, n-1 \right\}, in_{(i+h)} \right) \right\}$ where $E_i = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$ with $e_1, e_2 \in T \cup \{\varepsilon\}$ and
 $I_j = \left\{ \left(\left\{ \phi_r^i(B_i, E_i) = [v_i \ v_{i+1}] / i = 1, 2, \dots, n-1 \right\}, \alpha_i \right) \right\}$ where $j = i + h$ if $R_i \neq \emptyset$ else $j = i + h - 1$.

Working Procedure : For every row (or column) insertion rule R_i of the membrane M_i of EPCAPS there corresponds a pair of row (or column) insertion rules R'_i and R''_i in the respective membranes M'_i and M''_i of PCAIPS. Once this process is over, the resulting picture in PCAIPS is sent to the output membrane by the rules present in the penultimate membrane.

Theorem 3. $IPCAPC_h \subsetneq \mathcal{L}(PCAIDPS_h)$

Proof. For every internal parallel contextual array P system with h membranes generating a language L, we construct a corresponding parallel contextual array insertion deletion P system with deletion rules generating the same language L.

Consider a $\prod_{int} \in IPCAPC_h$ given by
 $\prod_{int} = \left(V, T, \mu, C, R, M_1, M_2, \dots, M_h, (R_1, \phi_1), (R_2, \phi_2), \dots, (R_h, \phi_h), \phi_c, \phi_r, i_0 \right)$
 generating a picture language L.

We construct a PCAIDPS_h,
 $\prod = \left(V', T, \mu, C', R', (M_1, I_1, D_1), \dots, (M_{h+1}, I_{h+1}, D_{h+1}), \phi_c^i, \phi_r^i, \phi_c^d, \phi_r^d, i_0 \right)$, where

$V' = V \cup \{Y\}$ and $\forall \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$c \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in C, \exists \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}, \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in C'$. Similarly,
 $\forall [u_i \ u_{i+1}] \$r [v_i \ v_{i+1}] \in R, \exists [u_i \ u_{i+1}], [v_i \ v_{i+1}] \in R'$. If $R_1 = \emptyset$ then $I_1 = D_1 = \emptyset$.

If $R_i = \left\{ \left(\left\{ \phi_c(A_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$c \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} / i = 1, 2, \dots, m-1 \right\}, \alpha_i \right) \right\}$ with $A_i = \begin{bmatrix} a_{ij} & \dots & a_{ik} \\ a_{(i+1)j} & \dots & a_{(i+1)k} \end{bmatrix}, 1 < j \leq k < n, \alpha_i \in \{here, out, in_t\}, u_i$ and u_{i+1} are of size $1 \times p, v_i$ and v_{i+1} are of size $1 \times q$ with $p, q \geq 1$, then

$I_i = \left\{ \left(\left\{ \phi_c^i(C_i, A_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \oplus \begin{bmatrix} Y \\ Y \end{bmatrix} / i = 1, 2, \dots, m-1 \right\}, here \right) \right\} \cup \left\{ \left(\left\{ \phi_c^i(A_i, C_i) = \begin{bmatrix} Y \\ Y \end{bmatrix} \oplus \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} / i = 1, 2, \dots, m-1 \right\}, here \right) \right\}$ where $C_i = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$ with $c_1, c_2 \in T \cup \{\varepsilon\}$.

$D_i = \left\{ \left(\left\{ \phi_c^d \left(\begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}, A_i \right) = \begin{bmatrix} Y \\ Y \end{bmatrix} / i = 1, 2, \dots, m-1 \right\}, in_{(h+1)} \right) \right\},$

$D_{h+1} = \left\{ \left(\left\{ \phi_c^d \left(A_i, \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \right) = \begin{bmatrix} Y \\ Y \end{bmatrix} / i = 1, 2, \dots, m-1 \right\}, \alpha_i \right) \right\}.$

If $R_i = \left\{ \left(\left\{ \phi_r(B_i) = [u_i \ u_{i+1}] \$r [v_i \ v_{i+1}] / i = 1, 2, \dots, n-1 \right\}, \alpha_i \right) \right\}$ with

$B_i = \begin{bmatrix} a_{ji} & a_{j(i+1)} \\ \vdots & \vdots \\ a_{ki} & a_{k(i+1)} \end{bmatrix}, 1 < j \leq k < m, \alpha_i \in \{here, out, in_t\}, u_i$ and u_{i+1} are of

size $p \times 1, v_i$ and v_{i+1} are of size $q \times 1$ with $p, q \geq 1$, then

$I_i = \left\{ \left(\left\{ \phi_r^i(E_i, B_i) = [u_i \ u_{i+1}] \ominus [Y \ Y] / i = 1, 2, \dots, n-1 \right\}, here \right) \right\} \cup \left\{ \left(\left\{ \phi_r^i(B_i, E_i) = [Y \ Y] \ominus [v_i \ v_{i+1}] / i = 1, 2, \dots, n-1 \right\}, here \right) \right\}$ where $E_i = \begin{bmatrix} e_1 & e_2 \end{bmatrix}$ with $e_1, e_2 \in T \cup \{\varepsilon\}$.

$D_i = \left\{ \left(\left\{ \phi_r^d([u_i \ u_{i+1}], B_i) = [Y \ Y] / i = 1, 2, \dots, n-1 \right\}, in_{(h+1)} \right) \right\}.$

$D_{h+1} = \left\{ \left(\left\{ \phi_r^d(B_i, [v_i \ v_{i+1}]) = [Y \ Y] / i = 1, 2, \dots, n-1 \right\}, \alpha_i \right) \right\}.$

For proper inclusion, we consider the picture language consisting of arrays of single row $L_3 = \{aa, aaa, aaaa, \dots\}$ with the axiom set $M = \{aa\}$. This language cannot be generated by any of the three P systems namely, IPCAPS, EPCAPS and PCAIPS_h; where as, it can be generated by PCAIDPS. We give below the construction of a PCAIDPS generating L_3 .

Consider the P system

$\Pi_3 = \left(V', T, \mu, C', R', (M_1, I_1, D_1), \dots, (M_{h+1}, I_{h+1}, D_{h+1}), \phi_c^i, \phi_r^i, \phi_c^d, \phi_r^d, i_0 \right),$

where $V' = V \cup \{Y\}$ with $V = \{a, Y\}$ and $T = \{a\}, C' = \{\lambda, a\}, R' = \emptyset,$

$(IR_1) = \left\{ \phi_r^i \begin{bmatrix} a & a \\ \lambda & \lambda \end{bmatrix} = [Y \ Y], \phi_c^i \begin{bmatrix} a & \lambda \\ Y & \lambda \end{bmatrix} = \begin{bmatrix} a \\ Y \end{bmatrix} \right\}; (IR_2) = \emptyset;$

$(DR_1) = \left\{ \phi_r^d \begin{bmatrix} a & a \\ \lambda & \lambda \end{bmatrix} = [Y \ Y] \right\};$

$(DR_2) = \left\{ \phi_r^d \begin{bmatrix} a & a \\ \lambda & \lambda \end{bmatrix} = [Y \ Y] \right\}.$

A sample derivation of a picture aaaaa is shown below:

$$\begin{array}{l}
 M = a a \xRightarrow{IR_1^{row_i}} \begin{array}{c} a a \\ Y Y \end{array} \xRightarrow{IR_1^{col_i}} \begin{array}{c} a a a \\ Y Y Y \end{array} \xRightarrow{IR_1^{col_i}} \begin{array}{c} a a a a \\ Y Y Y Y \end{array} \xRightarrow{IR_1^{col_i}} \\
 \begin{array}{c} a a a a a \\ Y Y Y Y Y \end{array} \xRightarrow{DR_2^{row_d}} a a a a a.
 \end{array}$$

Finally, the #'s bordering the picture (which are not shown here) is removed using the the rules in $(DR_2) = \{\phi_c^d[\lambda, a] = \{\#\}, \phi_c^d[a, \lambda] = \{\#\}\}$ in the membrane M_2 .

Working Procedure : For every insertion rule R_i of the membrane M_i of IPCAPS there corresponds a pair of insertion rules R'_i and R''_i in the respective membranes M'_i and M''_i followed by a pair of deletion rules R'_d and R''_d to delete the variables in the respective membranes of PCAIDPS. Depending upon the the value of $\alpha \in \{here, out, in_t\}$, the intermediate pictures are sent to the appropriate membrane. Once the pictures generated are free from variables, they are sent to the outermost membrane where the boundary symbols #'s gets deleted by the deletion rules and the resulting pictures are collected.

5 Conclusion

This paper strives to bring in a hierarchy among two-dimensional picture languages. There are other well known families of two-dimensional picture languages yet to be compared with the family of languages generated by PCAIDPS and thereby the generative power of PCAIDPS is revealed further and will be taken up in our future work. Some interesting classes of P systems to be compared imminently with PCAIDPS in terms of their generative powers are: (i) Parallel Contextual Array P Systems by shuffling contexts on trajectories [18], (ii) Contextual Array Grammars and Array P Systems of Henning Fernau et al. [9], (iii) Array rewriting P systems of Rodica Ceterchi et al. [3], and (iv) Parallel contextual array P systems of Somnath Bera et al. [2].

Daniel Díaz-Pernil et al. [7] surveyed applications of P systems in image processing. Rodica Ceterchi et al. [4] have constructed P systems to generate the approximations of geometric patterns of space filling curves such as Peanos curve, Hilberts curves and others. In this paper, we deal with applications of P systems in terms of rectangular arrays of letters yielding pictures or images by replacing letters by primitive symbols. The role of PCAIDPS in image analysis can be explored further.

References

1. Artiom Alhazov, Alexander Krassovitskiy, Yurii Rogozhin, and Sergey Verlan. P systems with minimal insertion and deletion. *Theoretical Computer Science*, 412(1):136–144, 2011.

2. Somnath Bera, Linqiang Pan, Bosheng Song, K.G. Subramanian, and Gexiang Zhang. Parallel contextual array p systems. *Int J Adv Eng Sci Appl Math*, 10(3):203–212, 2018.
3. Rodica Ceterchi, Madhu Mutyam, Gheorghe Paun, and K. G. Subramanian. Array-rewriting P systems. *Nat. Comput.*, 2(3):229–249, 2003.
4. Rodica Ceterchi and K. G. Subramanian. Generating pictures in string representation with P systems: the case of space-filling curves. *J. Membr. Comput.*, 2(4):369–379, 2020.
5. P. Helen Chandra, K. G. Subramanian, and D. G. Thomas. Parallel contextual array grammars and languages. *Electron. Notes Discret. Math.*, 12:106–117, 2003.
6. K. S. Dersanambika and Kamala Krithivasan. Contextual array P systems. *Int. J. Comput. Math.*, 81(8):955–969, 2004.
7. Daniel Díaz-Pernil, Miguel A. Gutiérrez-Naranjo, and Hong Peng. Membrane computing and image processing: a short survey. *J. Membr. Comput.*, 1(1):58–73, 2019.
8. Michael Domaratzki and Alexander Okhotin. Representing recursively enumerable languages by iterated deletion. *Theor. Comput. Sci.*, 314(3):451–457, 2004.
9. Henning Fernau, Rudolf Freund, Markus L. Schmid, K. G. Subramanian, and Petra Wiederhold. Contextual array grammars and array P systems. *Ann. Math. Artif. Intell.*, 75(1-2):5–26, 2015.
10. Rudolf Freund, Gheorghe Paun, and Grzegorz Rozenberg. Contextual array grammars. In *Formal Models, Languages and Applications*, volume 66 of *Series in Machine Perception and Artificial Intelligence*, pages 112–136. World Scientific, 2007.
11. Dora Giammarresi and Antonio Restivo. Recognizable picture languages. *Int. J. Pattern Recognit. Artif. Intell.*, 6(2&3):241–256, 1992.
12. Dora Giammarresi and Antonio Restivo. Two-dimensional languages. In *Handbook of Formal Languages (3)*, pages 215–267. Springer, 1997.
13. David Henry Haussler. *Insertion and Iterated Insertion as Operations on Formal Languages*. PhD thesis, University of Colorado, 1982.
14. S. Immanuel, Jayasankar S., Gnanaraj Thomas, and Meenakshi Paramasivan. Parallel contextual array insertion deletion P systems and tabled matrix grammars. In *Membrane Computing*, volume 12687 of *Lecture Notes in Computer Science*, pages 46–77, 06 2021.
15. S. James Immanuel, S. Jayasankar, D. Gnanaraj Thomas, Meenakshi Paramasivan, Robinson Thamburaj, and Atulya K. Nagar. Parallel contextual array insertion deletion P systems and Siromoney matrix grammars. *International Journal of Parallel, Emergent and Distributed Systems*, 36:335–358, 2021.
16. S. James Immanuel, D. G. Thomas, Robinson Thamburaj, and A. K. Nagar. Parallel contextual array P systems. In *In the Proceedings of Asian Conference on Membrane Computing ACMC 2014, IEEE Xplore*, pages 1–9, 2014.
17. S. James Immanuel, D. G. Thomas, Robinson Thamburaj, and Atulya K. Nagar. Parallel contextual array insertion deletion P system. In *IWCIA*, volume 10256 of *Lecture Notes in Computer Science*, pages 170–183. Springer, 2017.
18. S. James Immanuel and D.G. Thomas. Parallel contextual array P system with contexts shuffled on trajectories. In *Proceedings of National Conference on Mathematics and Computer Applications, NCMCA 2015*, pages 214–222, 2015.
19. Masami Ito, Lila Kari, and Gabriel Thierrin. Insertion and deletion closure of languages. *Theor. Comput. Sci.*, 183(1):3–19, 1997.
20. Lila Kari. *On Insertion and Deletion in Formal Languages*. PhD thesis, University of Turku, 1991.

21. Lila Kari and Gabriel Thierrin. Contextual insertions/deletions and computability. *Inf. Comput.*, 131(1):47–61, 1996.
22. Shankara Narayanan Krishna and Raghavan Rama. Insertion-deletion P systems. In *DNA*, volume 2340 of *Lecture Notes in Computer Science*, pages 360–370. Springer, 2001.
23. Kamala Krithivasan and Madhu Mutyam. Contextual P systems. *Fundam. Informaticae*, 49(1-3):179–189, 2002.
24. Solomon Marcus. Contextual grammars. *Rev. Roum. Math. Pures Appl.*, 14:1525–1534, 1969.
25. Gheorghe Paun. Computing with membranes. *J. Comput. Syst. Sci.*, 61(1):108–143, 2000.
26. Gheorghe Paun. *Membrane Computing: An Introduction*. Natural computing series. Springer, 2002.
27. Gift Siromoney, Rani Siromoney, and Kamala Krithivasan. Abstract families of matrices and picture languages. *Comput. Graph. Image Process.*, 1(3):284–307, 1972.
28. K. G. Subramanian, Do Long Van, P. Helen Chandra, and Nghiem Do Quyen. Array grammars with contextual operations. *Fundam. Informaticae*, 83(4):411–428, 2008.

Experimenting with Model Learning for Spiking Neural P Systems (Extended Abstract)

Florentin Ipate^{1,3}, Marian Gheorghe², Ionuț Mihai Niculescu³

¹Department of Computer Science
Faculty of Mathematics and Computer Science
University of Bucharest
Str. Academiei 14, Bucharest 010014, Romania
`florentin.ipate@unibuc.ro`

²Department of Computer Science
Faculty of Informatics and Engineering
University of Bradford, West Yorkshire, Bradford BD7 1DP, UK
`m.gheorghe@bradford.ac.uk`

³Faculty of Science
University of Pitești
Str. Târgul din Vale nr. 1, 110040, Pitești, Romania
`ionutmihainiculescu@gmail.com`

Abstract. This paper describes some experiments regarding the construction of cover automaton and Angluin-style model learning from queries, more precisely the L^l algorithm for learning a finite cover automaton, adapted to the more general X-machine model. A spiking neural P system is associated with an X-machine model and consequently the above mentioned construction can be extended to this P system.

1 Introduction

Membrane computing, introduced in [18], is a branch of natural computing inspired by the structure and functioning of the living cells. Its models are called membrane systems or P systems. The field has an initial research monograph [19] and a comprehensive handbook [20], followed by other developments, including both theoretical research and solid applications.

A specific model of membrane computing, called spiking neural P system, is inspired by the neuron cells. The model was introduced in [13] and the key results are presented in a survey paper [22]. A special line of research in membrane systems (many of them with respect to neural P systems) is dedicated to the formal analysis of these systems. This includes formal semantics [4, 5, 9, 10], reversible computation [1, 21, 3], causality [2, 16] and memory associated with these systems [8]. Testing, especially model-based approach using membrane systems, is another type of formal analysis of these systems looking at traces of execution (computation pathways) defined with respect to certain formal principles. This approach is also significant for validating applications of membrane systems.

Model based testing approaches have been introduced and studied for cell-like P systems [11, 14, 12]. Recently a testing approach based on the concept of cover automaton and Angluin-style model learning from queries, adapted to X-machines that are associated with spiking neural P systems, has been considered [15].

The research presented in this work describes some experiments regarding the construction of cover automaton and Angluin-style model learning from queries, more precisely the L^l algorithm for learning a finite cover automaton, adapted to the more general X-machine model. Such an X-machine model is associated with a spiking neural P system.

2 Description of the main results

The key concepts and results used in these experiments have been introduced and studied in [15] and some of them are briefly mentioned below and the experiments made in order to build a cover automaton are presented.

We use spiking neural P system model, as defined in [13, 17], and build an approximation of its finite execution by using a finite cover automaton [6, 7], through a learning algorithm [15]. This investigation is restricted to the case of spiking neural P systems reading binary sequences from the environment. The rules involved in the execution of the system are encoded as functions of an X-machine and the sequence of configurations is described as the sequence of associated processing functions. The labels of these processing functions are attached to the transitions of the cover automaton that will be built using the learning algorithm [15]. The experiments illustrated below use the example provided in [15].

The sequences of processing functions that can be triggered from some initial data values are presented in Table 1. The end of the input string providing two spikes to the input neuron is denoted by θ . These inputs have length up to 4.

Table 2 shows the sequences of processing functions of length up to 4 that can be triggered. These are the sequences of processing functions from Table 1 and their prefixes of length up to 4.

Using the sequences in Table 2, one can construct a deterministic cover finite cover automaton (DFCA) of U_l for $l \leq 4$. For the sake of simplicity, we only provide a DFCA of U_3 , as depicted in Figure 1 (in order to have the model completely defined, a loopback transition labelled by ϕ_6 has been added to q_5).

The observation table built according to the learning algorithm described in [15] is inserted here. Please note that the computations executed by the spiking neural P system are obtained as executions of an equivalent kernel P system using kPWorkbench and the finite cover automaton is generated by implementing the learning algorithm presented in [15].

Input sequence applied	Functions triggered
no input	ϕ_6
θ	$\phi_1 \phi_{4,1} \phi_{5,1}$
0θ	$\phi_1 \phi_{2,1} \phi_{4,2} \phi_{5,2}$
1θ	$\phi_1 \phi_{2,2} \phi_{4,3} \phi_{5,2}$
00θ	$\phi_1 \phi_{2,1} \phi_{3,1} \phi_{4,2} \phi_{5,2}$
01θ	$\phi_1 \phi_{2,1} \phi_{3,3} \phi_{4,3} \phi_{5,2}$
10θ	$\phi_1 \phi_{2,2}, \phi_{3,2} \phi_{4,2} \phi_{5,2}$
11θ	$\phi_1 \phi_{2,2} \phi_{3,4} \phi_{4,3} \phi_{5,2}$
000θ	$\phi_1 \phi_{2,1} \phi_{3,1} \phi_{3,1} \phi_{4,2} \phi_{5,2}$
001θ	$\phi_1 \phi_{2,1} \phi_{3,1} \phi_{3,3} \phi_{4,3} \phi_{5,2}$
010θ	$\phi_1 \phi_{2,1} \phi_{3,3} \phi_{3,2} \phi_{4,2} \phi_{5,2}$
011θ	$\phi_1 \phi_{2,1} \phi_{3,3} \phi_{3,4} \phi_{4,3} \phi_{5,2}$
100θ	$\phi_1 \phi_{2,2} \phi_{3,2} \phi_{3,1} \phi_{4,2} \phi_{5,2}$
101θ	$\phi_1 \phi_{2,2} \phi_{3,2} \phi_{3,3} \phi_{4,3} \phi_{5,2}$
110θ	$\phi_1 \phi_{2,2} \phi_{3,4} \phi_{3,2} \phi_{4,2} \phi_{5,2}$
111θ	$\phi_1 \phi_{2,2} \phi_{3,4} \phi_{3,4} \phi_{4,3} \phi_{5,2}$

Table 1. Input sequences and sequences of functions triggered

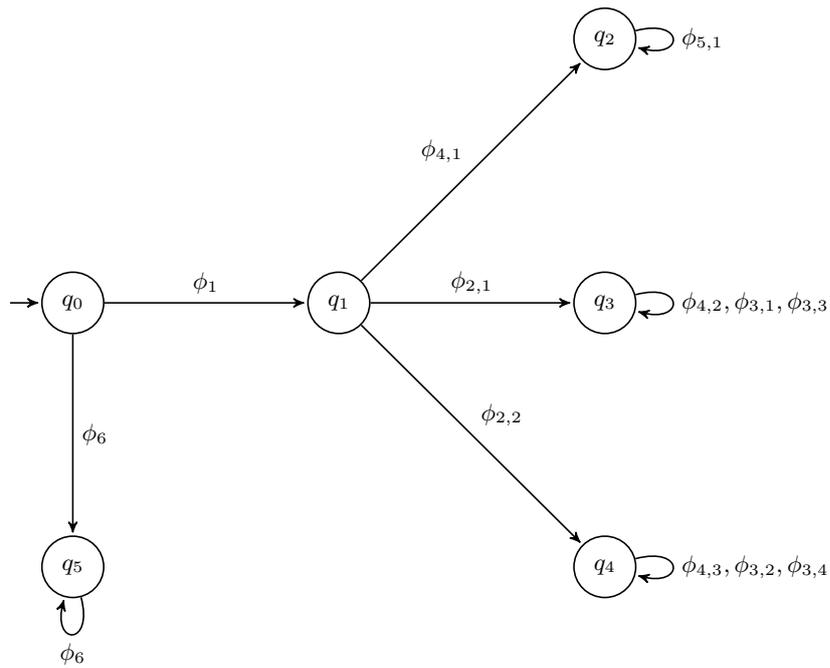


Fig. 1. State transition diagram of a DFCA of U_3

Length	Sequences triggered
1	ϕ_6 ϕ_1
2	$\phi_1 \phi_{4,1}$ $\phi_1 \phi_{2,1}$ $\phi_1 \phi_{2,2}$
3	$\phi_1 \phi_{4,1} \phi_{5,1}$ $\phi_1 \phi_{2,1} \phi_{4,2}$ $\phi_1 \phi_{2,1} \phi_{3,1}$ $\phi_1 \phi_{2,1} \phi_{3,3}$ $\phi_1 \phi_{2,2} \phi_{4,3}$ $\phi_1 \phi_{2,2} \phi_{3,2}$ $\phi_1 \phi_{2,2} \phi_{3,4}$
4	$\phi_1 \phi_{2,1} \phi_{4,2} \phi_{5,2}$ $\phi_1 \phi_{2,1} \phi_{3,1} \phi_{4,2}$ $\phi_1 \phi_{2,1} \phi_{3,1} \phi_{3,1}$ $\phi_1 \phi_{2,1} \phi_{3,1} \phi_{3,3}$ $\phi_1 \phi_{2,1} \phi_{3,3} \phi_{4,3}$ $\phi_1 \phi_{2,1} \phi_{3,3} \phi_{3,2}$ $\phi_1 \phi_{2,1} \phi_{3,3} \phi_{3,4}$ $\phi_1 \phi_{2,2} \phi_{4,3} \phi_{5,2}$ $\phi_1 \phi_{2,2} \phi_{3,2} \phi_{4,2}$ $\phi_1 \phi_{2,2} \phi_{3,2} \phi_{3,1}$ $\phi_1 \phi_{2,2} \phi_{3,2} \phi_{3,3}$ $\phi_1 \phi_{2,2} \phi_{3,4} \phi_{4,3}$ $\phi_1 \phi_{2,2} \phi_{3,4} \phi_{3,2}$ $\phi_1 \phi_{2,2} \phi_{3,4} \phi_{3,4}$

Table 2. Sequences of functions triggered of length up to 4

Table 3: Observation table.

T	ϵ	$\phi_{2,1}$	$\phi_{3,1}$	$\phi_{3,2}$	$\phi_{5,1}$	ϕ_6	$\phi_6 \phi_{2,1}$	$\phi_6 \phi_{3,1}$	$\phi_6 \phi_{3,2}$	$\phi_6 \phi_{5,1}$	$\phi_6 \phi_6$	$\phi_6 \phi_6 \phi_{2,1}$	$\phi_6 \phi_6 \phi_{3,1}$	$\phi_6 \phi_6 \phi_{3,2}$	$\phi_6 \phi_6$
ϵ	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
ϕ_1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$\phi_{2,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\phi_1 \phi_{2,1}$	1	0	1	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1 \phi_{2,2}$	1	0	0	1	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1 \phi_{4,1}$	1	0	0	0	1	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1 \phi_{2,1} \phi_{3,1}$	1	0	1	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1 \phi_{2,1} \phi_{4,2}$	1	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1 \phi_{2,2} \phi_{3,2}$	1	0	1	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1 \phi_{4,1} \phi_{5,1}$	1	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1 \phi_{2,1} \phi_{4,2} \phi_{5,2}$	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_{2,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\phi_{3,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Continued on next page

Table 3 – Continued from previous page

T	ϵ	$\phi_{2,1}$	$\phi_{3,1}$	$\phi_{3,2}$	$\phi_{5,1}$	ϕ_6	$\phi_6\phi_{2,1}$	$\phi_6\phi_{3,1}$	$\phi_6\phi_{3,2}$	$\phi_6\phi_{5,1}$	$\phi_6\phi_6$	$\phi_6\phi_6\phi_{2,1}$	$\phi_6\phi_6\phi_{3,1}$	$\phi_6\phi_6\phi_{3,2}$	$\phi_6\phi_6$
$\phi_{3,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\phi_{3,3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\phi_{3,4}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\phi_{4,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\phi_{4,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\phi_{4,3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\phi_{5,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\phi_{5,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ϕ_6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\phi_1\phi_1$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1\phi_{3,1}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1\phi_{3,2}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1\phi_{3,3}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1\phi_{3,4}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1\phi_{4,2}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1\phi_{4,3}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1\phi_{5,1}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1\phi_{5,2}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1\phi_6$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_1$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_{2,1}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_{2,2}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_{3,1}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_{3,2}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_{3,3}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_{3,4}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_{4,1}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_{4,2}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_{4,3}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_{5,1}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_{5,2}$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_{2,1}\phi_6$	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_1$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{2,1}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{2,2}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,2}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,3}$	1	0	0	1	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,4}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{4,1}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{4,3}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{5,1}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{5,2}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_6$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1

Continued on next page

Table 3 – *Continued from previous page*

T	ϵ	$\phi_{2,1}$	$\phi_{3,1}$	$\phi_{3,2}$	$\phi_{5,1}$	ϕ_6	$\phi_6\phi_{2,1}$	$\phi_6\phi_{3,1}$	$\phi_6\phi_{3,2}$	$\phi_6\phi_{5,1}$	$\phi_6\phi_6$	$\phi_6\phi_6\phi_{2,1}$	$\phi_6\phi_6\phi_{3,1}$	$\phi_6\phi_6\phi_{3,2}$	$\phi_6\phi_6$
$\phi_1\phi_{2,2}\phi_1$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,2}\phi_{2,1}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,2}\phi_{2,2}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,2}\phi_{3,1}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,2}\phi_{3,3}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,2}\phi_{3,4}$	1	0	0	1	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,2}\phi_{4,1}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,2}\phi_{4,2}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,2}\phi_{4,3}$	1	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,2}\phi_{5,1}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,2}\phi_{5,2}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,2}\phi_6$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_1$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_{2,1}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_{2,2}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_{3,1}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_{3,2}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_{3,3}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_{3,4}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_{4,1}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_{4,2}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_{4,3}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_{5,2}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{4,1}\phi_6$	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_1$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_{2,1}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_{2,2}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_{3,1}$	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_{3,2}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_{3,3}$	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_{3,4}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_{4,1}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_{4,2}$	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_{4,3}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_{5,1}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_{5,2}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{3,1}\phi_6$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{4,2}\phi_1$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{4,2}\phi_{2,1}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{4,2}\phi_{2,2}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{4,2}\phi_{3,1}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{4,2}\phi_{3,2}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{4,2}\phi_{3,3}$	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Continued on next page

Table 3 – *Continued from previous page*

T	ϵ	$\phi_{2,1}$	$\phi_{3,1}$	$\phi_{3,2}$	$\phi_{5,1}$	ϕ_6	$\phi_6\phi_{2,1}$	$\phi_6\phi_{3,1}$	$\phi_6\phi_{3,2}$	$\phi_6\phi_{5,1}$	$\phi_6\phi_6$	$\phi_6\phi_6\phi_{2,1}$	$\phi_6\phi_6\phi_{3,1}$	$\phi_6\phi_6\phi_{3,2}$	$\phi_6\phi_6$
$\phi_1\phi_{2,1}\phi_{4,2}\phi_{5,2}\phi_{5,2}$	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$\phi_1\phi_{2,1}\phi_{4,2}\phi_{5,2}\phi_6$	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

References

1. Agrigoroaiei, O., Ciobanu, G.: Reversing computation in membrane systems. *J. Log. Algebraic Methods Program.* **79**(3–5), 278–288 (2010)
2. Agrigoroaiei, O., Ciobanu, G.: Quantitative causality in membrane systems. In: *Proceedings of the 12th International Conference on Membrane Computing, CMC’12*, pp. 62–72. Springer-Verlag (2011)
3. Aman, B., Ciobanu, G.: Reversible computation in nature inspired rule-based systems. *J. Membr. Comput.* **2**(4), 246–254 (2020)
4. Andrei, O., Ciobanu, G., Lucanu, G.: A rewriting logic framework for operational semantics of membrane systems. *Theor. Comput. Sci.* **373**(3), 163–181 (2007)
5. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: Compositional semantics of spiking neural P systems. *J. Log. Algebraic Methods Program.* **79**(6), 304–316 (2010)
6. Câmpeanu, C., Sântean, N., Yu, S.: Minimal cover-automata for finite languages. In: *Revised Papers from the Third International Workshop on Automata Implementation, WIA ’98*, pp. 43–56. Springer-Verlag (1999)
7. Câmpeanu, C., Sântean, N., Yu, S.: Minimal cover-automata for finite languages. *Theor. Comput. Sci.* **267**(1-2), 3–16 (2001)
8. Ciobanu, G., Pinna, G.: Memory associated with membrane systems. *J. Membr. Comput.* **2**(3), 116–132 (2021)
9. Ciobanu, G., Todoran, E.N.: Denotational semantics of membrane systems by using complete metric spaces. *Theor. Comput. Sci.* **701**, 85–108 (2017)
10. Ciobanu, G., Todoran, E.N.: A semantic investigation of spiking neural P systems. In: *Proceedings of the 19th International Conference on Membrane Computing, CMC’19*, pp. 108–130. Springer-Verlag (2018)
11. Gheorghe, M., Ipate, F.: On testing P systems. In: *Membrane Computing*, pp. 204–216. Springer-Verlag (2009)
12. Gheorghe, M., Ipate, F., Konur, S.: Testing based on identifiable P systems using cover automata and X-machines. *Inf. Sci.* **372**, 565–578 (2016). <https://doi.org/10.1016/j.ins.2016.08.028>. URL <https://doi.org/10.1016/j.ins.2016.08.028>
13. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. *Fundam. Informaticae* **71**(2-3), 279–308 (2006). URL <http://content.iospress.com/articles/fundamenta-informaticae/fi71-2-3-08>
14. Ipate, F., Gheorghe, M.: Finite state based testing of P systems. *Natural Computing* **8**(4), 833–846 (2009)
15. Ipate, F., Gheorghe, M.: A model learning based testing approach for spiking neural P systems. *Theor. Comput. Sci.* (2021)
16. Pagliarini, R., Agrigoroaiei, O., Ciobanu, G., Manca, V.: An analysis of correlative and static causality in P systems. In: *Proceedings of the 13th International Conference on Membrane Computing, CMC’13*, pp. 323–341. Springer-Verlag (2012)

17. Păun, A., Păun, Gh.: Small universal spiking neural P systems. *Biosyst.* **90**(1), 48–60 (2007). <https://doi.org/10.1016/j.biosystems.2006.06.006>. URL <https://doi.org/10.1016/j.biosystems.2006.06.006>
18. Păun, Gh.: Computing with membranes. *Journal of Computer and System Sciences* **61**(1), 108–143 (2000)
19. Păun, Gh.: *Membrane Computing: An Introduction*. Springer-Verlag (2002)
20. Păun, Gh., Rozenberg, G., Salomaa, A.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc. (2010)
21. Pinna, G.: Reversing steps in membrane systems computations. In: *Proceedings of the 18th International Conference on Membrane Computing, CMC'18*, pp. 245–261. Springer-Verlag (2017)
22. Rong, H., Wu, T., Pan, L., Zhang, G.: Spiking neural P systems theoretical results and applications. In: *Enjoying Natural Computing*, pp. 258–268. Springer-Verlag (2018)

Extended Rules and Heterogeneous Derivation Modes in Spiking Neural P Systems with Stochastic Application of Rules

Prometheus Peter L. Lazo, Ren Tristan A. De La Cruz, Ivan Cedric H. Macababayao, Francis George C. Cabarle*, Henry N. Adorna

Algorithms & Complexity, Dept. of Computer Science,
University of the Philippines Diliman, Diliman 1101 Quezon City, Philippines.

Abstract. Spiking neural P systems with stochastic application of rules (\star SN P) are a class of SN P systems which introduce a stochastic process *a priori* to rule application. With this feature, \star SN P systems allows two behaviors which are (1) stochastic selection of rules rather than non-deterministic and (2) asynchronous rule application for rules with rule application probabilities that are < 1 . In the investigation of computational universality of \star SN P systems, the construction of \star SN P register machine modules ADD, SUB, and FIN required auxilliary neurons with rules that have an application probability of 1. This study investigates the restriction of rule application probabilities in \star SN P systems to be < 1 . This study argues that imposing the restriction is nontrivial and the introduction of extended rules (rules that allow the transmission of multiple spikes) is needed to address the computational universality when the rule application probability is strictly between 0 and 1. The study then shows that \star SN P systems using extended rules is computationally universal. To further improve the optimizations of the resources used when showing universality, heterogeneous derivation modes in rules and in neurons are also introduced such that a rule can be applied exhaustively or not. This study then shows universality while using heterogeneous derivation modes in \star SN P systems as well as the reduction of resources consumed.

1 Introduction

Spiking neural P (SN P) systems constitute a class of P systems that introduces the idea of spiking neurons [9]. The neurons in SN P systems consume and may send signals known as spikes to other SN P neurons based on rules defined per neuron. In the field of membrane computing, these rules have been a subject of various studies. This study continues the investigation on *a priori* stochastic rule application.

An SN P system variant called spiking neural P systems with stochastic application of rules (\star SN P systems) was introduced in [12] where the application of

* corresponding author fccabarle@up.edu.ph

rules is stochastic. Apart from the stochastic selection of which rule to be applied among multiple applicable rules, the variant has also inadvertently introduced a asynchronous behavior to rule application. While there are other stochastic SN P systems such as the stochastic spiking neural P system by [2], and extended spiking neural P system by [20], this study focuses on \star SN P systems specifically.

The proof of computational universality of \star SN P systems involve the simulation of a register machine and designing three \star SN P modules that simulate the ADD, SUB, and FIN instructions [12]. In the design of the modules, the idea of low-pass and high-pass neurons from [18] are implemented in most of the neurons. Some \star SN P neuron rules in the design used a rule application probability of 1. This study investigates the open problem stated in the work regarding the possibility of implementing a \star SN P system where all rules have a rule application probability of < 1 . The use of extended rules allowed reduction of neuron in the register machine modules as it removed the need for intermediary neurons to duplicate spikes. In line with the optimizations of rule application probabilities, further optimizations to the resources used in the computational completeness were investigated. This led into introducing heterogeneous derivation modes for neurons or rules to reduce the number of rules used.

In Section 2 prerequisite concepts, definitions, and constructs for this study are provided. Section 3 details existing work on spiking neural P systems with extended rules and spiking neural P systems with varying derivation modes. Section 4 presents how the problem is analyzed, how the solution is synthesized, and a comparison of the proposed model to the related work. Section 5 enumerates the opportunities for improvement and the conclusion.

2 Preliminaries

This section requires basic familiarity with *formal language* and *membrane computing* as discussed in [9].

Consider an alphabet V . Then V^* is the free monoid generated by V with respect to the concatenation operation and the identity λ (the empty string). Then $V^+ = V^* - \lambda$ denotes the set of all nonempty strings over V . The singleton alphabet $V = \{a\}$ is written as a^* and a^+ rather than $\{a\}^*$ and $\{a\}^+$. The cardinality of a string $x \in V^*$ is denoted by $|x|$.

RE denotes the class of recursively enumerable languages while NRE denotes the class of Turing computable sets of natural numbers.

2.1 Register Machines

A *register machine* is a computing model of the form $M = (m, H, l_0, l_h, I)$ where:

1. m is the number of registers $\{r_1, \dots, r_m\}$ with r_1 as the output register;
2. $H = \{l_0, \dots, l_h\}$ is a finite set of instruction labels where each label in H labels only one instruction;
3. l_0 is the start label (labeling an ADD instruction);

4. l_h is the halt label (set to instruction HALT);
5. I is a finite set of instructions of the following forms:
 - (a) $l_1 : (ADD(r), l_2, l_3)$, which adds 1 to register r then nondeterministically executes instruction l_2 or l_3 ;
 - (b) $l_1 : (SUB(r), l_2, l_3)$, which if $r > 0$ then subtracts 1 to r and executes instruction l_2 , else, executes instruction l_3 ;
 - (c) $l_h : (HALT)$ which is the halt instruction.

In the *generating mode*, the computation of M begins with all registers being empty. It is assumed, without loss of generality, that the output register r_1 is never decremented throughout the computation and all registers, except for r_1 are empty by the end of the computation. The computation follows the labels selected in an instruction to determine which instruction to execute next, beginning with the start instruction l_0 . The computation halts when a halt instruction is reached and the number stored in output register r_1 is said to be generated or computed by M . The nondeterminism in the ADD instruction implies multiple halting computations in M .

A register machine M can also work in *accepting mode* such that in the initial configuration, a number n is set in r_1 and all other registers are empty. If the computation of M halts based on an accepting mode initial configuration, then n is *accepted* by M . The register machine accepting mode is universal for both non-deterministic and deterministic acceptance. In deterministic acceptance, ADD instructions have the form $l_1 : (ADD(r), l_2, l_3)$ with $l_2 = l_3$, which implies passing to the next instruction without any choice involved and can essentially be written in the form $l_1 : (ADD(r), l_2)$.

$N(M)$ denotes the set of all natural numbers computed by M . The class of nondeterministic register machines, denoted by RM_{NDET} , is known to characterise NRE [14].

2.2 SN P systems

From [9], a *spiking neural P system* of degree $m \geq 1$ is of the form

$$\Pi = (O, \sigma_1, \dots, \sigma_m, syn, i_0)$$

where:

1. $O = a$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \dots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where:
 - (a) $n_i \geq 0$ is the *initial number of spikes* in the neuron σ_i ;
 - (b) R_i is a finite set of *rules* of the following two forms:
 - (1) $E/a^r \rightarrow a; d$, where E is a regular expression over O , $r \geq 1$, and $d \geq 0$;
 - (2) $a^s \rightarrow \lambda$ for some $s \geq 1$ with the restriction that $a^s \notin L(E)$ for any rule $E/a^r \rightarrow a; d$ of type (1) from R_i ;
3. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* among neurons);

4. $i_0 \in \{1, 2, \dots, m\}$ indicates the *output neuron*.

SN P Systems function synchronously, i.e. if a neuron can fire a rule in each step, it must do so. In this way, the entire system uses a global clock to index the time.

The rules of form (1) are called *firing rules*. A rule of form $a^r/a^r \rightarrow a; d$ can simply be written as $a^r \rightarrow a; d$. The notation $; d$ can be omitted if $d = 0$. At some step q , a neuron σ fires when it applies a firing rule $E/a^r \rightarrow a; d$ if σ contains n spikes such that $a^n \in L(E)$ and $n \geq r$. Neuron σ releases its spike at step $q + d$ to every neuron with a synapse from σ . This means that if $d = 0$ the spike leaves σ immediately while $d > 0$ means the spike will leave σ after d steps. If $d > 0$, then σ is *closed* within steps q and $q + d - 1$ where it cannot receive new spikes and consequently fire. Starting at step $q + d$ the neuron becomes *open*, i.e. it can receive new spikes and emit its spike. Any spike received by σ while it is closed is lost, i.e. the spike is never received by σ .

The rules of form (2) are called *forgetting rules*. When such rules are applied, that is when the number of spikes in a neuron $n = s$, then s spikes are removed from the neuron, i.e. they are lost, and no spike is produced. Neurons are allowed to have two firing rules $E_1/a^{r_1} \rightarrow a; d_1, E_2/a^{r_2} \rightarrow a; d_2$ with $L(E_1) \cap L(E_2) \neq \emptyset$. Nondeterministic application of rules occur when $a^{n_i} \in L(E_1) \cap L(E_2)$, i.e. the neuron nondeterministically chooses which rule it fires. A *coherence condition* forbids a firing rule and a forgetting rule to be applicable with the same number of spikes in a neuron.

A *configuration* is the distribution of spikes and the open/closed states of all neurons. The initial configuration is $C_0 = \langle n_1, n_2, \dots, n_m \rangle$ corresponding to initial spikes of neurons $\sigma_1, \sigma_2, \dots, \sigma_m$, with all neurons open. Given two configurations C_1 and C_2 , a *transition*, denoted by $C_1 \Longrightarrow C_2$, is the step of passing from C_1 to C_2 . A *computation* is then a sequence of transitions starting in the initial configuration. A *halting configuration* is then reached when there are no more rules that are applicable.

There are several ways for a result to be obtained with a computation. From the convention used in membrane computing where the number of objects in the membrane is counted, the number of spikes in the output neurons of SN P systems can be the result of a computation. Another result can be a finite or infinite binary string where each bit is retrieved per computation step from the output neuron with 1 associated to the neuron spiking and 0 if otherwise. Another result is the set to time steps when the output neuron released spikes i.e. if the output neuron releases spikes at time steps 4 and 7 before halting, then the result of the computation is the set $\{4, 7\}$. Lastly, the closest to the biological way of obtaining a result is the *time interval between the two spikes* of the output neuron

In accepting mode, SN P systems use i_0 as the input neuron rather than the output neuron. The input neuron then receives two spikes from the environment in two time steps and the interval between the two steps is the input value. Then if the computation halts, the value is accepted.

$N_2(\Pi)$ denotes the set of all natural numbers computed by Π , with subscript 2 denoting the way¹ the result of a computation is defined. Then the notation $Spik_2P_m(rule_k, cons_p, forg_q)$ denotes the family of all sets $N_2(\Pi)$ computed as previously discussed by SN P systems with at most $m \geq 1$ neurons, using at most $k \geq 1$ rules in each neuron, with all firing rules $E/a^r \rightarrow a; d$ having $r \leq p$, and all forgetting rules $a^s \rightarrow \lambda$ having $s \leq q$. When one of the parameters m, k, p, q is not bounded, then it is replaced with $*$. From the initial paper in [9] it is known that SN P systems are computationally universal, i.e. they simulate register machines, and hence also characterise NRE .

2.3 Constructing SN P Systems

From the formal definition of SN P systems, various membrane computing studies have investigated its features and its effect when simulating register machines to prove computational universality. Introducing new features have created new SN P system variants or extensions to address the limitations of the original formal definition. A framework was proposed in [17] to capture these features under investigation and define them in a formal manner. On the other hand, the study of normal forms investigated the removal or restriction of features while maintaining the same computational power. In [8], it is proven that the removal of delays and forgetting rules does not affect the universality of SN P systems while in [19], it is shown that neurons possessing the same rules is still computationally complete. The complexity, however, is shifted to the remaining features that are not removed or restricted.

From [8], where delays are not required in constructing SN P systems, shows that using a buffer neuron $\sigma = (0, \{a \rightarrow a\})$ replaces a delay. This buffer neuron can also be used as intermediate neurons to duplicate spikes when a pair or more of this buffer neuron receives a spike from a similar source and then sends its spike to a similar destination. Meanwhile, forgetting rules remain useful in neurons that serve as gates that validate the number of spikes it receives before taking action and ensuring zero spikes left in the system at the end of a computation. This is done in [18] and [15] using low-pass and high-pass neurons.

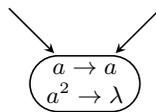


Fig. 1. SN P Low-Pass Neuron Example

A low-pass neuron is illustrated in Figure 1. Low-pass neurons only fire when it contains one spike and forgets when otherwise. When constructing a low-pass neuron, for each number of spikes greater than 1 that the low-pass neuron

¹ from [9], the subscript 2 denotes the time interval way to obtain a result

can receive, a respective forgetting rule is declared in its set of rules. Low-pass neurons are useful in situations where interrupting the release of spikes is needed.

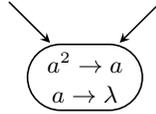


Fig. 2. SN P High-Pass Neuron Example

A high-pass neuron is illustrated in Figure 2. High-pass neurons work contrary to low-pass neurons where it only spikes when receiving the maximum possible spikes at a time step and forgets when otherwise. In constructing a high-pass neuron, for each number of spikes less than the maximum number of spikes it can receive at a time step, a respective forgetting rule is declared in its set of rules. A high pass neuron is useful in situations where rather than interruption is needed, an additional spike is needed to confirm the release of spikes.

3 Related Work

This study deals with several SN P system variants that introduced different unique features to the SN P system. The primary variant that this study focuses in is the spiking neural P systems with stochastic application of rules (\star SN P systems) by [12] which introduces stochastic rule application. This variant was introduced with the goal of replacing the nondeterministic rule selection with probabilities by specifically introducing the notation for rule application probability (p) to the forms of the firing and forgetting rules (the forms $E/a^r \rightarrow a; d$ and $a^s \rightarrow \lambda$ become $(p)E/a^r \rightarrow a; d$ and $(p)a^s \rightarrow \lambda$ for firing and forgetting rules respectively). The introduction of stochastic rule applications means that with this variant an applicable rule may or may not be applied depending on probabilities.

The next variant is SN P systems with extended rules. From the definition of SN P systems discussed, the introduction of extended rules was suggested to address the need for pairs of intermediate neurons when duplicating a spike [16] and producing strings over arbitrary alphabets [6]. From [7], SN P systems with extended rules modify the forms of the rules in the original SN P system formal definition to a single form $E/a^r \rightarrow a^s; d$ where E is a regular expression over a and $r \geq 1, s \geq 0$, with the restriction $r \geq s$, and d being entirely optional. The unified form of extended rules for both firing and forgetting rules from the previously defined two separate forms now means that firing and forgetting rules are determined whether $s > 0$ or $s = 0$ otherwise. The conditional omission of notations E and $; d$ still apply in this form. In this form, a forgetting rule $E/a^r \rightarrow a^s; d$ with $s = d = 0$ is written in the form $E/a^r \rightarrow \lambda$. The usage of

rules in this form is similar with standard rules. However, the combined form of firing and forgetting rules using extended rules allows a neuron to forget r spikes, which can be less than the number of spikes the neuron contains, unlike standard rules where all spikes are forgotten. The coherence condition from the original SN P system rules is removed, which allows a firing rule and a forgetting rule to be applicable with the same number of spikes in a neuron. This extended rules then allows a neuron to send s spikes at a computational step when $s > 0$, thus eliminating the need for intermediate neurons to duplicate a spike and produce strings over arbitrary alphabets. Lastly, SN P Systems with extended rules is also computationally complete.

From SN P system with extended rules, the introduction of asynchronous rule application and different derivation modes were investigated in related work. Driven by the neuro-biological point of view that it is rather natural to consider non-synchronized systems, asynchronous spiking neural P systems are then introduced by [4]. From SN P systems with extended rules, asynchronous SN P systems add that at a given time step, any neuron with an applicable rule is free to apply that rule or not. If an applicable rule is not applied, the neuron remains open to receive spikes at that step. Hence in the next step, if the number of spikes in that neuron changes, then it is possible for a different rule to be applicable or no rules being applicable. The change in rule application behavior entails that it is no longer possible to use two intermediate neurons to duplicate a spike as there is no guarantee of synchronized delivery from the intermediate neurons; hence, the use of extended rules. While it is proven in [4] that asynchronous SN P systems using extended rules are computationally universal, universality when using standard rules remains an open problem [3]. On derivation modes, [10] introduced SN P systems with exhaustive use of rules. This variant enables rules to be applied multiple times at a time step as long as the neuron contains enough spikes for the rule to consume. This allows neurons to send a large number of spikes at a single time step. From SN P systems with exhaustive use of rules, [11] developed the spiking neural P systems with generalized use of rules, which allows a nondeterministic number of times a rule is applied in a neuron, provided that the neuron contains enough spikes. This generalization allows varying numbers of spikes released by a neuron, unlike SN P systems with exhaustive use of rules which applies a rule with the maximum possible number of times at a step.

This study also subjects two stochastic SN P variants for comparison. The stochastic spiking neural P (SSN P) system of [5] introduces random delays to SN P systems, replacing the static optional delay of the classic model. Another stochastic SN P system is the extended spiking neural P system (ESNPS) of [20] which is an instance of a stochastic SN P model that introduces stochastic rule application to the output neurons. This model is used as a component of the optimization spiking neural P system (OSNPS) which is mainly used to solve combinatorial optimization problems like the knapsack problem.

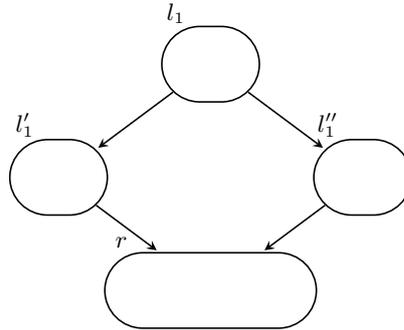


Fig. 3. \star SN P Module ADD neurons that increment a register

4 Results

In \star SN P systems, rule application probabilities of 1 are attributed to rules that need to be applied synchronously. Two situations force neurons to have rule probabilities restricted to 1.

The first situation is spike duplication. An example of this is illustrated in Figure 3 where the \star SN P ADD module increments the register. The conventional representation of a value n in a register is by $2n$ spikes in its respective neuron [1][2][5][7][9][12]. From the definition of \star SN P firing rules, the form $(p)E/a^r \rightarrow a; d$ only allows a neuron to send one spike. As described in [16], pairs of intermediary neurons is then required for duplicating the spike to increment the register. In \star SN P systems a rule application probability of < 1 to any of the intermediary neurons can result to an asynchronous transmission of spikes wherein instead of receiving two spikes at a time step, it receives one spike at two time steps. The register would then behave differently as it would decrement its value, or start emptying its contents in the case of the output register. This also applies to \star SN P low-pass (with a forgetting rule) and high-pass neuron as both need to receive two spikes synchronously to forget or fire respectively.

From [16], the pair of intermediary neurons suggest an extension of the rules of SN P systems to allow rules of the form $E/a^r \rightarrow a^s; d$ where $r \geq s$. This then allows neurons to send more than one spike in a computational step rather than using intermediary neurons to duplicate a spike. This extended rule can then generalize both firing and forgetting rules as an extended rule with $s = 0$ represents the forgetting of spikes [7]. Allowing this extended rules to \star SN P systems would then eliminate the need for intermediary neurons to duplicate a spike and have the source neuron spike send more than one spike instead. The firing rule of that neuron could then have have a application probability of < 1 .

The usage of extended rules remain sequential. Variant modes in the usage of extended rules have been proposed with the introduction of extended rules which are worth considering in this study. An exhaustive use of rules was proposed by [10] in which an applicable rule is applied as many times as possible for the

number of spikes in a neuron. This results in the arbitrarily large number of spikes produced at a time. Combining both sequential and exhaustive usage of extended rules, a generalized use of rules was presented by [11] wherein rather than a single application of a rule or the multiple applications of a rule until no longer applicable, the rule can be applied any possible number of times between the minimum and maximum, nondeterministically chosen.

The introduced two modes of extended rule application may be useful in specific scenarios, but may cause further restrictions when designing \star SN P systems. The exhaustive use of rules minimizes the opportunity to have a neuron maintain a number of spikes such that it can initially receive a number of spikes, then receive another set for a rule to then be applied, or reversely use a rule to consume a partial amount of its spikes so that the remaining spikes can be consumed in future time steps. On the other hand, the generalized use of extended rules introduces another point of nondeterminism in the system, which the study is not interested in especially with the objective of introducing stochastic processes in the nondeterminism of SN P systems. Furthermore this study is more interested in utilizing the number of spikes sent by a neuron without the conditions that relate to multiple consumption of a set of spikes at a step.

The second situation involves the output neuron i_0 . The result of a computation in \star SN P systems is based on the interval between two spikes of i_0 . If the rule application probability of the firing rule of output neuron is < 1 , then it renders the use of the time interval as an output inapplicable as the stochastic neuron may or may not fire when it has to in order to maintain the correct time interval. From [9], and used in [4], the result could instead be interpreted as the number of spikes that the output neurons have sent to the environment. However, this only considers results in the halting configuration.

A \star SN P system using extended rules changes the forms of the rules to a single form

$$(p)E/a^r \rightarrow a^s; d$$

which is the form of the extended rules with the notation (p) that indicates the rule application probability of the rule.

Given \star SN P systems using extended rules Π , $N_c(\Pi)$ denotes the set of all natural numbers computed by Π , with subscript c denoting the way the result of a computation is defined as the count of spikes sent to the environment. $Spik_c P_m(prob_\rho rule_k^x, cons_p, forg_q)$ denotes the family of all sets $N_c(\Pi)$ computed with at most $m \geq 1$ neurons, using at most $k \geq 1$ rules, using rule application probabilities within the interval ρ and using at most $k \geq 1$ rules in each neuron, with all firing rules $E/a^r \rightarrow a; d$ having $r \leq p$, and all forgetting rules $a^s \rightarrow \lambda$ having $s \leq q$. When one of the parameters m, k, ρ, p, q is not bounded, then it is replaced with $*$. A superscript x after *rule* in the notation indicates the use of extended rules.

There exists an SN P system using extended rules that exist for every \star SN P system using extended rules where $p = 1$ for all rule application probabilities. In addition, there exists an asynchronous SN P system for every \star SN P system

using extended rules where $p = 0.5$ for all rule application probabilities. This section then shows computational completeness of \star SN P systems where p for all rules can be any value between 0 and 1.

4.1 Computational Universality

Theorem 1. $SpikcP_*(prob_{(0,1)}, rule_k^x, cons_p, forg_q) = NRE, \forall k \geq 4, p \geq 8, q \geq 10$

Proof. Construct \star SN P system

$$II = \{\{a\}, \sigma_1, \dots, \sigma_m, syn, i_0\}$$

to simulate register machine M . II consists of three modules which simulate instructions ADD, SUB, and FIN of the register machine. The register values in this simulation are represented by $5n$ spikes rather than the conventional $2n$ spikes. The introduction of extended rules has provided significant optimizations to the design of the modules by leveraging the capability of neurons to produce > 1 spikes at a step. The varying number of outgoing spikes, which this study will refer as *spike strengths*, can then be classified to represent certain outcomes as shown in Table 1. In the construction of the modules, the rule application probabilities should also be greater than 0 so that the rules have the possibility to be applied, even despite setting a low probability. Since all rules can have application probabilities between 0 and 1, the notation $0 < p < 1$ is then abbreviated to p .

Table 1. Classification of spikes strengths received by label neurons

Spike Strength	Indication
5	Execute label instruction
4	Do not execute label instruction
3	Wake SUB module, the firing register is not caused by interference
2	Register interference - successful decrement of register
1	Register interference - unsuccessful decrement of register

The \star SN P ADD module is illustrated in Figure 4. All rule probabilities can be set to any value between 0 and 1, except in σ_1 where the probabilities are set to 0.5 to replicate the likelihood of the nondeterministic selection between two applicable rules. The instruction label neuron l_1 sends 5 spikes to the register neuron r (simulating adding 1 the register) and auxiliary neurons σ_1 and σ_2 . This neuron will then probabilistically select two applicable rules since it received five spikes from l_1 . Two outcomes may occur. If the rule $(0.5)a^5 \rightarrow a^5$ is applied, it will send five spikes to both l_2 and σ_2 . Instruction label neuron l_2 will then fire and σ_2 will forget since it contains ten spikes. Thus simulating the selection of

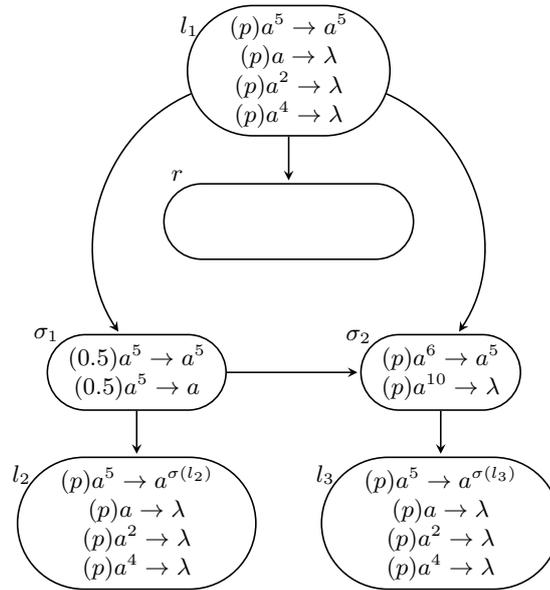


Fig. 4. \star SN P using Extended Rules Module ADD (simulating $l_1 : (\text{ADD}(r), l_2, l_3)$)

instruction label l_2 . On the other hand, if the rule $(0.5)a^5 \rightarrow a$ in σ_1 is applied, it will send a single spike to the l_2 and σ_2 . l_2 will then forget and σ_2 will then send five spikes to l_3 which consequently fires. Thus simulating the selection of instruction label l_3 .

Table 2 shows a simulation of the ADD module that selects l_2 . The simulation shows that only one neuron fires at every time step. $t_x, x \in \{l_1, \sigma_1, \sigma_2, l_2, l_3\}$ indicates the overhead rule application time of neuron x . The simulation table also shows that, if the computation selects l_3 , the computation takes at least one time step longer.

The \star SN P SUB module is illustrated in Figure 5. All rule probabilities can be set to any value between 0 and 1. The instruction label neuron l_1 sends 3 spikes to the register neuron r , auxiliary neuron σ_1 , and instruction label neurons l_2 and l_3 as an act of “waking” the module. The activity of waking the module prevents register interference such that r was used by other SUB modules and fires. If the interference occurs and the module is not awake, then neurons σ_1 and l_2 (neurons connected to r with a synapse) will simply forget the one or two spikes it may receive. On the other hand, if the module is awake, then neurons σ_1 and l_2 already contain three spikes and would behave differently. Two outcomes can occur in when r is decremented. If the value of r , say $n > 0$, then the neuron should contain at $3(5n)$ spikes, 3 of which came from l_1 . l_2 would then probabilistically apply rule $(p)a^3(a^5)^+/a^8 \rightarrow a^2$ which consumes eight spikes to send two spikes to σ_1 and l_2 , leaving it with $5n - 5$ spikes (the three spikes from l_1 was consumed as well). Both having five spikes, l_2 would then fire and σ_1 would

Table 2. A computation in the system from Figure 4

Step Neuron	t	$t + 1 + t_{l_1}$	$t + 2 + t_{\sigma_1}$	$t + 3 + t_{\sigma_2}$	$t + 3 + t_{l_2}$	$t + 4 + t_{l_3}$
l_1	a^5	$(p)a^5 \rightarrow a^5$	—	—	—	—
r	—	a^5	a^5	a^5	a^5	a^5
σ_1	—	a^5	if σ_1 fires a^5 : $(0.5)a^5 \rightarrow a^5$ — else: $(0.5)a^5 \rightarrow a$ —	—	—	—
σ_2	—	a^5	if σ_1 fires a^5 : — a^{10} else: — a^6	$(p)a^{10} \rightarrow \lambda$ — $(p)a^6 \rightarrow a^5$ —	—	—
l_2	—	—	if σ_1 fires a^5 : — a^5 else: — a^4	— a^5 — a^4	$(p)a^5 \rightarrow a^{\sigma(l_2)}$ — $(p)a^4 \rightarrow \lambda$ —	— — — —
l_3	—	—	—	if σ_1 fires a^5 : — — else: — a^5	— — — a^5	— — — $(p)a^5 \rightarrow a^{\sigma(l_2)}$ —

apply rule $(p)a^5 \rightarrow a$ which sends one spike to l_3 . l_3 would then forget since it contains four spikes. Thus simulating a successful decrement on the register and selecting l_2 . On the other hand, if $n = 0$, r would only contain spikes that came from l_1 . The rule $(p)a^3 \rightarrow a$ would then be applied which consumes all spikes in the neuron to send one spike to σ_1 and l_2 leaving the register neuron empty. Both having four spikes in this outcome, l_2 would then forget and σ_1 would apply rule $(p)a^4 \rightarrow a^2$ which sends two spikes to l_3 . l_3 would then fire since it contains five spikes. Thus simulating a failed decrement of a register with a value of 0 and selecting l_3 instead.

Table 3 shows a simulation of the SUB module that decrement a register. Similarly with the ADD module, the simulation table also shows that, if the computation selects l_3 due to a failed decrement, the computation takes at least one time step longer.

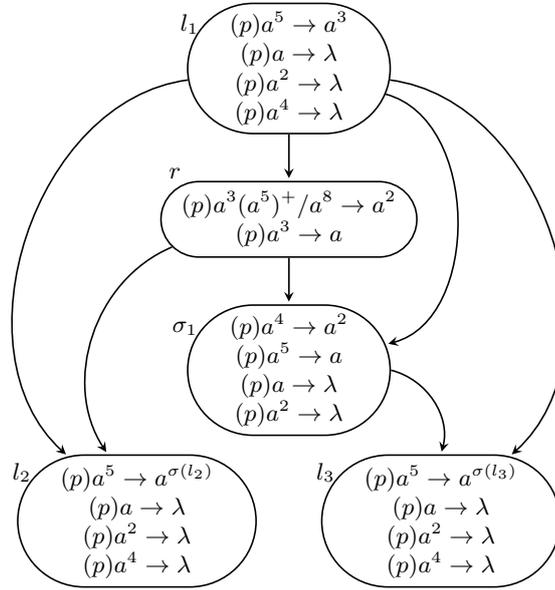


Fig. 5. \star SN P using Extended Rules Module SUB (simulating $l_1 : (\text{SUB}(r), l_2, l_3)$)

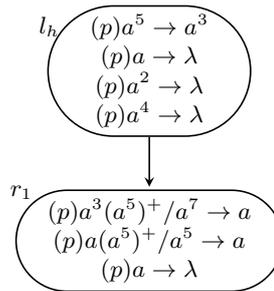


Fig. 6. \star SN P using Extended Rules Module FIN (simulating $l_h : \text{HALT}$)

Table 3. A computation in the system from Figure 5

Step Neuron	t	$t + 1 + t_{l_1}$	$t + 2 + t_r$	$t + 3 + t_{\sigma_1}$	$t + 3 + t_{l_2}$	$t + 4 + t_{l_3}$
l_1	a^5	$(p)a^5 \rightarrow a^3$ —	— —	— —	— —	— —
r	if $n > 0$:	— a^8	$(p)a^3(a^5)^+/a^8 \rightarrow a^2$ —	— —	— —	— —
	else:	— a^3	$(p)a^3 \rightarrow a$ —	— —	— —	— —
σ_1	—	— a^3	if $n > 0$:	$(p)a^5 \rightarrow a$ —	— —	— —
			else:	$(p)a^4 \rightarrow a^2$ —	— —	— —
l_2	—	— a^3	if $n > 0$:	— a^5	$(p)a^5 \rightarrow a^{\sigma(l_2)}$ —	— —
			else:	— a^4	$(p)a^4 \rightarrow \lambda$ —	— —
l_3	—	— a^3	— a^3	if $n > 0$:	— a^4	$(p)a^4 \rightarrow \lambda$ —
				else:	— a^5	$(p)a^5 \rightarrow a^{\sigma(l_3)}$ —

Lastly, the \star SN P FIN module is illustrated in Figure 6. The module is fairly straightforward as it is a pair of neurons from instruction label neuron l_h and the output register neuron r_1 . Like the ADD and SUB module, all rule probabilities can be set to any value between 0 and 1. r_1 serves as the output neuron and when it receives three spikes from l_h , it will apply rule $(p)a^3(a^5)^+/a^7 \rightarrow a$, consume seven spikes to send one spike to the environment, starting the output process. The neuron still contains more than one spike, it will then consume five spikes each step using the rule $(p)a(a^5)^+/a^5 \rightarrow \lambda$ and fire until one spike is left. Finally, $(p)a \rightarrow a$ will be applied to forget the remaining spike, leaving the neuron exhausted of spikes after the process.

Table 4 shows a simulation of the FIN module. The first spike sent to the environment occurred at $t + 2 + t_{r_1}$ and fires again at $t + 3 + t_{r_1}$. With two spikes sent to the environment, the value of 2, is successfully sent to the environment. The computation ends at least one computation step longer than the last fire of the output neuron to forget the single spike left in the output neuron.

Thus, $Spik_cP_*(prob_{(0,1)}, rule_k^x, cons_p, forg_q) = NRE, \forall k \geq 4, p \geq 8, q \geq 10$.

Table 4. A computation in the system from Figure 6

Step Neuron	t	$t + 1 + t_{l_h}$	$t + 2 + t_{r_1}$	$t + 3 + t_{r_1}$	$t + 4 + t_{r_1}$
l_h	a^5	$(p)a^5 \rightarrow a^3$ —	— —	— —	— —
r_1	$\frac{—}{a^{10}}$	$\frac{—}{a^{13}}$	$(p)a^3(a^5)^+/a \rightarrow a^7$ a^6	$(p)a(a^5)^+/a^5 \rightarrow a$ a	$(p)a \rightarrow \lambda$ —

4.2 Comparison with SN P systems, Stochastic SN P variants, and some Non-stochastic SN P variants

Table 5. Summary table of the comparison of \star SN P systems with SN P and other SN P variants

Model	Stochastic	Asynchronous	Extended Rules	Universal	Asynchronous and Universal	# of Neurons per Module	Worst Runtime ² for ADD and SUB	Results obtained before reaching halt
\star SN P systems using extended rules	Yes	Yes	Yes	Yes	Yes	ADD:6, SUB:5, FIN:2	ADD:4, SUB:4	One result, number of spikes
\star SN P systems[12]	Yes	Yes	No	Yes	Unproven	ADD:8, SUB:7, FIN:4	ADD:4, SUB:4	Multiple results, spike interval
SN P systems[9]	No	No	No	Yes	No	ADD:8, SUB:6, FIN:9	ADD:4, SUB:4	Multiple results, spike interval
SN P systems using extended rules[16][7]	No	No	Yes	Yes	No	ADD:7, SUB:5, FIN:3	ADD:4, SUB:4	Multiple results, spike interval
SN P systems with exhaustive use of rules[10]	No	No	Yes	Yes	No	ADD:12, SUB:10, FIN:8	ADD:4, SUB:5	Multiple results, spike interval
SN P systems with generalized use of rules[7]	No	No	Yes	Yes	No	ADD:7, SUB:13, FIN: 3	ADD:4, SUB:5	Multiple results, spike interval
Asynchronous SN P systems[4]	No	Yes	Yes	Yes	Yes	ADD:9, SUB:9, FIN:2	ADD:5, SUB:5	One result, number of spikes
Stochastic SN P systems[5]	Yes	Yes	No	Yes	Unproven	ADD:11, SUB:9, FIN:N/A	ADD:5, SUB:5	One result, number of spikes
Extended SN P system[20]	Yes	No	No ³	Unproven	No	N/A	N/A	Multiple results, spike train ⁴

Table 5 shows the summary of the comparison of \star SN P systems using extended rules with \star SN P systems using standard rules, SN P systems, SN P systems using extended rules, SN P systems with exhaustive use of rules, SN P

systems with generalized use of rules, Asynchronous SN P systems, Stochastic SN P systems, and the Extended SN P system. The \star SN P systems have the advantage of stochasticity in its application of rules only matched by its fellow stochastic SN P models. However, unlike the other stochastic SN P models, \star SN P systems using extended rules can be both asynchronous and computationally complete. A major disadvantage of \star SN P systems using extended rules is the loss of synchronization across its neurons thereby affecting the runtime if the rule application probabilities set are lower. For the runtime comparisons, it is assumed that the reliability of rule application for the asynchronous model is faultless.

SN P Systems With the use of extended rules, the number of neurons in the register machine modules are reduced as well since the use of extended rules removes the need of intermediary neurons for spike duplication. However, the reduction of neurons did not affect the runtime of the ADD and SUB modules. From a neuro-biological standpoint, \star SN P systems using extended rules is more natural as it does not require synchronized neurons. A disadvantage of \star SN P systems using extended rules, however, is that it is limited to one result per computation unlike SN P systems. This can be addressed by using standard rules instead and allowing synchronization of neurons.

SN P Systems using Extended Rules Two studies have proved computational completeness for SN P systems using extended rules, with different module designs. There is no nondeterministic ADD module shown in [16], but a SUB module with less neurons is shown compared to the work of [7]. A nondeterministic ADD module is shown in [7]. While the use of extended rules have reduced the number of neurons compared to SN P systems using standard rules, \star SN P systems using extended rules remain to have less neurons used for the ADD module while matching the number of neurons for the SUB module, all while maintaining the same runtime for both modules. However, this reduction of neurons was not due to stochasticity or asynchronicity but rather the leverage of using spike strengths which was also done in [16], hence the same number of neurons. Only \star SN P systems using standard rules can match the number of obtainable results by SN P systems using extended rules.

SN P Systems with Exhaustive Use of Rules The \star SN P register machine modules also have significantly less neurons in its register machine modules than that of SN P systems with exhaustive use of rules. This can be attributed to the low spike strength propagating around the system which is 2 at most. SN P systems with exhaustive use of rules also has a slightly slower SUB module runtime. In the design of the FIN module of SN P systems with exhaustive use of rules, [10] maintained the use of a time interval between two spikes to represent the result of its computation. It is worth noting that if the result of the computation was interpreted as the number of spikes sent to the environment,

this variant would then be able to use the register to output its spikes at one time step because of the exhaustive mode of sending spikes. This would have been the fastest means of sending results to the environment. Only \star SN P systems using standard rules can match the number of obtainable results by SN P systems with exhaustive use of rules as well.

SN P Systems with Generalized Use of Rules The modules of spiking neural P systems with generalized use of rules use more neurons than the register machine modules of \star SN P systems using extended rules which is also attributed to the low spike strength propagating around its system, despite both system leveraging the use of extended rules. SN P systems with generalized use of rules also has a slightly slower SUB module runtime similar to SN P systems with exhaustive use of rules. Like the comparison in SN P systems using extended rules and SN P systems with exhaustive use of rules, only \star SN P systems using standard rules can match the number of obtainable results by SN P systems with exhaustive use of rules as well.

Asynchronous SN P Systems With both the \star SN P systems using extended rules and asynchronous SN P system being asynchronous and using extended rules, both share the same advantages except stochasticity. However, differences in register machine module designs show that the asynchronous SN P modules use more neurons used and slower runtimes than standard. This is not caused by the features but rather suboptimal module design in general. \star SN P systems can also use standard rules, losing is asynchronous property but gaining the ability to produce multiple results which asynchronous SN P systems are unable to do.

SSN P Systems The two models in comparison uses different approaches when introducing stochasticity with respect to rule application, to which \star SN P systems uses an a priori approach while SSN P systems uses an a posteriori approach via stochastic delays. SSN P systems can be fully asynchronous by using random delays in all of its rules. However, it is not yet proven as of this writing whether it is computationally complete or not. This is also the case in asynchronous SN P systems if it instead used standard rules. The number of neurons in the register machine modules of \star SN P systems and \star SN P systems using extended rules is significantly less than SSN P systems. The runtime of SSN P system ADD and SUB modules also runs slower than standard. Like in the comparison with asynchronous SN P systems, \star SN P systems can also use standard rules, losing is asynchronous property but gaining the ability to produce multiple results which SSN P systems are unable to do.

Extended Spiking Neural P System Both models use an a priori approach in introducing stochasticity with respect to rule application. In [20], the model that the ESNPS is an instance of is not shown. Furthermore, no proof for computational completeness is provided. Thus, no comparative analysis with the of

register machine modules of the \star SN P system can be provided. In terms of other properties, the \star SN P system can be both synchronous and asynchronous depending on the use of extended rules or not which the ESNPS by design is unable to do.

With the \star SN P system being a formal definition of a stochastic SN P model, with the stochasticity in particular being introduced a priori to rule application, the \star SN P system can construct an instance of an ESNPS using extended rules only for the sake of removing the coherence condition. Let the ESNPS instance using \star SN P systems be

$$H = (O, \sigma_1, \dots, \sigma_m, \sigma_{m+1}, \sigma_{m+2}, \text{syn}, i_0)$$

where:

1. $O = \{a\}$;
2. $\sigma_1, \dots, \sigma_m, \sigma_{m+1}, \sigma_{m+2}$ are of the form:

$$\sigma_i = (n_i, R_i), 1 \leq i \leq (m + 2),$$

where:

- (a) $n_i = 1$;
- (b) if $i \leq m$, then $R_i = \{(p_i^1)a \rightarrow a, (p_i^2)a \rightarrow \lambda\}$ where $p_i^1 + p_i^2 = 1$, else $R_i = \{a \rightarrow a\}$
3. $\text{syn} = \{(m + 2, 1), \dots, (m + 2, m), (m + 1, m + 2), (m + 2, m + 1)\}$;
4. $i_0 = \{1, 2, \dots, m\}$.

In ESNPS, neurons σ_{m+1} and σ_{m+2} are non-stochastic. This is addressed in the \star SN P systems instance by setting a static probability of 1 to the rules in σ_{m+1} and σ_{m+2} . Other rule application probabilities remain dynamic as the values of it are set by the guider algorithm.

4.3 Introducing Heterogeneous Derivation Modes

From the previous section, it is shown that a \star SN P system using extended rule can be computationally complete with the rule application probability for all rule having the value between 0 and 1. However, in the register machine modules had an increase in the number of forgetting rules. Using exhaustive use of rules can reduce the number of forgetting rules for the label and auxiliary neurons such that a single forgetting rule used exhaustively $E'/a \rightarrow \lambda$ can be defined in those neurons where E' is the union of E of all forgetting rules in the neuron. This approach however, is not applicable for register neurons especially in the asynchronous context. Since the decrement of registers in the SUB module only requires partial removal of spikes in the register, the exhaustive use of rules cannot be employed that case. While there are other approaches to decrementing a register in exhaustive mode such as shown in [10], it requires synchronous behavior of neurons which is not applicable in \star SN P systems using extended rules. This then requires the introduction of a semantic to allow register neurons

to not apply its rules exhaustively while other neurons, or rules specifically can be used exhaustively.

A \star SN P system using extended rules with heterogeneous derivation mode of rules Π has neurons notated as $\sigma_i!$ with ! indicating if the rules within the neuron are used exhaustively and omitted if otherwise; and rules of the form $(p)E/a^r \rightarrow a^s!;d$ where ! indicates if the rule is used exhaustively and omitted if used in the standard way.

$N_c(\Pi)$ denotes the set of all natural numbers computed by Π , with subscript c denoting the way that the result is computed is by $5n$ spikes in the output neuron where n would be the output value.

Then $Spik_c P_{m,\mu}(prob_\rho, rule_{k,\nu}^x, cons_p, forg_q)$ denotes the family of all sets $N_c(\Pi)$ computed with at most $m \geq 1$ neurons where μ neurons are in exhaustive derivation mode, using rule application probabilities within the interval ρ and using at most $k \geq 1$ rules in each neuron where ν rules are in exhaustive derivation mode, with all firing rules $E/a^r \rightarrow a; d$ having $r \leq p$, and all forgetting rules $a^s \rightarrow \lambda$ having $s \leq q$. When one of the parameters m, k, ρ, p, q is not bounded, then it is replaced with $*$. A superscript x after *rule* in the notation indicates the use of extended rules.

Computational Universality

Corollary 1. $Spik_c P_*(prob_{(0,1)}, rule_k^x, cons_p, forg_q) = NRE, \forall k \geq 3, p \geq 8, q \geq 4$

Corollary 2. $Spik_c P_*(prob_{(0,1)}, rule_k^x, cons_p, forg_q) = NRE, \forall k \geq 3, p \geq 8, q \geq 4$

From the proof of computational completeness of \star SN P systems using extended rules, the way of obtaining the output is modified to the count of spikes by multiples of 5 in the output neuron instead. With this, there is no need to set the derivation mode for the output neuron and furthermore, no need for a FIN module similar to how [5] proved completeness for the stochastic spiking neural P systems.

Then every neuron with multiple forgetting rules can be merged into one forgetting rule that is in exhaustive mode as illustrated in Figure 7 for the ADD module and Figure 8 for the SUB module.

From rule-level heterogeneous derivation mode, every neuron with a rule used exhaustively can be generalized to a neuron that has rules that are all in exhaustive mode. The change of derivation mode of the firing rules for those neurons does not affect the simulation of the register machine modules since all of those firing rules already consume all spikes in the neuron when applied as illustrated in Figure 9 for the ADD module and Figure 10 of the SUB module.

With the introduction of heterogeneous derivation mode, the number of spikes forgotten in the system, q , is reduced from 10 to 4. However, the number of spikes consumed, p , remains at 8 due to the register neurons having standard derivation modes. Reduction of this parameter requires better leveraging of spike strengths.

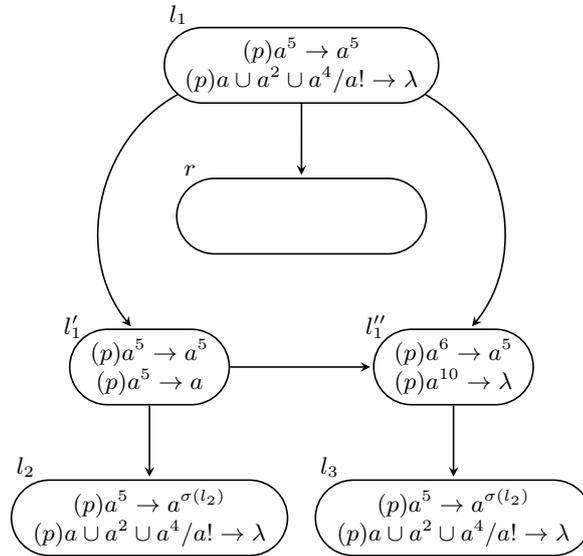


Fig. 7. \star SN P ADD (simulating $l_1 : (\text{ADD}(r), l_2, l_3)$)

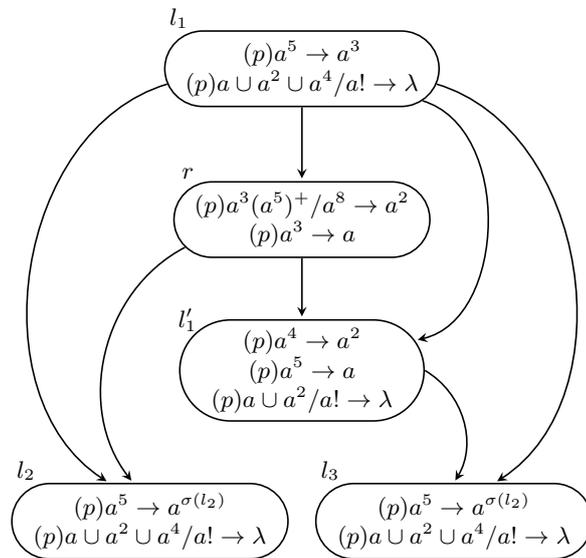


Fig. 8. \star SN P SUB (simulating $l_1 : (\text{SUB}(r), l_2, l_3)$)

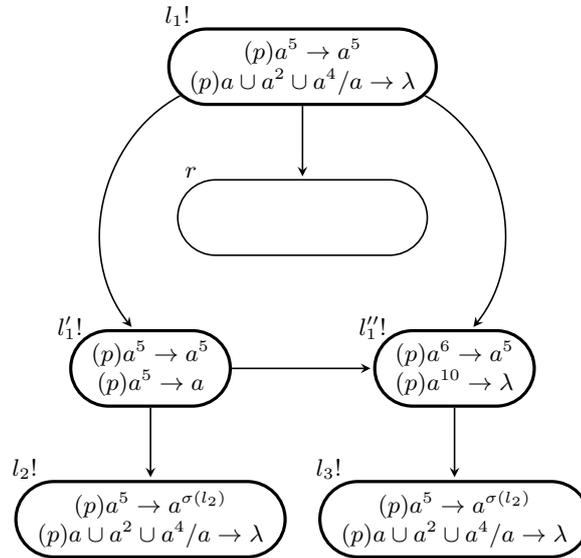


Fig. 9. *SN P ADD (simulating $l_1 : (\text{ADD}(r), l_2, l_3)$)

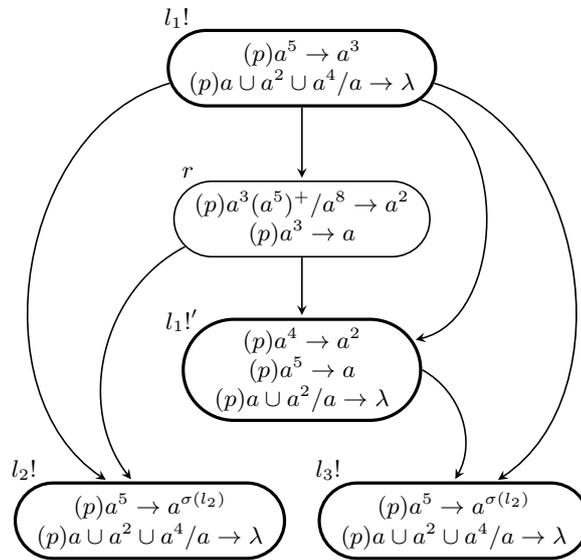


Fig. 10. *SN P SUB (simulating $l_1 : (\text{SUB}(r), l_2, l_3)$)

4.4 Into Normal Forms of \star SN P systems

In this study, the neurons used in the register machine modules along with the number of rules in some neurons were reduced. From the original objective of investigating the restriction of rule application probabilities, this study also explored normal forms [8] in line with the restriction of \star SN P systems. This study adhered to the restriction of removing delays as [8] showed that it is not needed to prove universality. However, the results of this study still used various regular expressions and forgetting rules when leveraging low-pass and high-pass neurons along with spike strengths despite being shown in [13] that computational completeness is possible with only using one type of regular expression $(a(aa)^*)$, and no use of forgetting rules. On the other hand, the results in [13] still use a synchronized rule application across its neurons and an odd-even number of spikes scheme for firing, which is problematic in \star SN P systems especially in the register neuron r used in the SUB module of \star SN P systems. Since all neurons are essentially asynchronous (given the restriction in probabilities), techniques leveraging topology with synchronous behavior is no longer useful. The question then focuses on how the register neuron communicate the fact where a decrement is successful or otherwise (when the value of the register is empty), provided that neurons are asynchronous. Furthermore, register interference is also another issue that needs to be addressed on top of the previously discussed predicament.

5 Final Remarks

In this study, the rule application probability of spiking neural P systems with stochastic application of rules was investigated. Conditions were identified that forces the rule application probabilities to be 1, thus showing that it is nontrivial for \star SN P systems to have all rule application probabilities to be < 1 . A solution was presented by extending the \star SN P design and introducing extended rules from [16]. \star SN P systems using extended rules are proved to be computationally universal while having less neurons and rule application probabilities of < 1 compared to using standard rules.

Further optimizations on the resources used was investigated which led to the introduction of heterogeneous derivation modes. The exhaustive derivation mode allowed the merging of forgetting rules in some neurons, which was then further generalized to exhaustive use of rules for its respective neurons.

It is important to remark, however, that resolving asynchronous behavior with heterogeneous derivation mode is formidable when showing computational universality. The non-standard mode of derivation compromises the register neurons in terms of decrementing its value and the known way of resolving this requires synchronous behavior. Future work then involves investigating the decrement behavior of asynchronous register neurons in exhaustive mode. Furthermore, generalized use of rules is not included as a possible derivation mode as it adds another layer of complexity that will deal with asynchronous behavior and exhaustive derivation mode.

This study also recommends further investigation in normal forms of \star SN P systems which includes idea of removing the forgetting rules, while using the odd-even firing scheme along with using a single type of regular expression. Normal forms then imply that low-pass and high-pass neurons are then no longer needed.

Acknowledgements

R.T.A. De La Cruz and I.C.H. Macababayao acknowledge support from ERDT scholarships of the DOST-SEI, Philippines. F.G.C. Cabarle acknowledges support from ERDT of DOST-SEI, and the Dean Ruben A. Garcia PCA from the University of the Philippines Diliman. H.N. Adorna is supported by the Semirara Mining Corporation PCA, also from UP Diliman.

References

1. Cabarle, F.G.C., Adorna, H.N., Jiang, M., Zeng, X.: Spiking neural P systems with scheduled synapses. *IEEE Transactions on Nanobioscience* **16**(8), 792–801 (2017)
2. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J., Song, T.: Spiking neural p systems with structural plasticity. *Neural Computing and Applications* **26**(8), 1905–1917 (2015)
3. Cavaliere, M., Egecioglu, O., Ibarra, O.H., Ionescu, M., Păun, G., Woodworth, S.: Asynchronous spiking neural p systems: Decidability and undecidability. In: *International Workshop on DNA-Based Computers*, pp. 246–255. Springer (2007)
4. Cavaliere, M., Ibarra, O.H., Păun, G., Egecioglu, O., Ionescu, M., Woodworth, S.: Asynchronous spiking neural P systems. *Theoretical Computer Science* **410**(24–25), 2352–2364 (2009)
5. Cavaliere, M., Mura, I.: Experiments on the reliability of stochastic spiking neural P systems. *Natural Computing* **7**(4), 453–470 (2008)
6. Chen, H., Freund, R., Ionescu, M., Păun, G., Pérez-Jiménez, M.J.: On string languages generated by spiking neural P systems. *Fundamenta Informaticae* **75**(1–4), 141–162 (2007)
7. Chen, H., Ionescu, M., Ishdorj, T.O., Păun, A., Păun, G., Pérez-Jiménez, M.J.: Spiking neural p systems with extended rules: universality and languages. *Natural Computing* **7**(2), 147–166 (2008)
8. Ibarra, O.H., Păun, A., Păun, G., Rodríguez-Patón, A., Sosik, P., Woodworth, S.: Normal forms for spiking neural P systems. *Theoretical Computer Science* **372**(2–3), 196–217 (2007)
9. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems. *Fundamenta informaticae* **71**(2, 3), 279–308 (2006)
10. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural p systems with an exhaustive use of rules. *International Journal of Unconventional Computing* **3**(2) (2007)
11. Jiang, Y., Su, Y., Luo, F.: An improved universal spiking neural p system with generalized use of rules. *Journal of Membrane Computing* **1**(4), 270–278 (2019)
12. Lazo, P.P., Cabarle, F.G., Adorna, H., Yap, J.M.: A return to stochasticity and probability in spiking neural p systems. In: *Pre-proc. International Conference on Membrane Computing (ICMC2020)*. Vienna, Austria and Ulaanbaatar, Mongolia (2020)

13. Macababayao, I., Cabarle, F.G.C. de la Cruz, R., Adorna, H., Xiangxiang, Z.: Notes on improved normal forms of spiking neural P systems and variants. In: Pre-proc. Asian Conference on Membrane Computing 2019 (ACMC2019) (2019)
14. Minsky, M.L.: Computation. Prentice-Hall Englewood Cliffs (1967)
15. Ochirbat, O., Ishdorj, T.O., Cichon, G.: An error-tolerant serial binary full-adder via a spiking neural p system using hp/lp basic neurons. *Journal of Membrane Computing* **2**(1), 42–48 (2020)
16. Păun, A., Păun, G.: Small universal spiking neural p systems. *BioSystems* **90**(1), 48–60 (2007)
17. Verlan, S., Freund, R., Alhazov, A., Ivanov, S., Pan, L.: A formal framework for spiking neural p systems. *Journal of Membrane Computing* **2**(4), 355–368 (2020)
18. Xu, Z., Cavaliere, M., An, P., Vrudhula, S., Cao, Y.: The stochastic loss of spikes in spiking neural P systems: Design and implementation of reliable arithmetic circuits. *Fundamenta Informaticae* **134**(1-2), 183–200 (2014)
19. Zeng, X., Zhang, X., Pan, L.: Homogeneous spiking neural P systems. *Fundamenta Informaticae* **97**(1-2), 275–294 (2009)
20. Zhang, G., Rong, H., Neri, F., Pérez-Jiménez, M.J.: An optimization spiking neural P system for approximately solving combinatorial optimization problems. *International Journal of Neural Systems* **24**(05), 1440006 (2014)

Appendix

The following subsections are from the work of [12].

Formal Definition of Spiking Neural P Systems with Stochastic Application of Rules

An SN P systems with **stochastic application of rules**, in short \star SN P systems, of degree $m \geq 1$ is of the form:

$$\Pi = (O, \sigma_1, \dots, \sigma_m, \text{syn}, i_0)$$

where:

1. $O = a$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \dots, \sigma_m$ are *neurons*, of the form:

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

where:

- (a) $n_i \geq 0$ is the *initial number of spikes* in the neuron;
- (b) R_i is a finite set of *rules* of the following two forms:
 - (1) $(\mathbf{p})E/a^r \rightarrow a; d$, where E is a regular expression over O , $r \geq 1$ and $d \geq 0$;
 - (2) $(\mathbf{p})a^s \rightarrow \lambda$ for some $s \geq 1$ with the restriction that $a^s \notin L(E)$ for any rule $E/a^r \rightarrow a; d$ of type (1) from R_i ;
with $\mathbf{p} \in [0, 1]$ as the *rule application probability* with the restriction that
 $\sum \mathbf{p}_{(1)} \leq 1$, for all $\mathbf{p}_{(1)}$ that are rule application probabilities of form (1) rules;
3. $\text{syn} \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $(i, i) \notin \text{syn}$ for $1 \leq i \leq m$ (*synapses* among neurons);
4. $i_0 \in \{1, 2, \dots, m\}$ indicates the *output neuron*.

This design extends forms (1) and (2) of R_i of the original SN P model, which allows the \star SN P designer to explicitly assign the probability of a rule being applied. No removal of syntax or semantics of the original SN P model was made in the design of this variant This design allows extension to other stochastic SN P systems such as SSN P systems where rule execution delays can be stochastic. Other extensions are also possible since the design is not restrictive. p can be assigned arbitrarily, through preprocessing similar to the multi-compartmental Gillespie algorithm, or through dynamic assignment using functions similar to the OSNPS guider or the stochastic transition system's scheduler. The probabilities can be fixed or dynamic during the computation as well, as long as the restriction for the firing rules is observed.

Stochastic Application of Rules

Since the rules of a neuron are modeled from the possible chemical reactions of the molecules that can occur within the membrane of a biological neuron, then the rule application probability represents the reactivity of those reactions. Furthermore, having more firing rules within a neuron means that there are more spike producing chemical reactions possible from the mixture of molecules within the limited confines of the neural membrane. Thus, having more possible reactions that can occur means less reactivity from all possible reactions. This is modeled by the restriction defined with the rule application probabilities for firing rules. Forgetting rules are not part of this restriction since the base definition of spiking neural P systems restrict the simultaneous application of both firing and forgetting rules [9].

The usage of the rule application probabilities depend on the cases enumerated in the previous subsection. In the first case, possible outcomes only include a rule being applied based on its rule application probability, p , or no rule being applied, \emptyset . An applicable rule with $p = 1$ means that \emptyset will not occur since $Pr(\emptyset) = 1 - p$. Since at a computational step where outcome \emptyset occurs means that the neuron did not apply the rule, then it did not perform any activity during that step, thus being idle.

The second case includes more outcomes than the first case. The possible outcomes now include the application of a firing rule from a set of > 1 applicable firing rules, or no firing rule is not applied at all. The definition of the restriction of the maximum total probability of rule application probabilities for all firing rules ($\sum p_{(1)} \leq 1$) satisfies the probability for the second case event space such that $Pr(F) = 1$. If $\sum p_{(1)} < 1$, then there is a probability for \emptyset to occur similar to the first case. Furthermore, the idle behavior is also the same when \emptyset occurs in this case.

Stochastic Behaviour

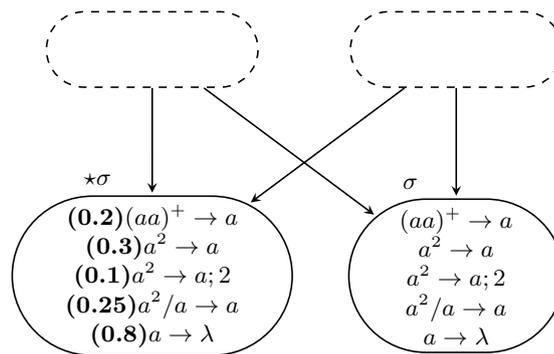


Fig. 11. *SN P vs SN P neurons

Figure 11 shows a \star SN P neuron $\star\sigma$ along with a regular SN P neuron σ to illustrate the effects in a neuron's behaviour when introduced with a stochastic process in its firing rules. Both neurons have the same set of rules but $\star\sigma$ adds a rule application probability to its firing rules.

When both neurons have one spike, $\star\sigma$ has a probability to not apply its forgetting rule while σ will definitely apply its forgetting rule. On the other hand, when both neurons have at least two spikes, both neurons similarly select firing rules randomly. However, the nature of the randomness on both neurons are different. In σ , the nondeterministic selection of rules implicitly means that all applicable firing rules have an equal probability (0.25) of selection. On the other hand, $\star\sigma$ explicitly indicates different probabilities of application for its firing rules. This means that when a significant number of firing rule selections are made from these two neurons, the results of the trials will show different distributions of selected firing rules despite both neurons having the same set of firing rules. Furthermore, $\sum p_{\star\sigma} = 0.85$ which means the probability of no firing rule being selected is $Pr(\emptyset) = 0.15$. This means that at any computational step where the firing rules of $\star\sigma$ are applicable, it has the probability of not applying a firing rule, which is not possible with σ . Furthermore, this stochastic idle state of a neuron means it can still receive spikes in that state.

Minimal cell-like membrane systems solving PSPACE-complete problems

David Orellana-Martín¹[0000-0002-2892-6775],
Luis Valencia-Cabrera^{1,2}[0000-0002-6576-9529], and
Mario J. Pérez-Jiménez^{1,2}[0000-0002-5055-0102]

¹ Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Universidad de Sevilla, Seville, 41012, Spain

² SCORE lab, I3US, Universidad de Sevilla, Seville, 41012, Spain
{dorellana, lvalencia, marper}@us.es

Abstract. In Membrane Computing, different variants of devices can be found by changing both syntactical and semantic ingredients. These devices are usually called membrane systems or P systems, and they recall the structure and behavior of living cells in the nature.

In this sense, rules are introduced as a way for objects to interact with membranes, giving P systems the ability to solve computational problems. Some of these rules, as division, separation and creation rules are inspired by the membrane division through the mitosis process or new membranes are created through gemmation. These rules seem to be crucial in the path to solve computationally hard problems. In this work, creation rules are used in classical P systems with symport/antiport rules, where objects travel through membranes without changing in order to achieve enough computational power to efficiently solve **PSPACE**-complete problems. More precisely, a solution to the **QSAT** problem is given by means of a uniform family of these systems.

Keywords: Membrane computing · Membrane creation · **QSAT** · Computational complexity theory.

1 Introduction

In 1998, Membrane Computing was introduced as a model of computation inspired by the structure and behaviour of living cells [17], abstracting the inner organelles as internal regions and chemical elements as objects of the system. The devices generated in this framework are called membrane systems or P systems. The type of region depends on the exact living model selected: On the one hand, a single cell with its inner compartments is abstracted as a cell-like membrane system (or P system) [17], where these membranes are semi-permeable walls that has control over the objects that can pass through it. The structure of these systems can be represented as a graph arranged as a rooted tree, where the root node is the outermost membrane, called *skin* membrane. On the other hand, several cells communicating among them can be found in nature forming a

tissue. We can find two main abstractions of this phenomenon: tissue-like membrane systems, or tissue P systems [6, 19], where cells can interchange objects with other cells or even with the environment, and spiking neural P systems [5], where the flow of communication through electrical impulses between neurons is abstracted through a directed graph.

Cell-like membrane systems use a membrane structure that can be either static or dynamic. In the first case, objects can move through the different membranes in each step of computation. P systems with symport/antiport rules [16] make an abstraction of the symport/antiport mechanism of the organelles of a cell that allow only some objects to pass, or can interchange chemical elements from two adjoining compartments. In transition P systems and P systems with active membranes [17, 18], the membrane structure could be changed, by dissolving the membranes or, in the latter, by duplicating a membrane and its contents by using division rules. Another method for duplicating a membrane is by using membrane separation rules [12], where the contents of the original membrane are distributed among the two new membranes. Another method to increase the number of membranes of the system is by means of membrane creation rules [7], where a single objects creates a new membrane. It could be seen as the chemical element taking some of the substrate of the membrane where it is and using it to create a new membrane. This kind of structure-changing mechanisms make P systems capable of efficiently solving computationally hard problems. More precisely, **NP**-complete or even **PSPACE**-complete problems have been solved by means of families of P systems that use one of these kinds of rules [23, 1, 2, 24].

In [22], a new variant of tissue P systems was introduced, where symport/antiport rules allow objects to evolve in the process of the communication from a region to other region. Some relevant results have been obtained in [13, 9, 10], where some frontiers of efficiency have been obtained. In the framework of cell-like membrane systems, creation rules have been used lately besides evolutionary communication rules in [21] to solve the SAT problem by means of a family of recognizer P systems with evolutionary communication rules and creation rules. In [11], an efficient solution to QSAT is given by means of a family of recognizer polarizationless P systems with active membranes and membrane creation, and in [8], an efficient solution to QSAT is given by means of a family of P systems with evolutionary symport/antiport rules of length $(1, 1)$ and creation rules. Creation rules have not been used in P systems with symport/antiport rules. In this work, we try to replicate the results of the latter paper by using recognizer P systems with symport/antiport rules with a minimal amount of objects per communication rule.

The rest of the work is organized as follows: Section 2 is devoted to introduce some formal language and set theory concepts used later through the paper. In Section 3, the definition of recognizer P systems with symport/antiport rules and membrane creation is given. In the following section, a polynomial-time and uniform solution to QSAT is given by means of a family of recognizer P systems with symport/antiport rules of length at most 1 and membrane creation, and

an overview of the computations and the formal verification of the design are specified. Finally, some remarks and open research lines are indicated.

2 Preliminaries

Some basic notions of formal languages, set theory and other terms used throughout the paper are recalled in this section. For a deeper explanation on formal languages and membrane computing, we refer the reader to [14, 20]

An alphabet Γ is a non-empty set, and its elements are called *symbols*. A string u over Γ is a finite sequence of symbols from Γ . The number of appearances of a symbol a in u is denoted by $|u|_a$. The *length* of u , denoted by $|u|$ is $\sum_{a \in u} |u|_a$.

A *multiset* over Γ is a pair (Γ, f) where $f : \Gamma \rightarrow \mathbb{N}$ is a mapping from Γ to the set of natural numbers \mathbb{N} . Let $m_1 = (\Gamma_1, f_1), m_2 = (\Gamma_2, f_2)$ two multisets over Γ . The union of m_1 and m_2 , denoted by $m_1 + m_2$ or $m_1 \cup m_2$ is defined as $(f_1 + f_2)(x) = f_1(x) + f_2(x)$. The relative complement of m_2 in m_1 , denoted by $m_1 \setminus m_2$, is defined as

$$(m_1 \setminus m_2)(x) = \begin{cases} f_1(x) - f_2(x) & \text{if } f_1(x) \geq f_2(x) \\ 0 & \text{if } f_1(x) < f_2(x) \end{cases}$$

The empty multiset is denoted by \emptyset , and the set of all finite multisets over Γ is denoted by $M_f(\Gamma)$.

The size of the set u is given by the total number of objects in u , and it is denoted by $|u|$.

The Cantor pairing function $\langle \cdot, \cdot \rangle$ is a bijective function defined as $\langle a, b \rangle = \frac{(a+b+1)(a+b)}{2} + b$.

3 Recognizer P systems with symport/antiport rules and creation rules

In this section, a definition of recognizer P systems with symport/antiport rules and creation rules is given, and both the syntax and semantics are recalled.

Definition 1. *A recognizer P system with symport/antiport rules and membrane creation of degree $q \geq 1$ is a tuple*

$$\Pi = (\Gamma, \Sigma, \mathcal{E}, H, \mu, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}_1, \dots, \mathcal{R}_q, i_{in}, i_{out})$$

where:

1. Γ is a finite alphabet of objects, and $\Sigma, \mathcal{E} \subseteq \Gamma, \Sigma \cap \mathcal{E} = \emptyset$;
2. H is a finite set of labels;
3. μ is a membrane structure whose elements are bijectively labelled by elements of H ;
4. $\mathcal{M}_i, 1 \leq i \leq q$ are finite multisets over $\Gamma \setminus (\Sigma \cup \mathcal{E})$;
5. \mathcal{R} is a set of rules of the following forms:

- *Symport rules:*
 - $(u, in) \in \mathcal{R}_i$, where $u \in M_f(\Gamma)$, except if i is the skin membrane, where $u \in M_f(\Gamma) \wedge u \notin M_f(\mathcal{E})$ (*send-in rules*);
 - $(u, out) \in \mathcal{R}_i$, where $u \in M_f(\Gamma)$ (*send-out rules*);
 - *Antiport rules:*
 - $(u, out; v, in) \in \mathcal{R}_i$, where $u, v \in M_f(\Gamma)$;
 - *Creation rules:* $[a \rightarrow [u]_i]_j$, where $i, j \in H, i \notin \{skin, i_{out}\}$, where *skin* is the label of the skin membrane, $a \in \Gamma, u \in M_f(\Gamma)$;
6. $i_{in} \in H$;
 7. $i_{out} = env$.

A configuration C_t of a P system with symport/antiport rules and creation rules is described by the membrane structure at the moment t and the multisets of objects over Γ of each membrane, and the multiset of objects over $\Gamma \setminus \mathcal{E}$ of the environment. We use the term region i to refer to a membrane if $i \in H$ and to the environment if $i = env$.

A symport rule $(u, in) \in \mathcal{R}_i$, called send-in rule, can be applied to a configuration C_t if there exists a membrane labelled by i , and the parent region contains a multiset of objects u . When applying such a rule, the multiset of objects u is consumed from the parent region and a multiset of objects u is produced in the membrane i in the next configuration.

A symport rule $(u, out) \in \mathcal{R}_i$, called send-out rule, can be applied to a configuration C_t if there exists a membrane labelled by i that contains a multiset of objects u . When applying such a rule, the multiset of objects u is consumed from the membrane i and a multiset of objects u is produced in the parent region.

An antiport rule $(u, out; v, in) \in \mathcal{R}_i$ can be applied to a configuration C_t if there exists a membrane labelled by i that contains a multiset of objects u , and whose parent region contains a multiset of objects v . When applying such a rule, the multisets of objects u and v are consumed from the membrane i and its parent region, respectively, and multisets v and u are produced in the membrane i and its parent region, respectively.

A creation rule $[a \rightarrow [u]_i]_j$ can be applied to a configuration C_t if there exists a membrane labelled by j that contains an object a . When applying such a rule, object a is consumed from membrane j and a new membrane labelled by i and containing the multiset of objects u appears as a child membrane of j .

A recognizer P system with symport/antiport rules and creation rules that does not send objects from the environment to the system is said to be a P system with symport/antiport rules and creation rules without environment. In this case, the set of objects of the environment \mathcal{E} is usually not defined in the tuple.

A *transition* of a P system Π is defined as a computational step of Π , passing from one configuration to the next one, and denoted by $C_t \Rightarrow_{\Pi} C_{t+1}$. A *computation* of a P system is a sequence of configurations such that a configuration C_{t+1} is always obtained from C_t by applying a computation step. C_0 is the initial configuration of Π .

In [3, 4], the semantics applied are *maximalist* in the following sense: In each membrane, an arbitrary number of creation rules can be applied, and they do not interfere with the application of other types of rules. In [11, 21], more restrictive semantics were introduced. When a creation rule is applied in a membrane h , no other rules can be applied in the same computational step. In this case, dealing with P systems with symport/antiport rules and membrane creation, either communication rules in a maximal parallel way or a single creation rule, can be applied in a membrane in a transition, but not both at the same time. As a recognizer membrane system, all the computation of a recognizer P system with communication rules and creation rules halt and either an object **yes** or an object **no** (but not both) is sent to the environment at the last step of the computation.

The length of a symport rule $r \equiv (u, in)$ or $r \equiv (u, out)$ is given by the number of objects in multiset u ; that is, it is equal to $|u|$. The length of an antiport rule $r \equiv (u, out; v, in)$ is given by the total number of objects in the rule; that is, it is equal to $|u| + |v|$. Let us denote the length of a rule r by $l(r)$.

The class of all recognizer P systems with symport/antiport rules and membrane creation of degree q is denoted by $\mathcal{CCC}(k)$, where k represents the maximal number of objects in a communication rule; that is, $k = \max(l(r) \mid r \in \mathcal{R}_i, 1 \leq i \leq q)$. The class of recognizer membrane systems of this type when environment plays a passive role; that is, when no objects can be set from the environment to the P system itself, is denoted by $\widehat{\mathcal{CCC}}(k)$.

All the concepts of a decision problem and the class of decision problems that can be solved by means of a uniform family of membrane systems from $\mathcal{CCC}(k)$ can be extracted from [21, 14, 15]. The class of problems that can be solved by means of a uniform family of recognizer P systems with symport/antiport rules of length at most k and membrane creation with environment (respectively, without environment) is denoted by $\mathbf{PMC}_{\mathcal{CCC}(k)}$ (resp., $\mathbf{PMC}_{\widehat{\mathcal{CCC}}(k)}$).

4 An efficient solution to QSAT in $\widehat{\mathcal{CCC}}(1)$

In this section, an efficient solution to the QSAT problem by means of a uniform family $\mathbf{\Pi}$ of P systems from $\widehat{\mathcal{CCC}}(1)$. Let $t = \langle n, p \rangle$. Each P system $\Pi(t)$, $t \in \mathbb{N}$, from $\mathbf{\Pi}$ solves all instances from QSAT with n variables and p clauses.

For each pair $n, p \in \mathbb{N}$, we consider a recognizer P system with symport/antiport rules of length 1 and creation rules

$$\Pi(\langle n, p \rangle) = (\Gamma, \Sigma, H, \mu, \mathcal{M}_{skin}, \mathcal{M}_{\langle 0, \# \rangle}, \mathcal{R}, i_{in}, i_{out})$$

that will solve all instances with n variables and p clauses, where

$$\begin{aligned} \text{cod}(\varphi) &= \{x_{i,j} \mid x_i \in C_j\} \cup \{\bar{x}_{i,j} \mid \neg x_i \in C_j\} \\ s(\varphi) &= \langle n, p \rangle \end{aligned}$$

1. The working alphabet is defined as follows:

$$\begin{aligned} \Gamma &= \Sigma \cup \{\text{yes}, \text{no}, d', d'_i, d'_f, d''\} \cup \\ &\quad \{\alpha_i \mid 0 \leq i \leq n^2 \cdot p + 5n + 2p + 3\} \cup \{\alpha'_i \mid 0 \leq i \leq n^2 \cdot p + 5n + 2m + 4\} \cup \end{aligned}$$

$$\begin{aligned} & \{c_{i,j,r} \mid 1 \leq i \leq n, 1 \leq j \leq p, r \in \{t, f\}\} \cup \\ & \{d_{i,r} \mid 0 \leq i \leq n, r \in \{t, f\}\} \cup \{z_i, z_{i,t}, z_{i,f} \mid 1 \leq i \leq n\} \cup \\ & \{x_{i',i,j,t}, \bar{x}_{i',i,j,f} \mid 1 \leq i, i' \leq n, 1 \leq j \leq p\}. \end{aligned}$$

2. The input alphabet $\Sigma = \{x_{i,j}, \bar{x}_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq p\}$.
3. $H = \{skin, 1, \dots, p, yes, no, \#\} \cup \{\langle i, r \rangle \mid 0 \leq i \leq n, r \in \{t, f\}\} \cup \{\langle i, Q, r, r' \rangle \mid 0 \leq i \leq n, Q \in \{\exists, \forall\}, r, r' \in \{t, f\}\} \cup \{\langle i, \#\rangle \mid 0 \leq i \leq n^2 \cdot p + 5n + 2m + 4\}$.
4. $\mu = [[\]_{\langle 0, \#\rangle}]_{skin}$.
5. $\mathcal{M}_{skin} = \{z_{1,t}, z_{1,f}\}, \mathcal{M}_{\langle 0, \#\rangle} = \{\alpha_0, \alpha'_0\}$.
6. The set of rules \mathcal{R} :

6.1 Rules for the counter of the elements of $\langle 0, \#\rangle$

$$\begin{aligned} & [\alpha_i \rightarrow [\alpha_{i+1}]_{\langle i+1, \#\rangle}]_{\langle i, \#\rangle} \text{ for } 0 \leq i \leq n^2 \cdot p + 5n + 2p + 2 \\ & [\alpha'_i \rightarrow [\alpha'_{i+1}]_{\langle i+1, \#\rangle}]_{\langle i, \#\rangle} \text{ for } 0 \leq i \leq n^2 \cdot p + 5n + 2p + 3 \\ & (\alpha_{n^2+5n+2p+3}, out) \in \mathcal{R}_{\langle i, \#\rangle} \text{ for } 1 \leq i \leq n^2 \cdot p + 5n + 2p + 3 \\ & (\alpha'_{n^2+5n+2p+4}, out) \in \mathcal{R}_{\langle i, \#\rangle} \text{ for } 1 \leq i \leq n^2 \cdot p + 5n + 2p + 4 \\ & (\alpha_{n^2+5n+2p+3}, out) \in \mathcal{R}_{\langle 0, \#\rangle} \\ & (\alpha'_{n^2+5n+2p+4}, out) \in \mathcal{R}_{\langle 0, \#\rangle} \end{aligned}$$

6.2 Rules to return a positive answer

$$\begin{aligned} & (\alpha_{n^2+5n+2p+3}, in) \in \mathcal{R}_{yes} \\ & [\alpha_{n^2+5n+2p+3} \rightarrow [\mathbf{yes}]_{\#}]_{yes} \\ & (\mathbf{yes}, out) \in \mathcal{R}_{\#} \\ & (\mathbf{yes}, out) \in \mathcal{R}_{yes} \\ & (\mathbf{yes}, out) \in \mathcal{R}_{skin} \end{aligned}$$

6.3 Rules to return a negative answer

$$\begin{aligned} & [\alpha'_{n^2+5n+2p+4} \rightarrow [\]_{no}]_{skin} \\ & (\alpha_{n^2+5n+2p+3}, in) \in \mathcal{R}_{no} \\ & [\alpha_{n^2+5n+2p+3} \rightarrow [\mathbf{no}]_{\#}]_{no} \\ & (\mathbf{no}, out) \in \mathcal{R}_{\#} \\ & (\mathbf{no}, out) \in \mathcal{R}_{no} \\ & (\mathbf{no}, out) \in \mathcal{R}_{skin} \end{aligned}$$

6.4 Rules to generate the membrane structure

$$\begin{aligned} & [z_{1,r} \rightarrow [z_1 d'']_{\langle 1, r \rangle}]_{skin} \text{ for } r \in \{t, f\} \\ & [z_{i,r} \rightarrow [z_i d']_{\langle i, r \rangle}]_{\langle i-1, r' \rangle} \text{ for } 2 \leq i \leq n, i \text{ odd}, r, r' \in \{t, f\} \\ & [z_{i,r} \rightarrow [z_i d'']_{\langle i, r \rangle}]_{\langle i-1, r' \rangle} \text{ for } 2 \leq i \leq n, i \text{ even}, r, r' \in \{t, f\} \\ & [z_i \rightarrow [z_{i+1,t} z_{i+1,f}]_{\#}]_{\langle i, r \rangle} \text{ for } 1 \leq i < n, r \in \{t, f\} \\ & (z_{i,r}, out) \in \mathcal{R}_{\#} \text{ for } 1 \leq i \leq n, r \in \{t, f\} \end{aligned}$$

6.5 Rules to check which clauses are satisfied

$$\begin{aligned}
 & [x_{i,j} \rightarrow [x_{1,i,j,t} x_{1,i,j,f}]_{\#}]_{skin} \text{ for } 1 \leq i \leq n, 1 \leq j \leq p \\
 & [\bar{x}_{i,j} \rightarrow [\bar{x}_{1,i,j,t} \bar{x}_{1,i,j,f}]_{\#}]_{skin} \text{ for } 1 \leq i \leq n, 1 \leq j \leq p \\
 & [x_{i',i,j,r} \rightarrow [x_{i'+1,i,j,t} x_{i'+1,i,j,f}]_{\#}]_{\langle i'+1,r \rangle} \\
 & \text{for } 1 \leq i' < i \leq n, 1 \leq j \leq p, r \in \{t, f\} \\
 & [\bar{x}_{i',i,j,r} \rightarrow [\bar{x}_{i'+1,i,j,t} \bar{x}_{i'+1,i,j,f}]_{\#}]_{\langle i'+1,r \rangle} \\
 & \text{for } 1 \leq i' < i \leq n, 1 \leq j \leq p, r \in \{t, f\} \\
 & (x_{i',i,j,r}, out) \in \mathcal{R}_{\#} \\
 & \text{for } 1 \leq i' < i \leq n, 1 \leq j \leq p, r \in \{t, f\} \\
 & (\bar{x}_{i',i,j,r}, out) \in \mathcal{R}_{\#} \\
 & \text{for } 1 \leq i' < i \leq n, 1 \leq j \leq p, r \in \{t, f\} \\
 & (x_{i'+1,i,j,r}, in) \in \mathcal{R}_{\langle i'+1,r \rangle} \\
 & \text{for } 1 \leq i < i' \leq n, 1 \leq j \leq p, r \in \{t, f\} \\
 & (\bar{x}_{i'+1,i,j,r}, in) \in \mathcal{R}_{\langle i'+1,r \rangle} \\
 & \text{for } 1 \leq i < i' \leq n, 1 \leq j \leq p, r \in \{t, f\} \\
 & [x_{i,i,j,t} \rightarrow [c_{i,j,t} c_{i,j,f}]_{\#}]_{\langle i,t \rangle} \text{ for } 1 \leq i < n, 1 \leq j \leq p \\
 & [\bar{x}_{i,i,j,f} \rightarrow [c_{i,j,t} c_{i,j,f}]_{\#}]_{\langle i,f \rangle} \text{ for } 1 \leq i < n, 1 \leq j \leq p \\
 & [x_{n,n,j,t} \rightarrow [c_{n,j,t}]_{\#}]_{\langle n,t \rangle} \text{ for } 1 \leq j \leq p \\
 & [\bar{x}_{n,n,j,f} \rightarrow [c_{n,j,f}]_{\#}]_{\langle n,f \rangle} \text{ for } 1 \leq j \leq p
 \end{aligned}$$

$$\begin{aligned}
 & (c_{i,j,r}, out) \in \mathcal{R}_{\#} \text{ for } 1 \leq i \leq n, 1 \leq j \leq p, r, r' \in \{t, f\} \\
 & (c_{i,j,r}, in) \in \mathcal{R}_{\langle i+1,r \rangle} \text{ for } 1 \leq i < n, 1 \leq j \leq p, r \in \{t, f\} \\
 & [c_{i,j,r} \rightarrow [c_{i+1,j,t} c_{i+1,j,f}]_{\#}]_{\langle i',r \rangle} \\
 & \text{for } 1 \leq i' \leq i < n, 1 \leq j \leq p, r \in \{t, f\}
 \end{aligned}$$

6.6 Rules to check if **all** clauses are satisfied

$$\begin{aligned}
 & [z_n \rightarrow [d_0]_{\#}]_{\langle n,r \rangle} \text{ for } r \in \{t, f\} \\
 & [c_{n,j,r} \rightarrow []_j]_{\langle n,r' \rangle} \text{ for } 1 \leq j \leq p, r, r' \in \{t, f\} \\
 & (d_j, in) \in \mathcal{R}_{j+1} \text{ for } 0 \leq j < p, r \in \{t, f\} \\
 & [c_{n,j,r} \rightarrow []_j]_{\langle n,r \rangle} \text{ for } 1 \leq j \leq p, r \in \{t, f\} \\
 & [d_j \rightarrow [d_{j+1}]_{\#}]_{i+1} \text{ for } 0 \leq j < p \\
 & (d_j, out) \in \mathcal{R}_{\#} \text{ for } 0 \leq j \leq p, r \in \{t, f\} \\
 & (d_j, out) \in \mathcal{R}_j \text{ for } 0 \leq j \leq p, r \in \{t, f\} \\
 & [d_p \rightarrow [d_{n,r}]_{\#}]_{\langle n,r \rangle} \text{ for } r \in \{t, f\} \\
 & (d_{n,r}, out) \in \mathcal{R}_{\#} \text{ for } r \in \{t, f\}
 \end{aligned}$$

6.7 Rules to check if quantifiers are satisfied

$$\begin{aligned}
 & (d_{i,r}, out) \in \mathcal{R}_{\langle i,r \rangle} \text{ for } 1 \leq i \leq n, r, r' \in \{t, f\} \\
 & [d_{i+1,r} \rightarrow []_{\langle i,\exists,r,r' \rangle}]_{\langle i,r' \rangle} \text{ for } 1 \leq i < n, i \text{ odd}, r, r' \in \{t, f\} \\
 & [d_{i+1,r} \rightarrow []_{\langle i,\forall,r,r' \rangle}]_{\langle i,r' \rangle} \text{ for } 2 \leq i < n, i \text{ even}, r, r' \in \{t, f\} \\
 & (d', in) \in \mathcal{R}_{\langle i,\exists,r,r' \rangle} \text{ for } 1 \leq i \leq n, i \text{ odd}, r, r' \in \{t, f\} \\
 & [d' \rightarrow [d_{i,r'}]_{\#}]_{\langle i,\exists,r,r' \rangle} \text{ for } 1 \leq i \leq n, i \text{ odd}, r, r' \in \{t, f\} \\
 & (d_{\langle i,r' \rangle}, out) \in \mathcal{R}_{\#} \text{ for } 1 \leq i \leq n, i \text{ odd}, r, r' \in \{t, f\} \\
 & (d_{\langle i,r' \rangle}, out) \in \mathcal{R}_{\langle i,\exists,r,r' \rangle} \text{ for } 1 \leq i \leq n, i \text{ odd}, r, r' \in \{t, f\}
 \end{aligned}$$

$$\begin{aligned}
 (d'', in) &\in \mathcal{R}_{\langle i, \forall, r, r' \rangle} \text{ for } 1 \leq i \leq n, i \text{ even}, r, r' \in \{t, f\} \\
 [d'' \rightarrow [d'_r]_{\#}]_{\langle i, \forall, r, r' \rangle} &\text{ for } 1 \leq i \leq n, i \text{ even}, r, r' \in \{t, f\} \\
 (d'_r, out) &\in \mathcal{R}_{\#} \text{ for } 1 \leq i \leq n, r, r' \in \{t, f\} \\
 (d'_r, out) &\in \mathcal{R}_{\langle i, \forall, r, r' \rangle} \text{ for } 1 \leq i \leq n-1, r, r' \in \{t, f\} \\
 (d'_{r''}, in) &\in \mathcal{R}_{\langle i, \forall, r, r' \rangle} \text{ for } 1 \leq i \leq n, i \text{ even}, r, r', r'' \in \{t, f\}, r \neq r'' \\
 [d'_{r''} \rightarrow [d_{i,r'}]_{\#}]_{\langle i, \forall, r, r' \rangle} &\text{ for } 1 \leq i \leq n, i \text{ even}, r, r', r'' \in \{t, f\}, r \neq r'' \\
 (d_{i,r'}, out) &\in \mathcal{R}_{\#} \text{ for } 1 \leq i \leq n, r, r' \in \{t, f\} \\
 (d_{i,r'}, out) &\in \mathcal{R}_{\langle i, Q, r, r' \rangle} \text{ for } 1 \leq i \leq n-1, r, r' \in \{t, f\}, Q \in \{\exists, \forall\} \\
 [d_{1,r} \rightarrow []_{yes}]_{skin} &\text{ for } r \in \{t, f\}
 \end{aligned}$$

7. $i_{in} = skin$.

8. $i_{out} = env$.

4.1 An overview of the computation

Let $\varphi^* = \exists x_1 \forall x_2 \dots Q_n x_n \varphi(x_1, \dots, x_n)$ an existential fully quantified formula associated with a Boolean formula $\varphi(x_1, \dots, x_n) \equiv C_1 \wedge \dots \wedge C_p$ in CNF, where each clause $C_j = l_{j,1} \vee \dots \vee l_{j,r_j}$, $Var(\varphi) = \{x_1, \dots, x_n\}$ and $l_{j,k} \in \{x_i, \neg x_i \mid 1 \leq i \leq n\}$. Let us suppose that the number of variables, n , and the number of clauses, p , is at least 2. We consider a polynomial encoding (cod, s) from QSAT in Π as follows: for each formula φ associated to an existential fully quantified formula φ^* with n variables and p clauses, $s(\varphi) = \langle n, p \rangle$ and $cod(\varphi) = \{x_{i,j} \mid x_i \in C_j\} \cup \{\bar{x}_{i,j} \mid \neg x_i \in C_j\}$.

The proposed solution follows a brute force scheme of recognizer P systems with symport/antiport rules and membrane creation without environment, and it consists of the following stages:

Generation and first checking stage By applying rules from 6.4, a membrane structure is generated. In some sense, it reminds a binary tree, but having some “garbage” membranes, labelled by $\#$, used to generate the objects $z_{i+1,t}$ and $z_{i+1,f}$. Besides, using rules from 6.5, objects from $cod(\varphi)$ will be passed throughout the membrane structure in such a way that in the level i , the i -th variable will be checked and, if the corresponding truth assignment makes true a literal in a clause j , then objects $c_{i,j,t}$ and $c_{i,j,f}$ will appear, that will be passed by the membranes up to a membrane labelled by $\langle n, r \rangle$. This stage takes $2n^2 \cdot 2p$ steps.

Second checking stage Rules from 6.6 are in charge of checking whether all the clauses are satisfied in a truth assignment. For that, if there exists an object $c_{n,j,r}$ in a membrane labelled by $\langle n, r \rangle$, it means that the corresponding truth assignment makes true the clause j . Therefore, a membrane labelled by j is created within such a membrane $\langle n, r \rangle$. Object d_0 will go through all membranes, creating a “garbage” membrane within them and passing to the next one, possibly arriving to membrane p . In that case, object d_p creates a new garbage membrane with an object $d_{n,r}$, that will be useful in the next stage. This stage takes $3p + 5$ steps.

Quantifier checking stage If an object $d_{i,r}$ ($r \in \{t, f\}$) appears in a membrane, then the quantifier in this level is checked. Depending on the parity of the level, either a universal or an existential quantifier should be checked. In the generation stage, objects d' and d'' were created for this purpose. On the one hand, when an existential quantifier is being checked, an object d' will exist in such a membrane, and will change into an object $d_{i-1,r}$ if and only if there is at least one object $d_{i,r}$ that has created a membrane $\langle i, \exists, r, r' \rangle$. On the other hand, when a universal quantifier is to be checked, an object d'' will be present in such a membrane, and will change into an object $d_{i-1,r}$ if and only if there are two objects $d_{i,r}$ that have created two membranes labelled by $\langle i, \forall, r, r' \rangle$. These objects will reach the skin membrane giving way to the last stage. This whole stage is computed by the application of rules from 6.6. This stage takes $8n - 6$ steps if n is even and $8n$ steps if n is odd.

Output stage Counters α and α' are used in this stage in order to know if an object $d_{1,r}$ has reached the skin membrane. In such a case, a membrane labelled by *yes* will be created, and when object $\alpha_{n^2+5n+2p+3}$ reaches the skin membrane, it will go into membrane *yes* and will change into an object **yes** that will be sent to the environment. In the case that object $d_{1,r}$ does not appear in the skin membrane, object $\alpha'_{n^2+5n+2p+4}$ will generate a membrane labelled by *no*, that will make the counter $\alpha_{n^2+5n+2p+3}$ change into an object **no**, and will be sent to the environment. Rules from 6.2 and 6.3 are the responsible in this stage. It takes $p + 8$ steps if n is even and $p + 9$ steps if n is odd.

4.2 Formal verification

Next, we prove that Π provides a polynomial time and uniform solution to QSAT.

Theorem 1. $\text{QSAT} \in \text{PMC}_{\widehat{\text{CC}}(1)}$

The family of P systems Π is polynomially uniform by Turing machines, polynomially bounded, sound and complete with regard to $(\text{QSAT}, \text{cod}, s)$ and both *cod* and *s* are polynomial-time computable functions.

Corollary 1. $\text{PSPACE} \subseteq \text{PMC}_{\widehat{\text{CC}}(1)}$

Proof. It suffices to know that QSAT is a PSPACE-complete problem, $\text{QSAT} \in \text{PMC}_{\widehat{\text{CC}}(1)}$ and the class $\text{PMC}_{\widehat{\text{CC}}(1)}$ is closed under polynomial-time reduction and under complementary.

5 Conclusions and future work

In this work, a result concerning evolutionary symport/antiport rules has been improved, in the sense that no evolution is needed in these kinds of rules while using creation rules. While in the previous work, a solution based on a family

of P systems from $\widehat{\mathcal{CC}\mathcal{E}\mathcal{C}}(1, 1)$ was detailed, in this work we restrict the number of objects used in a symport/antiport rule to one; that is, to P systems from $\widehat{\mathcal{CC}\mathcal{C}}(1)$. This is a demonstration of the power of creation rules, showing that rules with a minimal number of objects in symport/antiport rules is enough to reach presumed efficiency. The use of division rules and separation rules with this length of symport/antiport rules give P systems the power to efficiently solve only problems from **P**.

From the beginning, creation rules have been only used in cell P systems, because of the biological inspiration of the use of parts of a membrane to create a new membrane within it, but using creation rules in tissue P systems, where new cells would be created in the environment, and not in the cell itself, would be an interesting research line.

Acknowledgements:

This work was supported by “FEDER/Ministerio de Ciencia e Innovación Agencia Estatal de Investigación/ _Proyecto (TIN2017-89842-P)” - MABICAP. D. Orellana-Martín also acknowledges Contratación de Personal Investigador Doctor. (Convocatoria 2019) 43 Contratos Capital Humano Línea 2. Paidi 2020, supported by the European Social Fund and Junta de Andalucía.

References

1. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundam. Inf.* **58**(2), 6777 (Apr 2003)
2. Alhazov, A., Pan, L., Păun, Gh.: Trading polarizations for labels in P systems with active membranes. *Acta Inf.* **41**(23), 111144 (Dec 2004)
3. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A linear solution of subset sum problem by using membrane creation. In: Mira, J., Álvarez, J.R. (eds.) *Mechanisms, Symbols, and Models Underlying Cognition*. pp. 258–267. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
4. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A linear solution for qsat with membrane creation. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing*. pp. 241–252. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
5. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural P systems. *Fundam. Inf.* **71**(2,3), 279308 (Feb 2006)
6. Martín-Vide, C., Pun, G., Pazos, J., Rodríguez-Patn, A.: Tissue P systems. *Theoretical Computer Science* **296**(2), 295–326 (2003), machines, Computations and Universality
7. Mutyam, M., Krithivasan, K.: P systems with membrane creation: Universality and efficiency. In: *Proceedings of the Third International Conference on Machines, Computations, and Universality*. p. 276287. MCU '01, Springer-Verlag, Berlin, Heidelberg (2001)

8. Orellana-Martín, D., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: A solution to QSAT with evolutionary communication and membrane creation rules. *Theoretical Computer Science* **submitted** (2021)
9. Orellana-Martín, D., Valencia-Cabrera, L., Song, B., Pan, L., Pérez-Jiménez, M.J.: Narrowing frontiers with evolutionary communication rules and cell separation. In: *Proceedings of the 16th Brainstorming Week on Membrane Computing*. pp. 123–162. Sevilla, Spain (2018)
10. Orellana-Martín, D., Valencia-Cabrera, L., Song, B., Pan, L., Pérez-Jiménez, M.J.: Tuning frontiers of efficiency in tissue P systems with evolutionary communication rules. *Complexity* **2021** (Apr 2021)
11. Orellana-Martín, D., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Membrane creation in polarizationless P systems with active membranes. *Fundamenta Informaticae* **171**, 297–311 (2020)
12. Pan, L., Ishdorj, T.O.: P systems with active membranes and separation rules. *Journal of Universal Computer Science* **10**(5), 630–649 (may 2004)
13. Pan, L., Song, B., Valencia-Cabrera, L., Pérez-Jiménez, M.J., Hafstein, S.F.: The computational complexity of tissue P systems with evolutionary symport/antiport rules. *Complexity* **2018** (Jan 2018)
14. Paun, G., Rozenberg, G., Salomaa, A.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., USA (2010)
15. Pérez-Jiménez, M.J., Álvaro Romero-Jiménez, Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. *Natural Computing: An International Journal* **2**(3), 265–285 (Aug 2003)
16. Păun, A., Păun, Gh.: The power of communication: P systems with symport/antiport. *New Gen. Comput.* **20**(3), 295–305 (Jul 2002)
17. Păun, Gh.: *Computing with membranes*. Tech. rep., Turku Centre for Computer Science (1998)
18. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. *J. Autom. Lang. Comb.* **6**(1), 75–90 (Jan 2001)
19. Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Tissue P systems with cell division. *International Journal Of Computers Communications & Control* **3**(3), 295–303 (2008)
20. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages*. 3 vols. Springer-Verlag, Berlin, Heidelberg (1997)
21. Song, B., Li, K., Orellana-Martín, D., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: Cell-like P systems with evolutionary symport/antiport rules and membrane creation. *Information and Computation* **275**, 1045–1072 (2020)
22. Song, B., Zhang, C., Pan, L.: Tissue-like P systems with evolutionary symport/antiport rules. *Information Sciences* **378**, 177–193 (2017)
23. Sosik, P.: P systems attacking hard problems beyond NP: a survey. *Journal of Membrane Computing* **1** (09 2019)
24. Sosik, P., Rodríguez-Patn, A.: Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences* **73**(1), 137–152 (2007)

Feature Selection Algorithm Based on P Systems*

Hongping Song¹[0000-0001-6619-2787], Yourui Huang¹[0000-0002-9774-5791], Qi Song¹[0000--0001-7595-7964], Tao Han¹[0000--0001-6412-7455], and Shanyong Xu¹[0000--0001-7837-1117]

School of Electrical and Information Engineering, Anhui University of Science and Technology, Huainan 232001, China
shp7420@163.com

Abstract. Since the number of features of data sets is much higher than that of patterns, the curse of dimensionality has become an important problem. In order to overcome this problem, this paper proposes a feature selection algorithm based on P system (P-FS), which uses the parallel ability of cell-like P system and the advantage of evolutionary algorithms in search space to select features. Four datasets were used for the experiment, including three public datasets and one self-collected data set of edible oil spectrum. The experimental results show that the P-FS algorithm has good performance in classification accuracy, stability and convergence. Thus, P-FS algorithm is feasible in feature selection.

Keywords: Cell-like P systems · Feature selection · Genetic algorithm.

1 Introduction

Membrane computing is a new branch of natural computing. Inspired by the way cells deal with chemicals and the structure of cells, Pun[1], academician of the European Academy of Sciences, first proposed the membrane computing model in a research report of the computer center in Turku, Finland, in 1998. The first paper on membrane computing was published in 2000[2]. In recent years, the research of membrane computing has developed rapidly and become one of the frontier research fields in computational science. Membrane computing is a computing model abstracted from cells, tissues, organs and other biological structures, also known as P-system, which can be divided into three types: cellular, tissue and neural.

Cell-like P system is abstracted from the structure and function of the cell, which is mainly composed of membrane, object and evolution rule. Academician Paun introduced the formal definition and computability of cell-like p system in detail. He proved that cell-like P system [3] has the same computing power as Turing machine and has great parallelism, distribution and uncertainty.

* Supported by National Natural Science Foundation of China(Grant no. 61772033).

With the rapid development of machine learning and data processing technology, the problem of dimension disaster[4] is more serious. This kind of problem can be solved by dimension reduction. Feature extraction and feature selection[5] (FS) are two commonly used methods. Feature extraction maps feature space to smaller space. Feature selection reduces the number of features directly by selecting feature subset with enough information through evaluation criteria. Feature subset selection is an NP-Hard problem. The simplest method to evaluate each feature subset is the exhaustive method and finally determines the optimal feature subset. However, there are many shortcomings, such as time-consuming and high cost of the assessment. Using metaheuristic algorithm[6-7] can avoid increasing the computational complexity of feature selection. Recently, metaheuristic feature selection methods have developed rapidly. Genetic algorithm[8] (GA) is more easily applied to feature selection problems for using binary coding[9], so GAFS is widely used.

Inspired by membrane computing, this paper proposes a new feature selection method P-FS algorithm, which uses the advantages of GA algorithm in feature selection and the parallel characteristics of cell-like P system to find the optimal feature subset. P-FS algorithm and GAFS algorithm are tested on three common data sets and one spectral data set to verify the performance of P-FS algorithm in terms of classification accuracy, the number of selected features, stability and convergence.

2 Methodology

2.1 Cell-like P system

A cell-like P system with degree M [10] can be defined as:

$$\Pi = (V, O, H, \mu, \omega_1, \dots, \omega_m, R_1, \dots, R_m, i_0) \quad (1)$$

Where: V is a not empty finite alphabet, and the elements in the alphabet are called objects; $O \subseteq V$ is a collection of output objects; H is a set of membrane labels, $H = \{1, 2, \dots, m\}$; μ is a membrane structure with m membranes, and m is called the degree of Π ; $\omega_i \in V^*$ ($1 \leq i \leq m$), represents the multiple sets of objects in the region i of the membrane structure μ , and V^* is the set of strings composed of characters in V ; R_i ($1 \leq i \leq m$) is the set of evolution rules in each region i of membrane structure μ , which is usually written as $u \rightarrow v$. In this, v is the string in V^* , $v = v'$ or $v = v'\delta$. v' is a string on a collection $\{a_{here}, a_{out}, a_{inj} \mid a \in V, 1 \leq j \leq m\}$. δ is a special character and does not belong to V . When there is δ in the rule, the membrane is dissolved after the rule is executed; i_0 is the label of the output membrane of the membrane systems, $i_0 \in H$.

2.2 GA algorithm

GA is a metaheuristic algorithm, which belongs to the category of evolutionary algorithms (EA). It usually uses biological heuristic operators, such as mutation,

crossover and selection, to generate high-quality optimization and search solutions. GA for feature selection includes initialization of population, evaluation of individual fitness, selection, crossover, mutation and end condition judgment.

Population initialization: set the number of iterations and initialization to generate n chromosomes. A chromosome is represented by a binary string composed of 0 or 1. The length of the chromosome is the dimension of the dataset (the total number of features in the dataset). 0 means that the feature is not selected, and 1 means that the feature is selected.

Evaluation of individual fitness: fitness function is used to calculate the fitness value of the feature subset corresponding to each chromosome. In this paper, SVM algorithm was used to calculate the fitness value. In each iteration, the data set selected features according to the position of 1 in each chromosome, so as to generate new data sets with different features. The SVM model was used to classify the data set, and the classification accuracy of SVM was taken as the fitness value of each chromosome.

Selection: different fitness values of chromosomes lead to different probability of being selected. This paper uses roulette method to select chromosomes.

Crossover: two chromosomes exchange genes in a certain position according to a certain probability to produce new chromosomes.

Mutation: a gene in a chromosome changes from 0 to 1 or from 1 to 0 according to a certain probability to produce a new individual.

End condition judgment: when the current number of iterations is equal to the maximum number of iterations, the algorithm ends running.

3 Feature selection algorithm based on P system

GA has powerful search ability in feature selection, but it is computationally complex and takes a long time to process high-dimensional data. At the same time, the frequency of mutation is certain, and the effect of using mutation factor to jump out of local optimum has volatility. P system has the ability of parallel processing, so P-FS algorithm is proposed. Using the computing rules of cell-like P system and GA to select features has strong ability of global search, which is helpful to jump out of the global optimum.

3.1 Algorithm design

The designed P-FS algorithm adopts the membrane structure of cell-like P system, and its structure is shown in Figure 3-1. Its structural form is defined as:

$$\Pi = (V, O, H, \mu, \omega_1, \dots, \omega_4, R_1, \dots, R_4, i_0) \quad (2)$$

Where:

- (1) V is a non empty finite alphabet whose object is the feature subset corresponding to each chromosome in genetic algorithm;
- (2) $O \subseteq V$ is the output alphabet and the output algorithm results;

- (3) H is a set of membrane labels, $H = \{1, 2, 3, 4\}$;
- (4) μ is a membrane structure with m membranes, $m = 4$, as shown in Fig. 1;
- (5) $\omega_i \in V^*$ ($1 \leq i \leq m$), represents the multiple sets of objects in region i in membrane structure μ , corresponding to the initial binary chromosome populations in membrane 3 and membrane 4;
- (6) R_i ($1 \leq i \leq m$) is a set of evolution rules in each region of membrane structure, including computing fitness values for chromosomes, the selection, crossover and mutation in GA, and the communication rules for transferring objects in membrane to adjacent regions;
- (7) i_0 is the label of the output membrane of the membrane systems, $i_0 = 2$.

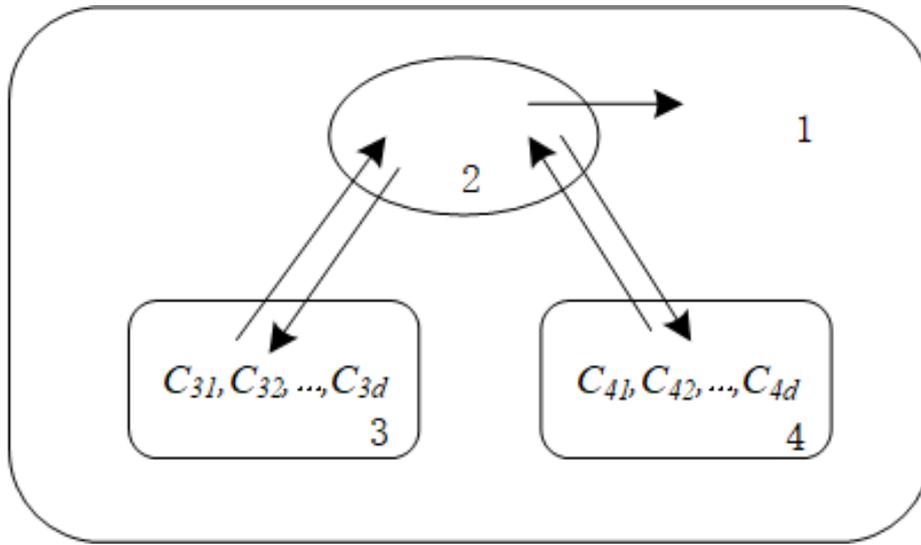


Fig. 1. Schematic diagram of membrane structure.

3.2 Evolutionary rules

There is no operation in membrane 1, but only the chromosome transferred from membrane 2 is recovered. As the main membrane, membrane 2 receives the chromosome population and corresponding fitness value transmitted by membrane 3 and membrane 4. Then, it transfers the population containing the best fitness value to membrane 3 and membrane 4, and the other population to membrane 1. Membrane 3 and membrane 4 mainly explore the search space globally to find the region where the optimal solution is located.

The chromosomes in membrane 3 were updated according to selection, crossover and mutation. At each iteration, the chromosomes are sorted according to the fitness value from large to small. After sorting, the population and the optimal

fitness value are transferred to membrane 2. The evolution rules of membrane 3 are as follows:

$$\begin{aligned}
 r_{31} &: C_{31}^t, C_{32}^t, \dots, C_{3k}^t \rightarrow C_{31}^{t'}, C_{32}^{t'}, \dots, C_{3k}^{t'} \\
 & \quad f_{31}^t, f_{32}^t, \dots, f_{3k}^t \rightarrow f_{31}^{t'}, f_{32}^{t'}, \dots, f_{3k}^{t'} \\
 r_{32} &: C_{31}^{t'} C_{32}^{t'} \dots C_{3k}^{t'} \rightarrow C_{31}^{t'} C_{32}^{t'} \dots C_{3k}^{t'} \left(C_{31}^{t'} C_{32}^{t'} \dots C_{3k}^{t'} \right)_{in2} \\
 & \quad f_{31}^{t'} f_{32}^{t'} \dots f_{3k}^{t'} \rightarrow f_{31}^{t'} f_{32}^{t'} \dots f_{3k}^{t'} \left(f_{31}^{t'} \right)_{in2}
 \end{aligned} \tag{3}$$

Where: k is the number of the population in membrane and the number of the population in membrane 3 and membrane 4 is equal; $C_{31}^{t'} C_{32}^{t'} \dots C_{3k}^{t'}$ is the sequence of chromosome selection, crossover, mutation and sorting in membrane 3 when the number of iterations is t ; $f_{31}^{t'}, f_{32}^{t'}, \dots, f_{3k}^{t'}$ is the fitness value of chromosomes in membrane 3 when the number of iterations is t .

The chromosomes in membrane 4 were updated according to selection, crossover and mutation. At each iteration, the chromosomes are sorted according to the fitness value from large to small. After sorting, the population and the optimal fitness value are transferred to membrane 2. The evolution rules of membrane 4 are as follows:

$$\begin{aligned}
 r_{41} &: C_{41}^t, C_{42}^t, \dots, C_{4k}^t \rightarrow C_{41}^{t'}, C_{42}^{t'}, \dots, C_{4k}^{t'} \\
 & \quad f_{41}^t, f_{42}^t, \dots, f_{4k}^t \rightarrow f_{41}^{t'}, f_{42}^{t'}, \dots, f_{4k}^{t'} \\
 r_{42} &: C_{41}^{t'} C_{42}^{t'} \dots C_{4k}^{t'} \rightarrow C_{41}^{t'} C_{42}^{t'} \dots C_{4k}^{t'} \left(C_{41}^{t'} C_{42}^{t'} \dots C_{4k}^{t'} \right)_{in2} \\
 & \quad f_{41}^{t'} f_{42}^{t'} \dots f_{4k}^{t'} \rightarrow f_{41}^{t'} f_{42}^{t'} \dots f_{4k}^{t'} \left(f_{41}^{t'} \right)_{in2}
 \end{aligned} \tag{4}$$

Where: $C_{41}^{t'} C_{42}^{t'} \dots C_{4k}^{t'}$ is the sequence of chromosome selection, crossover, mutation and sorting in membrane 4 when the number of iterations is t ; $f_{41}^{t'}, f_{42}^{t'}, \dots, f_{4k}^{t'}$ is the fitness value of chromosomes in membrane 4 when the number of iterations is t .

In membrane 2, the optimal fitness values transmitted from membrane 3 and membrane 4 were sorted from large to small, the populations corresponding to large fitness values were transmitted to membrane 3 and membrane 4, and the populations corresponding to small fitness values were transmitted to membrane 1. The evolution rules of membrane 2 are as follows:

$$\begin{aligned}
 r_{21} &: f_{31}^{t'} f_{41}^{t'} \rightarrow f_1^{t'} f_2^{t'} \\
 r_{22} &: f_1^{t'} f_2^{t'} \rightarrow f_1^{t'} f_2^{t'} \left(C_1^{t'}, C_2^{t'}, \dots, C_k^{t'} \right)_{in3,4} \\
 & \quad f_1^{t'} f_2^{t'} \rightarrow f_1^{t'} f_2^{t'} \left(C_1^{t'}, C_2^{t'}, \dots, C_{k'}^{t'} \right)_{in1}
 \end{aligned} \tag{5}$$

Where: $f_1^{t'} f_2^{t'}$ is the sequence of fitness values after sorting when the number of iterations is t . $C_1^{t'} C_2^{t'} \dots C_k^{t'}$ is the population corresponding to $f_1^{t'}$ when the number of iterations is t . $C_1^{t'}, C_2^{t'}, \dots, C_{k'}^{t'}$ is the population corresponding to $f_2^{t'}$ when the number of iterations is t .

The flow chart of the algorithm is shown in Fig. 2:

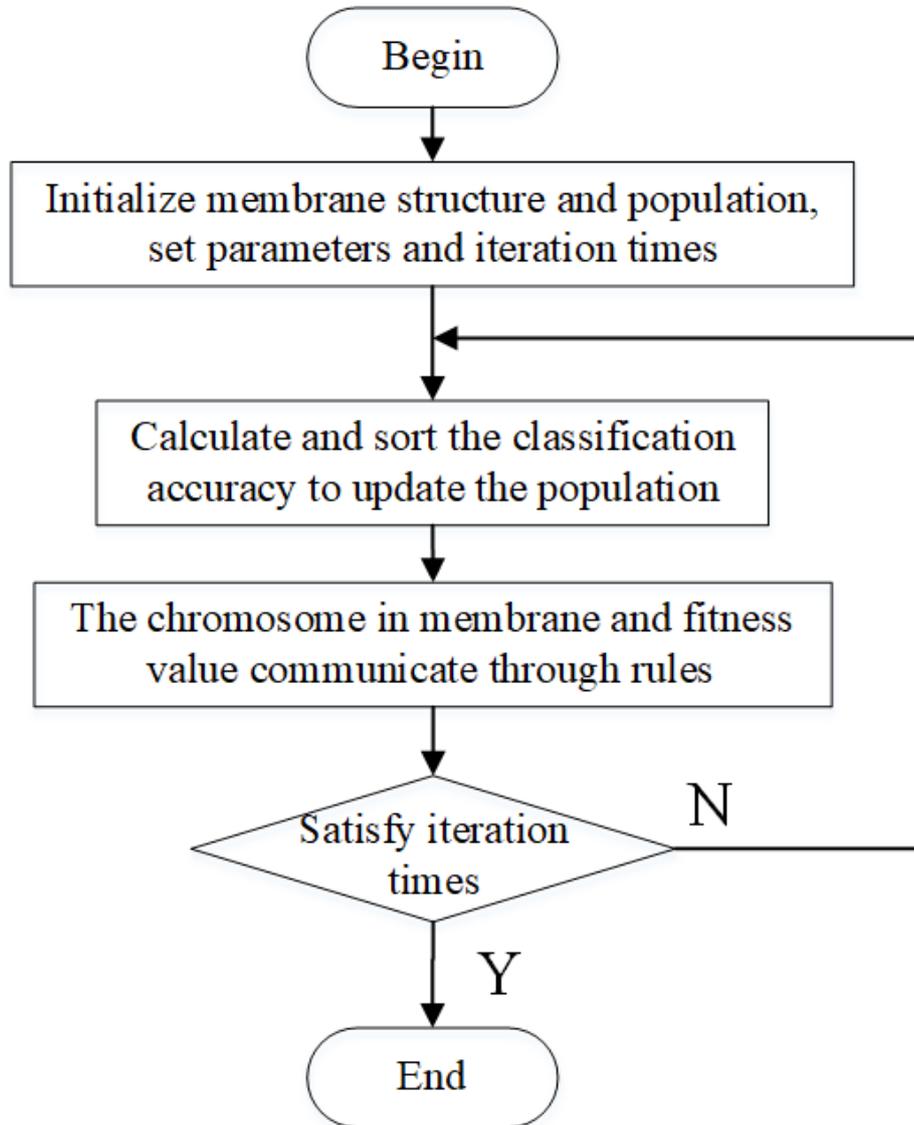


Fig. 2. Flow chart of P-FS algorithm.

4 Results and discussion

4.1 Dataset

We collected three classification data sets from different fields of UCI Machine Learning Repository, and used laser-induced fluorescence technology to collect the fluorescence spectrum data of edible oil. In order to calculate fitness, we divided each data set into the training set and the testing set. Table 1 shows the statistical data of the four datasets. From Table 1, we can see that the number of features and the number of categories contained in the four datasets are obviously different, which can verify the performance of P-FS algorithm in a different number of features.

Table 1. Statistics of data sets.

Dataset	Number of feature	Number of classes	Number of instances	
			Training	Validation
Gas sensor array drift (Gas)	128	6	333	112
Musk	166	2	357	119
Urban land cover data setUlc	147	9	506	169
Oil	2048	4	300	100

4.2 Classifier

We use the classification accuracy of Support Vector Machine[11] (SVM) as the fitness value of P-FS algorithm. The classification idea of SVM algorithm is simple and the classification effect is good. The parameters of SVM in our paper use the default value. The steps of using SVM model to calculate the fitness value of P-FS algorithm are as follows:

Step 1: In each iteration, produces n chromosomes, and n new data sets are formed by selecting characteristic subsets according to the position of 1 in each chromosome.

Step 2: N new data sets were taken as the input of SVM model, and each data set was randomly divided into training set and verification set at a ratio of 3:1.

Step 3: Use the training set to train the SVM model, and use the verification set to make prediction. Then, compare the predicted results with the actual ones, and obtain the classification accuracy of the verification set. Each chromosome corresponds to a new dataset, and the classification accuracy of each dataset can be obtained using the SVM model. Therefore, the classification accuracy of SVM model was taken as the fitness value of each chromosome.

4.3 Performance of the algorithm

The P-FS algorithm and GAFS algorithm are tested on three common data sets and one spectral data set. The parameter design is shown in Table 2. The chromosome number, iteration times and mutation probability are set to be the same, so as to compare the performance of the two algorithms.

Table 2. Parameter setting of the algorithm.

Parameters	Algorithm	
	P-FS	GAFS
Number of chromosomes	20	20
Iterations	200	200
Mutation rate	0.05	0.05

The fitness values and selected feature numbers of P-FS algorithm and GAFS algorithm on three common data sets and one spectral data set are shown in Table 3 and Table 4. The convergence curves of P-FS algorithm and GAFS algorithm on three data sets are shown in Fig. 3. From Table 3, we can see that the accuracy of P-FS algorithm in four datasets is better than that of GAFS algorithm, while from Table 4, we can see that the number of features selected by P-FS algorithm is less than that of GAFS algorithm. From Fig. 3, we can see that the convergence of P-FS algorithm is higher than that of GAFS algorithm. Through the experimental data, we can see that P-FS algorithm has great advantages over GAFS algorithm in accuracy, search efficiency, stability and convergence.

Table 3. Comparison of fitness values of algorithms on three datasets.

Datasets	Accuracy of P-FS	Accuracy of GAFS
Gas	0.9286	0.9286
Musk	0.8487	0.8067
Ulc	0.8521	0.8462
Oil	0.97	0.92

Table 4. Comparison of characteristic numbers of algorithms on three datasets.

Datasets	P-FS	GAFS
Gas	30	33
Musk	50	53
Ulc	57	50
Oil	549	561

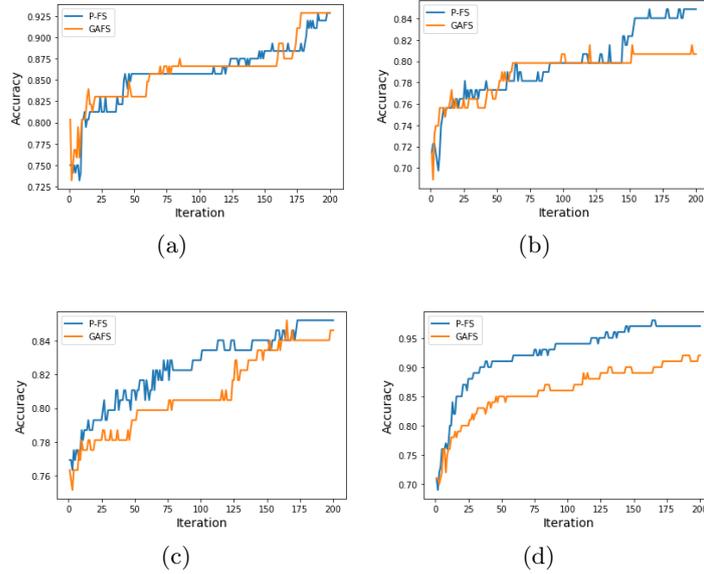


Fig. 3. Classification accuracy curve of p-fs algorithm and GAFS algorithm. (a) Gas data set (b) Musk data set (c) Ulc data set (d) Oil data set.

4.4 Discussion

In this section, we mainly discuss the advantages and applicability of P-FS algorithm. We have shown the advantages of our P-FS algorithm in feature extraction. From experiments, on the one hand, we can see that P-FS algorithm uses the parallel processing ability of cell-like P system to expand the search ability of feature space, and at the same time uses the communication between membranes to search the optimal region faster. On the other hand, the mutation factor can help the algorithm jump out of the local optimum and improve the ability of feature space search. In addition to the advantages mentioned above, our P-FS algorithm also has some limitations. Firstly, the initialization of the population has a great influence on the search results; Secondly, the algorithm takes a lot of computing resources and takes a long time; Finally, the algorithm only uses the fitness value as the evaluation standard, and the number of features selected is more, but the less the number of features selected, the better.

In future work, we plan to optimize our P-FS algorithm from the following aspects: first, the initialization method. In the experiment, we find that the initial subset has a great influence on the final result of the algorithm. Later, we want to initialize the population of the algorithm by the filtering method in order to improve the stability of the algorithm. Secondly, the algorithm is only tested on four datasets, and only compared with the performance of GAFS algorithm, the work is relatively small. In the future, we will use additional public datasets and fitness functions to verify the feasibility of the algorithm as comprehensively as

possible, and compare the performance with other feature selection algorithms to study the advantages of the proposed algorithm. Finally, the kernel of our proposed method is GA. In the later stage, the authors hope to design a feature selection algorithm that mimics the biofilm structure, and provide a new idea for the application research of membrane calculation.

References

1. Paun G. Computing with membranes. *Computer and System Sciences*, 2000, 61(1): 108-143.
2. Paun G, Rozenberg G, Salomaa A. *Handbook of membrane computing*. Oxford: Oxford University Press, 2009: 35-40.
3. Krishna S N. Universality results for P systems based on brane calculi operations. *Theoretical Computer Science*, 2007, 371(1-2): 83-105.
4. Dong H, Sun J, Sun X, et al. A many-objective feature selection for multi-label classification. *Knowledge-Based Systems*, 2020, 208(7): 106456.
5. Farahat A K, Ghodsi A, Kamel M S. Efficient greedy feature selection for unsupervised learning. *Knowledge & Information Systems*, 2013, 35(2): 285-310.
6. Welikala R A, et al. Genetic algorithm based feature selection combined with dual classification for the automated detection of proliferative diabetic retinopathy. *Comput Med Imaging Graph*, 2015, 43: 64-77.
7. Singh U, Singh S N. A new optimal feature selection scheme for classification of power quality disturbances based on ant colony framework - ScienceDirect. *Applied Soft Computing*, 2019, 74:216-225.
8. Wu J, Lu Z, Long J. A novel hybrid genetic algorithm and Simulated Annealing for feature selection and kernel optimization in support vector regression // *Information Reuse and Integration (IRI)*, 2012 IEEE 13th International Conference on. IEEE, 2012.
9. Xue Y, Xue B, Zhang M. Self-Adaptive Particle Swarm Optimization for Large-Scale Feature Selection in Classification. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2019.
10. Ning Fang-hua, Zhou Wei-zong, Zhang Feng-ying, et al. The architecture of cloud manufacturing and its key technologies research // *Proc of IEEE International Conference on Cloud Computing and Intelligence Systems*. Piscataway, NJ: IEEE Computer Society, 2011: 259-263.
11. Cortes C, Vapnik V. Support-Vector Networks. *Machine Learning*, 1995, 20(3):273-297.

GPU simulations of spiking neural P systems on modern web browsers

Arian Allenson M. Valdez^{1*}, Filbert Wee¹, Francis George C. Cabarle¹, Miguel Á. Martínez-del-Amor²

¹Dept. of Computer Science, University of the Philippines Diliman, Diliman 1101 Quezon City, Philippines.

²Department of Computer Science and Artificial Intelligence, Universidad de Sevilla

Abstract. In this work we present a novel and proof of concept Spiking Neural P system (for short, SN P systems) simulator that runs on modern web browsers whilst using graphics processing units (for short, GPUs). By creating an SN P system that both utilizes the GPU and runs on modern web browsers, we allow a much more performant SN P simulator that would also be a lot more accessible for researchers to experiment with, and can be integrated into other tools or visualizations transparently without having to learn specific GPU knowledge or techniques.

Using previous results on representing SN P system computations using linear algebra, we analyze and implement a computation simulation algorithm on web browsers that runs on the GPU. Since web browsers (at this time) do not have any capabilities for General Purpose computing on GPUs (for short, GPGPU), we exploit the Web Graphics Library (for short, WebGL) and create shaders to generate textures that correspond to computational results of our SN P simulation algorithm. To our knowledge, this is the first work on simulating SN P systems on browser GPUs.

Here, we present two different implementations and algorithms as case studies to analyse and compare the performance of the simulations, with particular interest in speedup compared to CPU approaches.

Keywords: Spiking Neural P Systems, GPU, Web browsers, Matrix Representation, Simulation

1 Introduction

Spiking neural P systems (in short, SN P systems) are parallel modes of computation inspired by the functioning and structure of neurons that was introduced in 2016 in [18].

* corresponding authors: me@arianv.com, fccabarle@up.edu.ph

Since P systems were introduced, many simulators using different parallel devices have been produced [14], including CPU clusters [13], as well as GPUs [10,11]. These efforts show that parallel devices are very suitable in simulating P systems, at least for the system variants to have been introduced. GPUs are currently one of the foremost candidates for simulating P systems due to several significant reasons. One is that GPUs offer large speedups versus CPU only implementations (including clustered CPUs), by consuming less energy at the fraction of the cost of setting up and maintaining CPU clusters [19]. Parallel computing concepts such as hardware abstraction, scaling, and so on, are also handled efficiently by current GPUs [19].

Another reason is that GPGPU computing (general purpose computations on the GPU), which is specifically designed for massively parallel computations, mean that GPU architecture are laid bare to programmers [17]

Given that SN P systems have already been represented as matrices [21], and in the GPU [10,11], there is considerable existing and evolving efforts for SN P simulations on the GPU. However, there is a distinct lack of tools for simulating SN P systems on the web browser. While CPU simulators such as [6] or Web-Snapse [15] exist, no GPU simulators for SN P systems on the browser exists to our knowledge. Bringing GPU-based SN P simulators on the web browser is the main focus of our research.

We hypothesize that such a simulator should allow for a more performant system to run on the web. Running it on such an environment should also allow for a more accessible area for researchers to experiment with, and allow for easier integration with other tools [16,6] or visualizations transparently without having to learn specific GPU knowledge or techniques.

In this work, we present a novel way to implement a GPU based SN P system on the browser using two implementations of SN P simulators. We give definitions for the data structures that would work in this new environment, as well as the algorithms themselves and benchmarks to compare their performance with their CPU counterparts. Limitations of our approach will also be pointed out, as well as work for further research.

This paper is organized as follows: Section 2 provides the preliminary definitions for Spiking Neural P Systems. SN P systems are formally defined, with some notions and notations from formal language theory used throughout the paper. Section 3 provides information for the technologies and techniques used for our simulation. Section 4 presents the simulation algorithms and their corresponding analysis, as well as the benchmark and stress tests for the simulation. The test hardware setup are also presented in the same section, together with the results. Lastly, we provide conclusions and notes for future work.

2 Spiking Neural P Systems

The reader is assumed to be familiar with basics of membrane computing and formal language theory. We only briefly mention notions and notations which will be useful throughout the paper, as was done in the seminal paper for SN P systems [18]

A Spiking neural P system is of the form:

$$\Pi = (O, \sigma_1, \dots, \sigma_m, syn, in, out), \text{ where:}$$

1. $O = a$ is the alphabet containing a single symbol (the spike);
2. $\sigma_1, \dots, \sigma_m$ are neurons of the form $\sigma_i = (a_i, R_i), 1 \leq i \leq m$, where:
 - (a) $a_i \geq 0$ is the initial number of spikes contained in σ_i .
 - (b) R_i is a finite set of rules of the following two forms:
 - i. (Spiking Rule) $E/a^c \rightarrow a^p; d$ where E is a regular expression over O and $c \geq p \geq 1, d \geq 0$
 - ii. (Forgetting Rule) $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a^p; d$ of type (i) from R_i , we have $a^s \notin L(E)$;
3. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ with $i \neq j$ for all $(i, j) \in syn, 1 \leq i, j \leq m$ (synapses between neurons);
4. $in, out \in \{1, 2, \dots, m\}$ indicate the input and the output neurons respectively.

To further expound on the rules mentioned in (b), the spiking rule $E/a^c \rightarrow a^p; d$ can be applied to a neuron σ_i that contains k spikes and $a^k \in L(E), k \geq c$. That is to say, c spikes are to be removed from the neuron σ_i so that $k - c$ spikes remain in the neuron which fires off p spikes after d time. During the d time of processing a rule, the neuron σ_i is said to be *closed* and spikes fired to this neuron during this time will be lost. The forgetting rule, on the hand, may be applied to a neuron σ_i with s spikes. All of the s spikes will be removed from the neuron σ_i .

In the case where more than one rule may apply, only one rule will be applied. That rule is non-deterministic chosen by the system. A *configuration* of the system at step t is shown as $C_t = \langle r_1/k_1, \dots, r_m/k_m \rangle$ for $1 \leq i \leq m$, where σ_i contains r_i . A *transition* is the "movement" of the system from one configuration to another. A *computation* is any sequence of configuration such that: (a) the first term is the initial configuration, and (b) any other configuration is obtained from the previous configuration in one transition. A computation may either be finite or infinite. In the case of a finite computation (also known as a *halting computation*), the last term of the sequence (also called a *halting configuration*) contains all open neuron and no rule may be applied.

3 Simulator Technologies

The following discussion details the specific technologies and techniques used in this paper to allow SN P computations on the web browser.

There is currently no support for General Purpose computing on GPUs (GPGPU) on web browsers. However, research and implementation is being done in this area [7]. Due to its current unavailability in contemporary web browsers, we use the Web Graphics Library (WebGL) in this research to exploit the GPU and obtain results for SN P simulation.

Web Graphics Library (WebGL) is a JavaScript API for rendering high-performance interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins. WebGL does so by introducing an API that closely conforms to OpenGL ES 2.0 that can be used in HTML5 canvas elements. This conformance makes it possible for the API to take advantage of hardware graphics acceleration provided by the user's device. [5]

While the stable release for WebGL 2.0 is relatively recent (2017) [2], support for WebGL is almost universal (currently rated at 97.6% of Global Users), with most modern browsers supporting the technology (Firefox 4+, Google Chrome 9+, Opera 12+, Safari 5.1+, Internet Explorer 11+, and Microsoft Edge build 10240+) [5,1].

Support is also dependent on hardware GPU support, and may not be available on older devices. More info can be seen in the Khronos Whitelist And Blacklists page [3]. The official WebGL website offers a simple test page at <http://get.webgl.org/>

As implied by the name, WebGL is a technology suited for graphics processing, and is not really made with GPGPU in mind. Indeed, WebGL is an immediate mode 3D rendering API designed for the web. To use the WebGL API, we must obtain a WebGLRenderingContext object for a given HTMLCanvasElement or OffscreenCanvas. This object is used to manage OpenGL state and render to the drawing buffer, which must be created at the time of context creation. [4]

The drawing buffer into which the API calls are rendered shall be defined upon creation of the WebGLRenderingContext object.

The Drawing Buffer Table 1 shows all the buffers which make up the drawing buffer, along with their minimum sizes and whether they are defined or not by default. The size of this drawing buffer shall be determined by the width and height attributes of the HTMLCanvasElement or OffscreenCanvas. The table below also shows the value to which these buffers shall be cleared when first

created, when the size is changed, or after presentation when the `preserveDrawingBuffer` context creation attribute is false. [4]

Buffer	Clear value	Minimum size	Defined by default?
Color	(0, 0, 0, 0)	8 bits per component	yes
Depth	1.0	16 bit integer	yes
Stencil	0	8 bits	no

Table 1. The Drawing Buffer Table

Rendering with OpenGL ES 2.0 requires the use of shaders, written in OpenGL ES’s shading language, GLSL ES. Shaders must be loaded with a source string (`shaderSource`), compiled (`compileShader`) and attached to a program (`attachShader`) which must be linked (`linkProgram`) and then used (`useProgram`). [4]

In this research, we use a specific technique in exploiting WebGL to allow us to implement operations on the GPU. We create specific shader code that would ‘render’ texture data that corresponds to computational results of our SN P simulation algorithm. We map each individual data point from the matrix representation [21] of an SN P configuration to as input textures and run our shaders to generate resulting textures corresponding to an output SN P matrix.

4 Algorithms and Experimental Results

We present two algorithms and implementations in our research. The first one we detail is an SN P simulator that supports Non Deterministic SN P without delays [10]. The second SN P simulation we detail is an SN P simulator that supports Deterministic SN P with support for delays [11].

We implement the algorithm with typescript 4.1.2, targeting native JavaScript. Our code structure is split up in 3 yarn packages, ‘gpusnapse/snp’, which represents the core algorithm, ‘gpusnapse/benchmarks’ which is a simple ‘node’ script that would run the simulations using a ‘node’ implementation of the canvas, and ‘gpusnapse/web’, which runs a web server that can run the simulation.

We run the experiments on a MacBook Pro (13-inch, 2018, Four Thunderbolt 3 ports) model, with an Intel Iris Plus Graphics 655 GPU - with 384 Shading Units, and non-dedicated shared memory (8GB). The browser used is Google Chrome 90.0.4430.

4.1 Algorithm 1: SN P Simulation with no delays

Our first SN P systems simulation algorithm relies on the same matrix representation and algorithms given in [10].

Algorithm 1: Overview of SN P system simulation. **SEQ** and **PAR** indicate which step is done sequentially or in parallel, respectively

Require: Inputs: $C_0, M_{II}, R, \#_{C_k}$ of SN P system II for

$1 \leq i \leq n, 1 \leq j \leq m.$;

0. (**SEQ**) Load inputs: $M, R,$ and C_0 are loaded once only;

I. (**SEQ**) Load every C_{k+1} afterwards for $k \geq 0$;

II. (**SEQ**) With respect to current C_k , determine if a rule $r_i \in R_j$ is applicable by checking if $a^{a_j} \in L(E)$ for E associated with the rule $r_{i1}, \alpha_j \in C_k, R_j \in \sigma_j$. Then generate all possible S_k from all applicable rules;

III. (**PAR**) Produce all C_{k+1} from all possible S_k with respect to current C_k ;

IV. (**SEQ**) Repeat steps I to IV, till all unique C_k are produced (stopping criterion(1)) or the number of computed C_k is equal to $\#_{C_k}$ (stopping criterion (2));

A CPU only version of Algorithm 1 is also created so that step III of the algorithm is done sequentially and we designate this as *snpcpu* while the CPU-GPU simulator is designated as *snpgpu*.

4.2 Algorithm 1: Runtime comparisons

For Algorithm 1, we use the same benchmark SN P system defined in [10] to analyze the runtime of our SN P system.

Figure 1 shows the running time of *snpcpu* and *snpgpu* simulators with regards to our parameters. As the number of neurons and rules exponentially increased, the resources in the GPU are better fulfilled. We report a 2x speedup increase for 14 neurons in our benchmark, which is roughly consistent with the results from [10]

4.3 Algorithm 2: SN P Simulation with delays and GPU

This particular representation and algorithm supports delays, which Algorithm 1 does not support. This specific algorithm also reduces the amount of host-device data transfers by having more steps from Algorithm 1 be done entirely through the GPU.

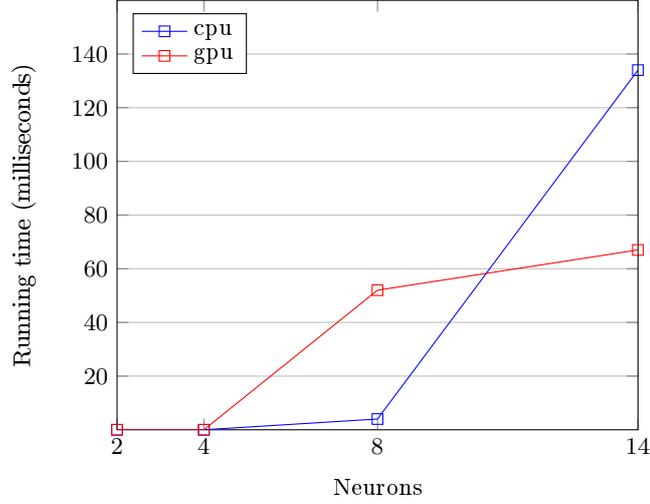


Fig. 1. Running time of *snpcpu* versus *snpgpu*

Our second SN P systems simulation algorithm is based on the same matrix representation and algorithms given in [11]. We modify the calculation of certain vectors and matrices in order to work with specific WebGL constraints.

In particular, we cannot use shared memory for our implementation, rendering the main algorithm as described in [11] unusable. We introduce modifications of the matrix representations to simulate SN P systems with delay on WebGL. Let Π be an SN P system with delay having m neurons and n rules. We use the following definitions to represent Π .

Definition 1 (Configuration Vector). The configuration vector $C^{(k)} = \langle c_1, \dots, c_n \rangle$ where c_i is the amount of spikes in σ_i at time k .

Definition 2 (Spiking Vector). A **spiking vector** $S^{(k)} =$ where

$$s_i^{(k)} = \begin{cases} 1, & \text{if } E_i \text{ is satisfied and } r_i \text{ is applied,} \\ 0, & \text{otherwise.} \end{cases}$$

Definition 3 (Status Vector). The k th status vector is denoted by $St^{(k)} =$ where for each $i \in \{1, 2, \dots, m\}$,

$$st_i = \begin{cases} 1, & \text{if neuron } m \text{ is open,} \\ 0, & \text{if neuron } m \text{ is closed} \end{cases}$$

Definition 4 (Rules). $R^{(k)} = \langle r_1, \dots, r_n \rangle$ where r_i is the regular expression for rule i .

Definition 5 (Neuron index vector). $ni = \langle n_1, \dots, n_m \rangle$ where ni_i is the starting index of the rules for neuron i on R

Definition 6 (Rule index vector). $RI = \langle RI_1, \dots, RI_m \rangle$ where RI_i is the ending index of the rules for neuron i on R

Definition 7 (Current Delays Vector). $d' = \langle d'_1, \dots, d'_n \rangle$ where for each $i = 1, \dots, n$:

$$d' = \begin{cases} -1 & \text{if the rule is not fired,} \\ 0, & \text{if the rule is fired,} \\ \geq 1, & \text{if the rule is currently on delay (i.e. neuron is closed).} \end{cases}$$

Definition 8 (Delay Vector). The delay vector $D = \langle d_1, \dots, d_n \rangle$ contains the delay count for each rule $r_i, i = 1, \dots, n$ in Π

Definition 9 (Loss Vector). The loss vector $LV^{(k)} = \langle lv_1, lv_2, \dots, lv_m \rangle$ where for $i \in \{1, 2, \dots, m\}$, lv_i is the number of consumed spikes in σ_i at the step k .

Definition 10 (Gain Vector). The k th gain vector is denoted by $GV^{(k)} = \langle gv_1, gv_2, \dots, gv_m \rangle$ where for each $i \in \{1, 2, \dots, m\}$, gv_i is the number of spikes sent by neighboring neurons to neuron σ_i at the step k .

Definition 11 (Transition Matrix). The transition matrix of Π , is an ordered set of vectors TV defined as $TV = \{tv_1, \dots, tv_n\}$ where for each $i \in \{1, 2, \dots, n\}$, $tv_i = \langle p_1, \dots, p_m \rangle$ such that if $r_i \in \sigma_s$:

$$p_j = \begin{cases} \text{number of spikes produced by } r_i, \text{ if } (s, j) \in \text{syn} \\ 0, \text{ otherwise.} \end{cases}$$

Definition 12 (Indicator Matrix). The indicator vector $IV^k = \langle iv_1, \dots, iv_m \rangle$ indicates which rule will produce spikes at time k .

Definition 13 (Removing Matrix). The removing matrix of Π is $RM = \{rm_1, rm_2, \dots, rm_n\}$ where for each $i \in \{1, 2, \dots, n\}$, $rm_i = \langle t_1, \dots, t_m \rangle$ such that if $r_i \in \sigma_s$:

$$t_j = \begin{cases} \text{number of spikes consumed by } r_i, \text{ if } s = j \\ 0, \text{ otherwise.} \end{cases}$$

Definition 14 (Net Gain Matrix). The **Net Gain vector** of Π at step k is defined as $NG^{(k)} = C^{(k+1)} - C^{(k)}$

With these definitions, we can compute the next configuration as follows:

$$C^{(k+1)} = C^{(k)} + St^{(k)} \otimes (IV^{(k)} \cdot TV) - S^{(k)} \cdot RM$$

Procedure: Compute $C^{(k+1)}(C^{(k)}, R, Tv, St^{(k)})$

```

Compute  $LV^{(k)}$ 
Compute  $IV^{(k)}$ 
 $GV^{(k)} \leftarrow TV * IV^{(k)}$ 
 $NG^{(k)} \leftarrow GV^{(k)} \otimes St^{(k)} - LV^{(k)}$ 
 $C^{(k+1)} \leftarrow C^{(k)} + NG^{(k)}$ 
return  $C^{(k+1)}$ 

```

Procedure: Compute $St^{(k)}$

```

 $ri \leftarrow ni_{(i)}$ ;
if  $ri \leq 0$  then
  | return 0
end
 $v \leftarrow 1$ ;
while  $ri \leq RI_{(i)}$  do
  | if  $S_{ri}^k = 1$  then
  | | if  $d'[i] = 0$  then
  | | |  $v \leftarrow 1$ 
  | | | else
  | | | |  $v \leftarrow 0$ 
  | | | end
  | | else
  | | |  $v \leftarrow 1$ 
  | | end
  | |  $ri \leftarrow ri + 1$ 
end
 $St^{(k)} = v$ 

```

4.4 Algorithm 2: Runtime comparisons

Algorithm 2 is able to simulate only deterministic SN P systems with delays. For our runtime comparison, we implement a bitonic sorting network provided in more detail in [12] and compare the runtimes of sequential (CPU) and parallel (GPU) simulators.

Generalized sorting networks with input size of 128, 160, 192 were generated for testing. The simulators were run for 1 single step and the input number used for sorting was randomly generated from 0 - 99 inclusive allowing repetitions. The 128-input generalized sorting network (smallest size) has 7,296 neurons and 10,752 rules, while the largest (192-input) has 10,944 neurons and 16,128 rules.

As shown in figure 2, the GPU simulator shows a slower performance in the three input sizes being tested, however, the speedup does seem to indicate an upward trend. We conjecture that the performance increase could potentially keep increasing as the input size grows larger, eventually reaching a "sweet spot"

Procedure: Compute $LV^{(k)}$

```
ri ← ni(i);
LVi(k) = 0
while ri ≤ RIri do
  if S(k) = 1 then
    LVi(k) = C(ri)
    return
  end
  ri ← ri + 1
end
```

Procedure: Compute $S^{(k)}$

```
ri ← ni(i)
while ri ≤ RIri do
  if Stj(k) = 0 then
    Si(k) ← 0
  else
    if L(Ei) matches Cj(k) then
      Si(k) ← 1
    else
      Si(k) ← 0
    end
  end
  ri ← ri + 1
end
```

where the GPU solution would be faster. Unfortunately, testing larger input sizes were not feasible at the environment we tested in due to memory constraints.

Research from [11] show slower performance with smaller input sizes, with a speedup of greater than 1 at about the 128-input. There are a couple of reasons why our current implementation is currently slower for such an input size, the first one being the general overhead of running our simulation on a browser, which is effectively a virtual machine running non-natively on the user's machine.

Furthermore, the nature of our WebGL texture computation technique relies on 'rendering' computations on a resulting texture for most operations. This means that each non-trivial GPU computation would result in a texture held in the GPU's memory, increasing the general amount of memory required to do computations. This also means that shared memory cannot be feasibly used in our implementation, resulting in a linear time increase for computing certain vectors such as the Status Vector.

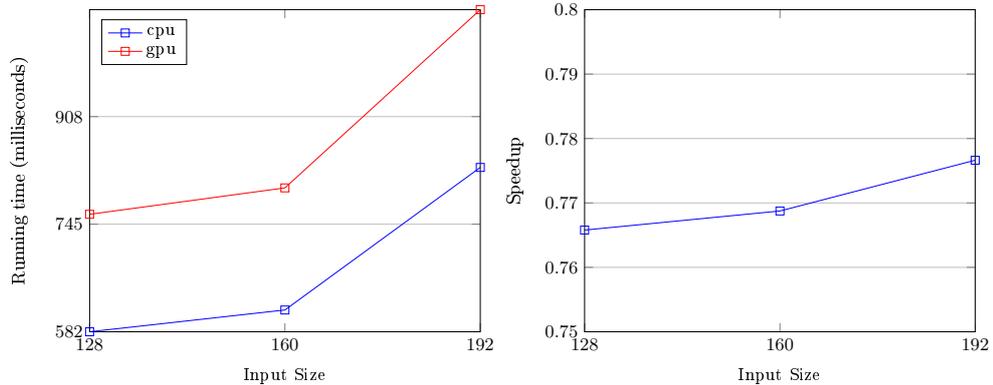


Fig. 2. Running time of *snpcpu* versus *snpgpu*

5 Final Remarks

In this work, we presented a proof of concept SN P system simulator that runs on modern web browsers whilst using graphics processing units. Furthermore, we were able to present two algorithms that simulate both a non-deterministic SN P system that does not support delays, and a deterministic SN P system that supports delays.

We were able to implement and present new algorithms for both, resulting in a trend very consistent with previous research from [10] and [11] running in the context of a web browser.

With our benchmark tests for Algorithm 1, we were able to show a 2x speedup. At the same time, we were able to simulate a bitonic sorter in the GPU for Algorithm 2, with an upwards trend for speedup as the input-size grows larger.

We also plan on publishing our resulting implementation at '@gpunapse/core' as an *npm* library allowing interoperable use for other use cases such as visualizations. *npm* is the default package manager for the JavaScript runtime environment *Node.js*.

There is a significant area of possible improvements for future work. We have implemented a family of libraries under the @gpunapse namespace (@gpunapse/core for the SN P system simulation, @gpunapse/web for testing it online, and @gpunapse/benchmarks for benchmarking), and we intend to create a parser that can accept .pli files (input files for the P-Lingua simulator [20]).

There might also still be possible improvements that could be done for Algorithm 2. In particular, a closed form formula for some vectors, or reducing

the amount of intermediary textures can increase the performance and reduce memory. Using sparse matrix-vector operations such as in [8] can also lead to reducing memory requirements. Simulating Numerical Spiking Neural P Systems [9] is another avenue of research.

At the same time, general improvements to the calculations can also be made to the shader code. Our matrix operations for example, does not rely on parallel reduction, which should be possible using a texture window algorithm.

With the continuing evolution of WebGPU, it might be of interest to investigate implementing the algorithm using the heimtechnology, as the functionality should eventually arrive to web browsers at a future date. This can give even more significant performance increase, as actual GPGPU capabilities would be available in it.

Because our implementation is done through GLSL ES, one other possible area for future work is deploying it for native runtimes, allowing a common, shared language to be used for both native runtimes and the web. This can allow us to advance the state of the art in SN P Simulations and target both native and web with one common, shared API and language. This can be even more fruitful if WGSL is used, as it is the shader language for WebGPU, which would remove the inherent limitations in this research and is made for creating actual compute kernels, possibly providing almost native performance. Research in this area must be careful not to introduce any performance regressions.

References

1. Can I Use: WebGL - 3D Canvas graphics. <https://caniuse.com/webgl>
2. Khronos, WebGL 2.0 Arrives. <https://www.khronos.org/blog/webgl-2.0-arrives>
3. Khronos: WebGL Blacklists And Whitelists. <https://www.khronos.org/webgl/wiki/BlacklistsAndWhitelists>
4. Khronos, WebGL specs. <https://www.khronos.org/registry/webgl/specs/latest/1.0/#1>
5. MDN Web Docs: WebGL API. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
6. Secretmapper/Snaps. <https://github.com/Secretmapper/Snaps/>
7. W3C Working Draft, WebGPU. <https://www.w3.org/TR/webgpu/>
8. Martínez-del Amor, M.Á., Orellana-Martín, D., Pérez-Hurtado, I., Cabarle, F.G.C., Adorna, H.N.: Simulation of spiking neural p systems with sparse matrix-vector operations. Processes 9(4) (2021), <https://www.mdpi.com/2227-9717/9/4/690>
9. Ballesteros, K.J., Cailipan, D.P.P., Cruz, R.T.D.L., Cabarle, F.G.C., Adorna, H.N.: Matrix representation and simulations of numerical spiking neural p systems. (submitted) International Conference on Membrane Computing (ICMC2021), Chengdu, China and Debrecen, Hungary, August 24-28, 2021 (2021)

10. Cabarle, F., Adorna, H., Martínez-del Amor, M., Pérez-Jiménez, M.: Improving gpu simulations of spiking neural p systems. *Romanian Journal of Information Science and Technology* 15, 5–20 (01 2012)
11. Carandang, J.P., Villaflores, J.M.B., Cabarle, F.G.C., Adorna, H.N., Martínez del Amor, M.Á.: Cusnp: Spiking neural p systems simulators in cuda. *Romanian Journal of Information Science and Technology (ROMJIST)*, 20 (1), 57-70. (2017)
12. Ceterchi, R., Tomescu, A.I.: Implementing sorting networks with spiking neural p systems. *Fundam. Inf.* 87(1), 35–48 (Jan 2008)
13. Ciobanu, G., Wenyuan, G.: P systems running on a cluster of computers. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing*. pp. 123–139. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
14. Díaz-Pernil, D., Graciani-Díaz, C., Gutiérrez-Naranjo, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Software for p systems. *The Oxford Handbook of Membrane Computing* pp. 437–454 (2010), <http://www.us.oup.com/us/catalog/general/subject/Mathematics/ComputerScience/?view=usa&sf=toc&ci=9780199556670>
15. Dupaya, A.G.S., Galano, A.C.A.P., Cabarle, F.G.C., Cruz, R.T.D.L., Macababayao, I.C.H., Ballesteros, K.J., Lazo, P.P.L.: A web-based visual simulator for spiking neural p systems. (submitted) *International Conference on Membrane Computing (ICMC2021)*, Chengdu, China and Debrecen, Hungary, August 24-28, 2021 (2021)
16. Fernandez, A.D.C., Fresco, R.M., Cabarle, F.G.C., de la Cruz, R.T.A., Macababayao, I.H., Ballesteros, K.J., Adorna, H.N.: Snapse: A visual tool for spiking neural p systems. *Processes* 9(1) (2021), <https://www.mdpi.com/2227-9717/9/1/72>
17. Harris, M.: Mapping computational concepts to gpus. In: *ACM SIGGRAPH 2005 Courses*. p. 50–es. SIGGRAPH '05, Association for Computing Machinery, New York, NY, USA (2005), <https://doi.org/10.1145/1198555.1198768>
18. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural p systems. *Fundamenta informaticae* 71(2, 3), 279–308 (2006)
19. Kirk, D.B., Hwu, W.m.W.: *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. (2010)
20. Macías-Ramos, L.F., Pérez-Hurtado, I., García-Quismondo, M., Valencia-Cabrera, L., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A p–lingua based simulator for spiking neural p systems. In: Gheorghe, M., Păun, G., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) *Membrane Computing*. pp. 257–281. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
21. Zeng, X., Adorna, H., Martínez-del Amor, M.Á., Pan, L., Pérez-Jiménez, M.J.: Matrix representation of spiking neural p systems. In: *International conference on membrane computing*. pp. 377–391. Springer (2010)

Target-indicating Spiking Neural P Systems with Polarizations and Numerical Value

Qi Wu¹ and Yuzhen Zhao^{1*}

Shandong Normal University, Jinan, 250307, Shandong, China

Abstract. By extending the concept of target indication to the spiking neural P systems, a new spiking distribution mechanism is introduced into the spiking neural P systems, and the resulting model is called the spiking neural P systems with target indications. On the basis of the spiking neural P systems with target indications, polarization rules and numerical values are added, and a target-indicating spiking neural P system with polarizations and numerical values is proposed. In this system, in addition to the neuron execution rules that can send numerical values to other neurons, the execution of the neuron rules also needs to consider the polarity of the charge carried by the neuron, and the system also expands the previous standard spiking objects to numerical values. The calculation versatility of the target-indicating spiking neural P systems with polarizations and numerical value as an arbitrary natural number generation and recognition device is studied. The results show that the system can calculate any Turing computable natural number and has the same computing power as the Turing machine.

Keywords: Spiking neural P systems · Target indications · Polarization · Numerical value · Turing universality

1 Introduction

In 1998, Gheorghe Paun put forward the concept of membrane computing in a research report of the Turku Computer Center in Finland, which opened up a new field of computer science. Membrane computing is a biological computing method abstracted from the most basic unit of living organisms cells. Its computing model is called membrane system or P system. The P system is composed of three parts: membrane structure, objects and evolution rules. Objects and evolution rules are placed in the area surrounded by a membrane, and objects evolve continuously according to the evolution rules of the area. Under normal circumstances, the P system runs in an extremely parallel mode, which has the same computing power as the Turing machine, and even has the possibility of surpassing the limitations of the Turing machine. The proposal of the P system brings a new distributed and parallel computing model to computer science.

*Corresponding author: Yuzhen Zhao

The P system is divided into three main categories: the cell-like P systems, the tissue-like P systems, and the neural-like P systems. There have been some studies on cell-like and tissue-like P systems in recent years. A cell-like P system with evolutionary cooperative port/antiport rules and membrane production is proposed in [30]. A cell-like P system with polarization and minimal rules, and studied the computing power of the P system with polarization and minimal rules [18]. A new variant of the organization P system [31], the P system improved the original design and allowed the parallel mechanism to be used in the application of rules on the channel to improve the operating efficiency of the system.

Among the neural-like P systems, the spiking neural P systems have been proven to have powerful computing capabilities and have been widely concerned by researchers. The spiking neural P systems are a kind of computational model inspired by the phenomenon that neurons process information in the form of spiking in biological neural networks and use spiking for information interaction between neurons. It belongs to a kind of neural-like P systems. Ionescu et al. develop it first in 2006. For the standard spiking neural P systems, their topological structure can be expressed as a directed graph, in which the nodes of the directed graph are neurons, and the edges of the directed graph are the synapses connecting the neurons to each other. There are mainly two types of rules in the spiking neural P systems: firing and forgetting rules. Standard firing rules indicate that when the regular expression is satisfied, the neuron consumes some of its own spikings and sends them to other neurons after a period of time; the standard forgetting rule is that the neuron consumes its own spiking, and neither produces new ones. The numerical value does not send numerical values to other neurons. There is an output neuron in spiking neural P systems. The output neuron can not only send spiking to neighboring neurons, but also send spiking to the environment to output calculation results. In the working process of the spiking neural P systems, neurons execute rules. When there are no rules that can be used in the systems, the systems stop working and output the calculation results. As the third-L generation neural network model, the spiking neural P systems have been proven to have powerful computing capabilities, so they have always been a hot issue in the field of membrane computing research.

In recent years, researches on spiking neural P systems are increasing. Researchers have proposed many variants of the standard spiking neural P systems. The following is a detailed introduction from the theoretical research, application research, simulation and realization research of the spiking neural P systems.

Self-organizing SNP systems [9], SNP systems with neuron division and bud value [35], SNP systems with only neuron division [19], which introducing neuron dissolution rules into the system to delete redundant neurons. SNP systems with neuronal division and dissolution [45], the SNP-IR systems based on inhibitory synapses [13]. SNP systems with structural plasticity [24]. On this basis, weighted SNP systems with structural plasticity [42] work under the maximum peak strategy. Neurons can use plasticity rules to Create and delete synapses. Research on system objects, rules, system operation, refer to [35, 39, 25, 44, 32, 40, 20, 21, 28, 36, 26, 33, 38, 12, 2, 41, 43, 34, 23]. About the computing power and

complexity of SNP systems, refer to [17, 8, 3, 37, 1, 22, 29]. Some progress has also been made in the application research of SNP systems, refer to [7, 15, 16, 27, 11, 5, 14, 6]. Research progress on simulation and realization of SNP systems, refer to [10, 4].

In the spiking neural P systems and most of their variants, the spiking distribution mechanism generates a specific number of spiking and distributes them to all target neurons. Nevertheless, this spiking distribution mechanism is subject to two substantial limitations, that is, for each neuron, the same group of target neurons (1) must be fixed, and for each rule. (2) the number of spiking sent to all target neurons must be the same. In response to such shortcomings, in order to alleviate the limitation, a more flexible spiking distribution mechanism is proposed, that is, by extending the concept of target indication to the spiking neural P systems, the new system is introduced into a new spiking distribution mechanism. This new spiking distribution mechanism allows different rules in neurons to act on different target neurons, instead of requiring all rules in neurons to act on the same target neuron defined in the standard spiking neural P systems.

The calculation cost of the integrated conditions for rules in the form of regular expressions is very high, which means that determining whether a natural number is included in the language length set described by the regular expressions is a defect of NP-complete problem, and inspired by the biological phenomenon that every neuron has a positive or negative charge. We considered a new mechanism to control the application of the rules: that is, using three kinds of charges (positive, neutral, and negative) instead of using the regular expressions in the original definition of the spiking neural P systems.

In view of the symbolic data representation of some spiking forms that make it difficult for the spiking neural P systems to solve the problem of numerical information, numerical variables are introduced into the systems. In fact, many applications are encountered in practical engineering and scientific fields, such as the control of autonomous robots and industrial process control, which involve the numerical characterization of a large amount of information and require precise and quantitative modeling. Therefore, the computational model needs to be capable of processing numerical information ability.

The above expansion of the standard spiking neural P systems, to a certain extent, enables spiking neural P systems to have more flexible syntax and stronger control capabilities. It also proves that the new systems can be used as a Turing general receiving and generating device, and a small general-purpose computing device is constructed. It also shows that the spiking neural P systems have powerful computing power. The above variants construct a new system by changing the rules and objects. From this we put forward an idea: on the basis of three models, combining the advantages of each variant to expand a brand new spiking neural P systems, so will the brand new spiking neural P systems have even more amazing computing power? This is worth studying.

This paper adds two biological phenomena on the basis of target indication:

- (1) In biology, the electric potential in a neuron is actually a real number, so we add the numerical value as an object to the system.
- (2) Since the internal environment of the neuron will affect the occurrence of the reaction, for example, nerve impulses can only be generated and transmitted under action potentials. We use three kinds of charges, positive, negative and neutral to control the use of rules. This method is called polarization rules.

This paper proposes a new variant of the spiking neural P systems, called the target-indicating spiking neural P systems with polarizations and numerical value (abbreviated as SNP-TIPN systems). The form of the firing rules are $\alpha/a^c \rightarrow a^p(\text{tar});\beta$. In the firing rules, α represents the polarity charge of the current neuron, β represents the polarity charge sent by the current neuron to neighboring neurons, $\text{tar} \subseteq \{1, 2, \dots, m, \text{env}\}$ are the receiving neurons. In the standard spiking neural P systems, the use of firing rules and forgetting rules must meet certain conditions, that is, regular expressions must be met, and the number of spiking in the neuron must be greater than or equal to the number of spiking consumed by the rules. Therefore, in the SNP-TIPN systems, polarizations and numerical value are designed for indicating firing. Only when the neuron has a certain charge and the number of values in the neuron meets the condition, the rule can be executed. After the rule is executed, the product is directed to be sent.

2 Preliminaries

2.1 The Register Machine

Similar to the computational completeness work of the existing P systems, the register machine can be used to prove the Turing versatility of the P systems. There are addition, subtraction and stop instructions in the register machine, its working principle is to increase or subtract one from the value in the register r , and at the same time jump to the next instruction l_j or l_k non-deterministically. The register machine is a five-tuple $M = (m, H, l_0, l_h, I)$, and its instruction form is as follows:

- (1) $l_i : (ADD(r), l_j, l_k)$ (add 1 to register r , then transfer to instruction l_j or instruction l_k);
- (2) $l_i : (SUB(r), l_j, l_k)$ (if the number in the register r is greater than 0, decrease by 1, and then enter the instruction l_j , otherwise directly enter the instruction l_k);
- (3) $l_h : \text{HALT}$ (stop command).

$N_{gen}(M)$ represents a set of numbers generated non-deterministically by the register machine M . Its working principle is as follows. In the beginning, all registers except the first register are empty, and the first register is not affected by the SUB instruction during the calculation. M starts with instruction l_0 , and then continues to execute other instructions. When it turns to the instruction l_h , the calculation is complete, and the final result is stored in the first register.

As we all know, the register machine M can calculate any Turing computable number set (denoted by NRE).

The register machine M can also be used to accept numbers. Let $N_{acc}(M)$ denote the number set accepted by M . Its working principle is as follows. In the beginning, all registers except the first register are empty, and a number is introduced to the first register. In the accepting mode, the register machine is deterministic, that is, use $l_i : (ADD(r), l_j)$ to replace $l_i : (ADD(r), l_j, l_k)$ as the ADD instruction.

3 SNP TIPN systems

3.1 Definition

A target-indicating *spiking neural P system with polarizations and numerical value* of degree $m \geq 1$ is a construct of the form:

$$\Pi = (O, \sigma_1, \sigma_2 \dots \sigma_m, syn, in, out)$$

where

- (1) $O = \{a\}$ is the singleton alphabet, a is the numerical value in the neuron, $a \in R$;
- (2) $\sigma_1, \dots, \sigma_m$ are neurons in the form: $\sigma_i = (a_i, n_i, R_i)$, $0 < i < m$;
 a_i represents the polarity charge of the current neuron;
 $n_i \geq 0$ represents the number of initial values stored by the neuron σ_i ;
 R_i is the set of rules, which have two forms:
Firing rules: $\alpha/a^c \rightarrow a^p(\text{tar})$; β , where $c \geq p$, $\alpha, \beta \in \{+, -, 0\}$, α represents the polarity charge of the current neuron, β represents the polarity charge sent by the current neuron to neighboring neurons, $\text{tar} \subseteq \{1, 2, \dots, m, env\}$ are the receiving neurons;
Forgetting rules: $\alpha/a^s \rightarrow \lambda$; β , where $s \geq 1$, $\alpha, \beta \in \{+, -, 0\}$, α represents the polarity charge of the current neuron, β represents the polarity charge sent by the current neuron to neighboring neurons;
- (3) $syn \in \{1, \dots, m\} \times \{1, \dots, m\}$, which indicates the synaptic connections;
- (4) in, out indicate the input neuron σ_{in} and the output neuron σ_{out} .

In the SNP-TIPN systems, certain conditions must be met for using the firing rules and the forgetting rules. Only when the number of numerical values in the neuron is greater than or equal to the number of numerical values consumed by the rule, and the neuron has a certain charge, the rule can be executed. If the firing rule is used, a certain number of numerical values are consumed, and the numerical values and corresponding charges are sent to the designated neuron. If the forgetting rule is used, a certain number of numerical values are removed, and the corresponding charge is sent to the designated neuron.

In the firing rules, we avoid the use of regular expressions, but consider a new mechanism to control the use of rules, that is, the application of rules of three kinds of charges (positive, neutral, and negative). The firing rule is of the

form $\alpha/a^c \rightarrow a^p(tar); \beta$, where α and β are the electrical charges. The rule applies only when the neuron has a charge α and at least c values. Not only is the number p , but the charge β is also transferred to the designated neuron. Then, the charge received by the neuron, together with the current charge of the neuron, is calculated in a natural way as the next charge of the neuron (Regardless of delay and only use the *standard firing rules* of $p = 1$ and the forgetting rules of $p = 0$). In this way, we add a significantly weaker control to the application of the rules that use these three types of charges, and send the values to designated neurons.

Similarly, in the forgetting rules $\alpha/a^s \rightarrow \lambda; \beta$, only when the neuron has a charge α and at least s values, can the value in the neuron be ablated, and the charge β is sent to the designated neuron.

The changes in the charge of neurons are as follows:

- 1) If the neuron has a neutral charge in the initial state: the neuron still has a positive charge after receiving a positive charge; the neuron still has a negative charge after receiving a negative charge; the neuron does not change the charge state after receiving a neutral charge.
- 2) If the neuron is positively charged in the initial state: the neuron still has a positive charge after receiving a positive charge; the neuron receives a negative charge and then has a neutral charge; when the neuron receives more than two negative charges, according to the calculation of the positive and negative charge, the neuron is negatively charged.
- 3) If the neuron is negatively charged in the initial state: the neuron receives a positive charge and then bears a neutral charge; the neuron receives a negative charge and still bears a negative charge; when the neuron receives more than two positive charges, according to the calculation of the positive and negative charge, the neuron is positively charged.

In particular, these rules are used sequentially in each neuron, but the neurons in the systems work in parallel. In each time step, if a certain rule meets the conditions, it must be used. When multiple rules in the neuron meet the conditions, one will be selected indefinitely for execution. At the same time, the execution of the rules meets the principle of maximum numerical value consumption, that is, the rules that consume more numerical values are always executed first, until the number of numerical values cannot meet the condition, other rules will be selected for execution. For example, two rules $0/a^u \rightarrow a^p; 0(tar)$ and $0/a^{u'} \rightarrow a^{p'}; 0(tar)$ in neuron σ_i can be applicable. In this case, only one of them will be applied according to the following criterion (maximum spike consumption strategy is adopted): if $u > u'$, then rule $0/a^u \rightarrow a^p; 0(tar)$ is chosen to apply; if $u = u'$, then one of them is non-deterministically selected.

The initial configuration of the SNP-TIPN systems can be described by the number of numerical values initially contained in each neuron, such as $C_0 = (n_1, n_2, \dots, n_i)$. By using firing rules or forgetting rules, the system will change from one configuration to another, such a process is called system conversion. Starting from the initial configuration, a series of configuration transfers are called the calculation process of the system. When the system transfers from a

certain configuration to a configuration with no rules available, it indicates that the calculation is stopped. Finally, the time interval of the first two numerical values output to the environment is taken as the calculation result of the system.

The configuration of the SNP-TIPN systems refers to the charge of each neuron in the systems and the distribution of numerical values within the neuron. Therefore, the initial configuration of the systems can be described by the initial charge, and the number of numerical values contained in each neuron, that is, $C_0 = (\alpha_1, \alpha_2, \dots, \alpha_i; n_1, n_2, \dots, n_i)$. By using firing rules or forgetting rules, the systems will change from one configuration to another. Such a process is called a transition process. Starting from the initial configuration, a series of configuration transitions are called the calculation process of the systems. When the systems transfer from a certain configuration to a configuration with no rules available, it indicates that the calculation is stopped. Finally, the time interval of the first two numerical values output to the environment is taken as the calculation result of the systems.

In the generating mode, $N_{gen}(II)$ is used to represent the number group generated by II , and $N_{gen}SNPTIPN_m^n$ is used to represent all sets of $N_{gen}(II)$ families generated by the SNP-TIPN system, where the system contains at most m neurons, and each neuron at most n rules. When m or n is not restricted, it can be represented by “*”.

Similarly, in the accepting mode, the output neuron is removed, and the input neuron receives the numerical sequence from the environment. The received number n is introduced into a specific neuron with n numerical values. If the calculation stops, it is said that the systems will accept the number n . The number set accepted by II is denoted by $N_{acc}(II)$, and all the sets accepted by the SNP-TIPN systems $N_{acc}(II)$ family are denoted by $N_{acc}SNPTIPN_m^n$.

3.2 An example

In order to clearly illustrate the calculation process of the spiking neural P systems constructed in this article, a simple SNP-TIPN system is constructed below, and its operation process is described in detail.

As shown in Fig. 1, the system contains four neurons $\sigma_{i_1}, \sigma_{i_2}, \sigma_{i_3}$, and σ_{out} . In the initial state, neurons σ_{i_1} and σ_{i_3} are neutrally charged, neuron σ_{i_2} is negatively charged, and neuron σ_{out} is positively charged; neuron σ_{i_1} contains two numerical values, and the number of numerical values in other neurons is empty. Assuming that the initial time is t , at this time neuron σ_{i_1} is excited, according to the principle of maximum numerical value consumption, the first rule $0/a^2 \rightarrow a^2; 0(\{i_2\})$ is executed, sending two numerical values and a neutral charge to neuron σ_{i_2} , and neuron σ_{i_2} is still negatively charged after receiving it. It can be seen from **Fig. 1** that the two rules in neuron σ_{i_2} have the same trigger conditions, and one of them is selected indefinitely. If the first rule $-/a^2 \rightarrow a; 0(\{out\})$ in neuron σ_{i_2} is used, a numerical value and a neutral charge are sent to neuron σ_{out} . At the next moment, rule $+/a \rightarrow a; -(\{emv\})$ in neuron σ_{out} is excited, a numerical value and a negative charge are delivered to the environment, and the calculation stops. If the second rule $-/a^2 \rightarrow a, -(\{out\})$

in neuron σ_{i_2} is used, consume two numerical values and send one numerical value and one negative charge to neuron σ_{out} . At this time, one negative charge meets one positive charge, neuron σ_{out} will become a neutral charge and contain one numerical value. At the next moment, rule $0/a \rightarrow a, 0(\{env, i_3\})$ in neuron σ_{out} is activated, sending a numerical value and a neutral charge to the environment, and at the same time sending it to neuron σ_{i_3} . After neuron σ_{i_3} is received, rule $0/a \rightarrow \lambda; +(\{out\})$ is activated, the numerical value in neuron σ_{i_3} is eliminated, and a positive charge is sent to neuron σ_{out} to restore the charge state of neuron σ_{out} to the initial state.

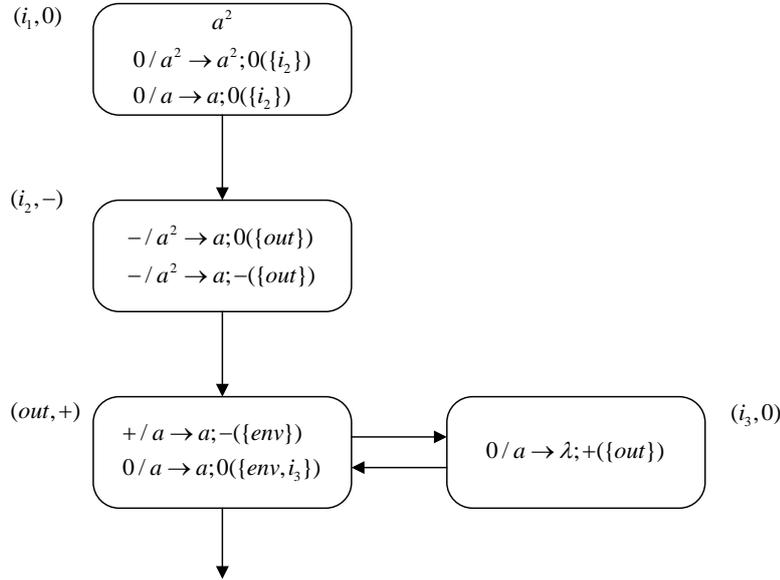


Fig. 1. An example of SNP-TIPN systems.

4 Turing universality of SNP-TIPN systems as number generating/accepting devices

This section demonstrates the universality of the SNP-TIPN systems as number generating and number accepting devices. The Turing machine can generate and accept the set of all recursively enumerable numbers, denoted by NRE. The register machine of the register generation and number acceptance device is also equivalent to the Turing machine, which is equivalent to the register machine,

which can also generate/receive the set NRE of all recursively enumerable numbers. Therefore, we can prove the Turing universality of the SNP-TIPN system by simulating the register machine. We will construct two SNP-TIPN systems, which work in generating mode and accepting mode, respectively, to prove that they can generate and accept all recursively enumerable numbers (NRE) sets.

4.1 The generating mode

Theorem 1. $N_{gen}SNPTIPN_*^2 = NRE$

Proof. To prove theorem 1, only need to prove $NRE \subseteq N_{gen}SNPTIPN_*^2$. This section constructs a SNP-TIPN system Π' , and proves that it is universal in the calculation as a device for generating number sets, that is, it can generate Turing computable natural numbers by simulating the register machine. The register machine M_1 is composed of a 5-tuple $M_1 = (m, H, l_0, l_h, I)$, where m is the number of register machines, H represents the set of instruction tags, l_0 represents the start instruction, l_h represents the stop instruction, I represents a set of instructions. There are three forms of instructions in the register machine:

- (1) ADD instruction (increase the number stored in register r by 1, and transfer to the next instruction);
- (2) SUB instruction (if register r is not empty, make the number stored in register r decrease by 1, and transfer to the next instruction. If the register is empty, then directly transfer to another different instruction);
- (3) HALT instruction (terminate the calculation of the register, and use the number in the register 1 as the result of the calculation).

Special provisions: When the calculation is stopped, all registers except register 1 are empty, and register 1 does not execute subtraction instructions, that is, the number of numerical values in the neuron used to simulate register 1 in the SNP-TIPN system Π' will not decrease.

The system Π' is composed of ADD module, SUB module and FIN module. In each module, when two values are added to the neuron σ_r , it is equivalent to an increase of 1 in the corresponding register r ; in the same way, when two values are reduced in the neuron σ_r , it is equivalent to the corresponding register r minus 1. When the neuron σ_{l_h} in the output module fires, the system stops, and the time interval of the first two values sent by the output neuron σ_{out} to the environment is used as the output result of the system.

- (1) ADD module-simulating instruction $l_i : (ADD(r), l_j, l_k)$.

The ADD module is shown in **Fig. 2**, which is used to simulate the addition instruction $l_i : (ADD(r), l_j, l_k)$. It is equivalent to increasing the value in the register r by 1, and non-deterministically selecting l_j or l_k as the execution instruction at the next moment.

Suppose the system Π' starts to simulate the addition instruction l_i at a certain time t , the neuron σ_{l_i} receives two numerical values from the environment,

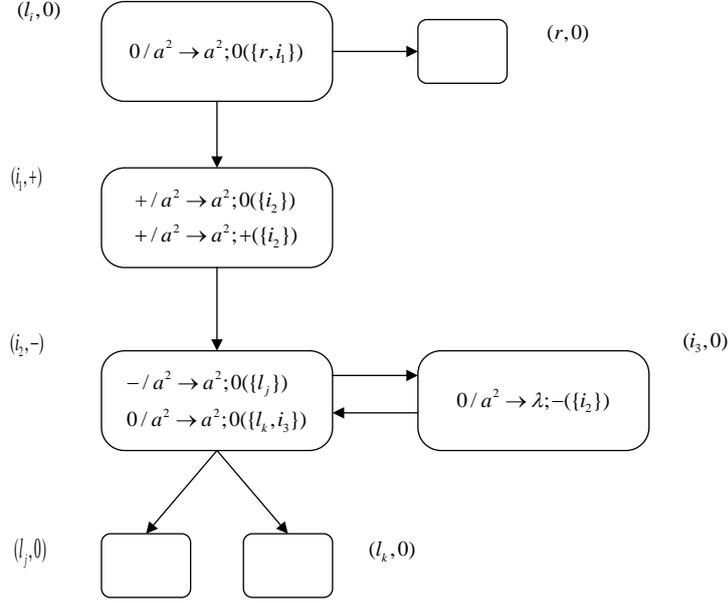


Fig. 2. Module ADD (simulating instruction $l_i : (ADD(r), l_j, l_k)$).

and the rule $0/a^2 \rightarrow a^2; 0(\{r, i_1\})$ in the neuron σ_{l_i} is activated and sent two numerical values and a neutral charge to the neuron σ_r and σ_{i_1} , means that the value of the register r is increased by 1, and at the same time, the neuron σ_{i_1} is still positively charged. At $t + 1$, the two rules $+/a^2 \rightarrow a^2; 0(\{i_2\})$ and $+/a^2 \rightarrow a^2; +(\{i_2\})$ in the neuron σ_{i_1} will select one of them non-deterministically for use. The following two cases are discussed:

- 1) If the rule $+/a^2 \rightarrow a^2; 0(\{i_2\})$ is used, neuron σ_{i_1} sends two numerical values and one neutral charge to neuron σ_{i_2} . At step $t + 2$, there are two numerical values in the negatively charged neuron σ_{i_2} , then the rule $-/a^2 \rightarrow a^2; 0(\{l_j\})$ in neuron σ_{i_2} is excited, and two numerical values enter the neuron σ_{l_j} , Which means that the instruction l_j is executed next.
- 2) If the rule $+/a^2 \rightarrow a^2; +(\{i_2\})$ is used, neuron σ_{i_1} sends two numerical values and one positive charge to neuron σ_{i_2} . After the negatively charged neuron σ_{i_2} receives a positive charge, the charge becomes neutral. At step $t + 2$, the neuron σ_{i_2} uses the rule $0/a^2 \rightarrow a^2; 0(\{l_k, i_3\})$ to make the neuron σ_{l_k} receive two numerical values and one neutral charge, which means that the command l_k will be executed next. At the same time, neuron σ_{i_3} receives two numerical values and one neutral charge, and still maintains a neutral charge. At step $t + 3$, neuron σ_{i_3} uses the forgetting rule $0/a^2 \rightarrow \lambda; -(\{i_2\})$, two numerical values are removed, and a negative charge is sent to neuron σ_{i_2} to restore the charge state of σ_{i_2} to the initial state.

In summary, the ADD module can correctly simulate the addition instruction of the register machine $l_i : (ADD(r), l_j, l_k)$, increase the number of numerical values in the register r by 1, and non – deterministically choose to execute the l_j or l_k instruction.

(2) SUB module-simulating instruction $l_i : (SUB(r), l_j, l_k)$.

The SUB module is shown in **Fig. 3**, which is used to simulate the subtraction instruction of the register machine. Suppose that at time t , the neuron σ_{l_i} receives two numerical values from the environment, and its initial charge is neutral. Neuron σ_{l_i} is excited, applying the rule $0/a^2 \rightarrow a; +(\{r, i_1, i_2\})$, sending a numerical value and a positive charge to neurons σ_r, σ_{i_1} and σ_{i_2} respectively. Therefore, the charge state of the neuron σ_r changes, neuron σ_r is positively charged, and the charge state of neuron σ_{i_2} remains unchanged and still positively charged. At time $t + 1$, neurons σ_{i_1} and σ_{i_2} are activated at the same time, using rules $+/a \rightarrow \lambda; -(\{r\})$ and $+/a \rightarrow a; 0(\{r\})$ respectively. Neuron σ_{i_1} uses the forgetting rule $+/a \rightarrow \lambda; -(\{r\})$ to consume a numerical value, and sends a negative charge to neuron σ_r , turning the charge of neuron σ_r into a neutral charge. At the same time, neuron σ_{i_2} uses rule $+/a \rightarrow a; 0(\{r\})$ to send a numerical value and a neutral charge to neuron σ_r . Assuming that the number of numerical values in neuron σ_r in the initial state is $2n$, at this time neuron σ_r has a neutral charge, and the number of numerical values is $2n + 2$, and neuron σ_r uses one of the rules according to the number of numerical values it contains indefinitely.

- 1) When the number of numerical values in the initial state of neuron σ_r is non-empty $2n(n \geq 1)$, At time $t + 2$, rule $0/a^4 \rightarrow a^2; 0(\{l_j\})$ is applied to neuron σ_r , consumes four numerical values, and sends two numerical values and a neutral charge to neuron σ_{l_j} . Because neuron σ_r received a total of 2 numerical values in the whole process, which is equivalent to consuming two numerical values, the value of neuron σ_r is reduced by one. The neuron σ_{l_j} receives two numerical values and starts to simulate the l_j command.
- 2) When the number of numerical values in the initial state of neuron σ_r is empty $2n(n = 0)$, at $t + 2$, neuron σ_r applies the rule $0/a^2 \rightarrow a^2; 0(\{l_k\})$, consumes two numerical values, and sends two numerical values and a neutral charge to the neuron σ_{l_k} . The neuron σ_{l_k} receives two numerical values, which means that the system has started to simulate the command l_k .

It can be seen that the SUB module starts from the neuron σ_{l_i} , assuming that in the initial state, the neuron σ_{l_i} receives two numerical values from the environment. Then, the rule is executed indefinitely according to the number of numerical values in the neuron σ_r . If the number of numerical values in neuron σ_r in the initial state is not empty, then jump to the l_j instruction; if the number of numerical values in neuron σ_r in the initial state is empty, jump to the l_k instruction. This means that the SUB module correctly simulates the SUB instruction of the register machine.

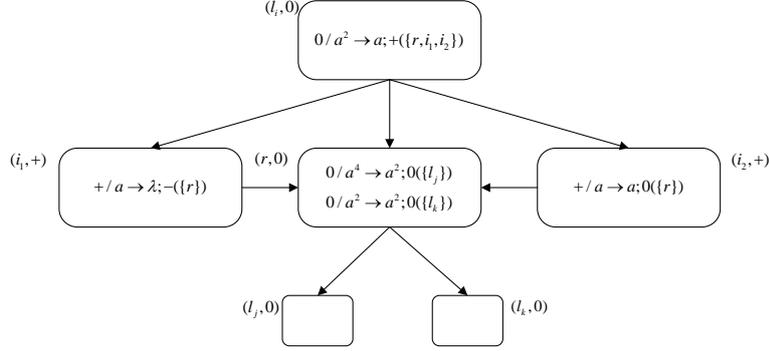


Fig. 3. Module SUB (simulating instruction $l_i : (SUB(r), l_j, l_k)$).

(3) FIN module - outputting the computation result

The FIN module is mainly used to stop the calculation and export calculation results, shown in **Fig. 4**. The calculation result is stored in register 1, and register 1 has nothing to do with the subtraction instruction. Suppose the system Π' at a certain time t , the neuron σ_{l_h} receives two numerical values from the environment, and executes the stop command $l_h = \text{HALT}$. Neuron σ_{l_h} uses firing rule $0/a^2 \rightarrow a; +(\{i_1, i_2, 1\})$ to send a numerical value and a positive charge to neuron σ_{i_1} , σ_{i_2} and neuron σ_1 respectively. Neurons σ_{i_1} and σ_{i_2} are positively charged in the initial state, and the charge state remains unchanged after receiving a positive charge. Neuron σ_1 has a neutral charge in its initial state, and has a positive charge after receiving a positive charge. At time $t + 1$, the rule $+/a \rightarrow a; 0(\{out\})$ in neuron σ_{i_1} is excited, and one numerical value and a neutral charge are sent to neuron σ_{out} , and the charge state of neuron σ_{out} remains unchanged. At this moment, neuron σ_1 contains $2n + 1$ ($n \geq 1$) numerical values, the rule $+/a^3 \rightarrow a; 0(\{i_2\})$ in neuron σ_1 is excited, three numerical values are consumed, and one numerical value and a neutral charge are generated and sent to neuron σ_{i_2} , and the charge state of neuron σ_{i_2} remains unchanged. At the same time, the rule $+/a \rightarrow a; 0(\{1\})$ of neuron σ_{i_2} is also excited, neuron σ_{i_2} sends a numerical value and a neutral charge to neuron σ_1 , and the charge state of neuron σ_1 remains unchanged. At $t + 2$, since the neuron σ_{out} contains one numerical value and has a neutral charge, the rule $0/a \rightarrow a; 0(\{env\})$ is excited, and the first numerical value is sent to the environment. It should be noted that the rules in neuron σ_1 and σ_{i_2} will be used in every time unit, until $t + n$, there is only one numerical value left in neuron σ_1 . At time $t + n + 1$, neuron σ_1 executes the second rule $+/a \rightarrow a; 0(\{out\})$, sending one numerical value and one neutral charge to neuron σ_{out} . At time $t + n + 2$, the neuron σ_{out} sends a second numerical value to the environment by using the rule $0/a \rightarrow a; 0(\{env\})$. Therefore, the time interval for the system to send two numerical values to the

environment is $(t + n + 2) - (t + 2) = n$, and this number is exactly equal to the number stored in register 1 when the register machine M_1 stops.

It can be seen that in the generating mode, the SNP-TIPN system Π' can correctly simulate register machine M_1 . Therefore, Theorem 1 holds.

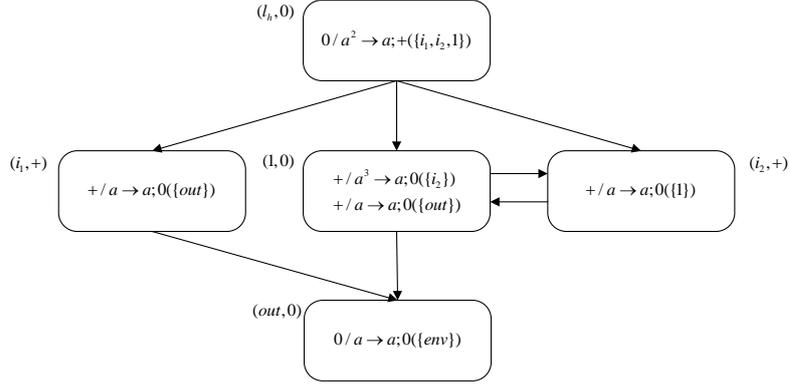


Fig. 4. Module FIN for outputting the result.

4.2 The accepting mode

Theorem 2. $N_{acc}SNPTIPN_*^2 = NRE$

Proof. To prove theorem 2, The SNP system Π is constructed as follows to simulate the register machine $M_2 = (m, H, l_0, l_h, I)$, including the input module, the addition module and the subtraction module. The SNP-TIPN system working in accepting mode adds an input module for receiving values from the environment, and cancels an output module. When the time interval for the input module to receive the first two values from the environment is n , the number n is used as the result of the system calculation. The subtraction module has the same principle as the subtraction module in the generation mode.

The INPUT module is shown in **Fig. 5**. At the initial moment, all neurons in the input module are empty. Suppose that at time t , neuron σ_{in} receives the first numerical value from the environment, neuron σ_{in} is activated, and sends one numerical value and a positive charge to neuron σ_{i_2} and σ_{i_3} respectively. Neurons σ_{i_2} and σ_{i_3} are negatively charged in the initial state, and neutrally charged after receiving a positive charge. At $t + 1$, the rules $0/a \rightarrow a; 0(\{1, i_3\})$, $0/a \rightarrow a; 0(\{1, i_2\})$ in neuron σ_{i_2} and σ_{i_3} are simultaneously excited, neuron σ_1 receive two numerical values; from $t + 1$, neuron σ_{i_2} and σ_{i_3} exchange 1 numerical value in each time unit, neuron σ_1 receives two numerical values in

each time unit, which means that register 1 is in each time unit. The time unit increases by 1. Until $t + n$, neuron i receives the second numerical value from the environment, rule $0/a \rightarrow a; +(\{i_2, i_3\})$ is activated, neuron σ_{in} sends the second numerical value to neurons $\sigma_{i_2}, \sigma_{i_3}$. At $t + n + 1$, neuron σ_{i_2} uses the rule $+/a^2 \rightarrow a^2; 0(\{l_0, i_1, i_4\})$ to send two numerical values and a neutral charge to neurons $\sigma_{l_0}, \sigma_{i_1}, \sigma_{i_4}$, neuron σ_{l_0} receives two numerical values, after that, start to simulate the l_0 instruction of the register machine M . At the same time, neuron σ_{i_3} uses forgetting rule $0/a^2 \rightarrow \lambda; 0(\{1\})$, two numerical values are removed, and a neutral charge is sent to neuron σ_1 . At $t + n + 2$, neurons σ_{i_1} and σ_{i_4} are excited at the same time and send a negative charge to neurons σ_{i_2} and σ_{i_3} respectively, so that the charge states of σ_{i_2} and σ_{i_3} are restored to their initial states. It is worth noting that at time $t + n$, neuron σ_1 still receives two numerical values, and finally neuron σ_1 accumulates $2n$ numerical values, corresponding to the number n stored in register 1.

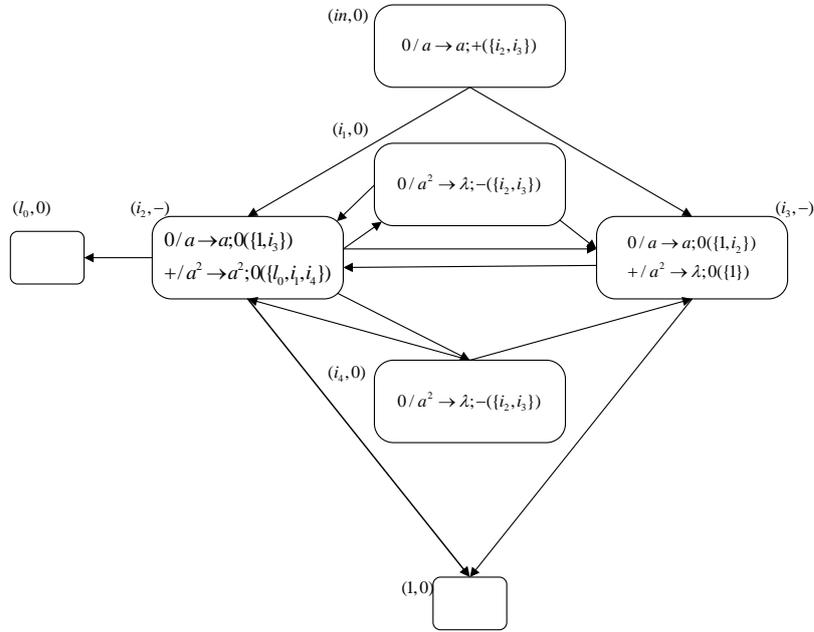


Fig. 5. INPUT module in accepting mode.

Compared with the generating mode, the ADD instruction in the accepting mode is simpler, that is, the deterministic instruction $l_i : (ADD(r), l_j)$, as shown in Fig. 6.

Suppose that at time t , neuro σ_{i_i} receives two numerical values from the environment, rule $0/a^2 \rightarrow a^2; 0(\{r, i_1\})$ in neuron σ_{i_i} is excited, and sends two

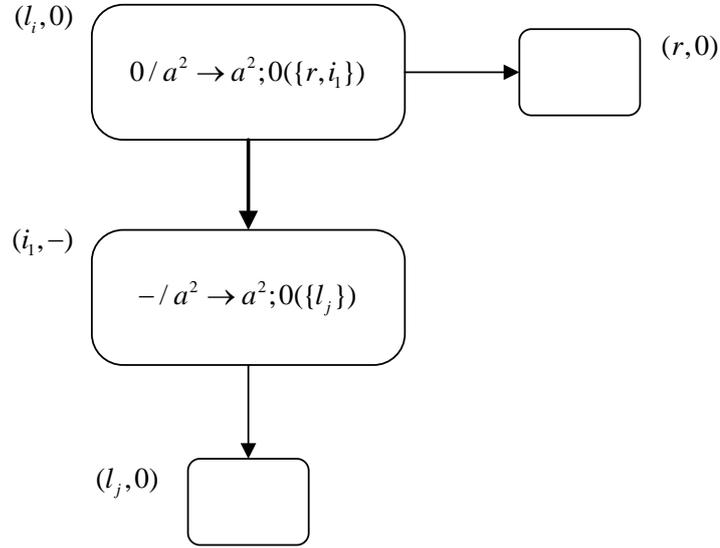


Fig. 6. ADD module in accepting mode.

numerical values and a neutral charge to neurons σ_r and σ_{i_1} . After neuron σ_r receives the value plus one, neuron σ_{i_1} is still negatively charged. At time $t + 1$, rule $-/a^2 \rightarrow a^2; 0(\{l_j\})$ in neuron σ_{i_1} is activated, so that neuron σ_{l_j} receives two numerical values and a neutral charge, which means that the next step is to simulate the l_j command.

The system Π'' in the accepting mode can correctly simulate the addition instruction $l_i : (ADD(r), l_j)$. It is worth noting that the subtraction module of the system Π'' in the receiving mode is the same as the subtraction module in the generation mode. The output module is removed, leaving only one neuron σ_{i_h} without any rules available. When the register machine M_2 executes the command l_h , the simulation register machine M_2 is over, and the system Π'' stops running.

It can be seen that in the accepting mode, the TIPN-SNP system Π'' can correctly simulate register machine M_2 . Therefore, Theorem 2 holds.

5 Final remarks

This paper studies a variant of the spiking neural P systems, that is, the target indicating spiking neural P systems with polarizations and numerical value: the polarization rule is added on the basis of the target indication, avoiding the use of regular expressions. Instead, three types of charges, positive, neutral, and

negative, are introduced to determine the use of firing rules, which making the spiking neural P systems with target indication have more flexible grammar. At the same time, the Turing versatility of the spiking neural P systems is analyzed. The innovations of this article are:

1. Expanding the spiking neural P systems with target indication, analyzing and verifying that the new model of the system is Turing universal as a digital generator and function calculation device.
2. Establishing a target-indicating spiking neural P system with polarizations and numerical value. By extending the application of polarization and numerical values to the spiking neural P systems with the target, a new numerical value grammar mechanism is applied to the numerical value neural membrane system indicated by the target. In this system, in addition to the neuron execution rules, which can send numerical values to other neurons, the execution of the neuron rules also needs to consider the polarity of the charge carried by the neuron. And this system also expands the previous standard numerical value object into a numerical form.

Through the implementation of the above research content, this article will verify and evaluate the computing power of the numerical valued neural membrane system as a digital generator and function computing device based on the establishment of a new model of the target indicating numerical valued neural membrane system analysis, and construct a general-purpose small computing device. The number of neurons needed to explore the prospect and superiority of the spiking neural P systems in the field of computing provides a more practical system for solving complex problems. It is still worth studying to solve the NP problem.

The practical application of the SNP-TIPN system is worth studying, for example, designing controllers for mobile robots or other practical problems that require formal quantitative modeling and numerical calculation techniques; monitoring, flexible manufacturing systems and fault diagnosis; complex time-dependent pattern recognition, information processing and learning, machine learning and natural language processing.

*This work was supported in part by the National Natural Science Foundation of China(No.61806114, 61472231, 61876101, 61602282, 61402187, 61502283, 61802234, 61703251), the China PostdoctoralScience Foundation(No.2018M642695, 2019T120607)

References

1. Aman, B., Ciobanu, G.: Modelling and verification of weighted spiking neural systems. *Theoretical Computer Science* **623**, 92–102 (2016)
2. Aman, B., Ciobanu, G.: Synchronization of rules in membrane computing. *Journal of Membrane Computing* **1**(4), 233–240 (2019)
3. Cabarle, F.G.C., de la Cruz, R.T.A., Cailipan, D.P.P., Zhang, D., Liu, X., Zeng, X.: On solutions and representations of spiking neural p systems with rules on synapses. *Information Sciences* **501**, 30–49 (2019)
4. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulating a p system based efficient solution to sat by using gpus. *The Journal of Logic and Algebraic Programming* **79**(6), 317–325 (2010)
5. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Peng, H.: Membrane computing and image processing: A short survey. *Journal of Membrane Computing* **1**(1), 58–73 (2019)
6. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Peng, H.: Membrane computing and image processing: A short survey. *Journal of Membrane Computing* **1**(1), 58–73 (2019)
7. Frias, T., Sanchez, G., Garcia, L., Abarca, M., Diaz, C., Sanchez, G., Perez, H.: A new scalable parallel adder based on spiking neural p systems, dendritic behavior, rules on the synapses and astrocyte-like control to compute multiple signed numbers. *Neurocomputing* **319**, 176–187 (2018)
8. Garcia, L., Sanchez, G., Vazquez, E., Avalos, G., Anides, E., Nakano, M., Sanchez, G., Perez, H.: Small universal spiking neural p systems with dendritic/axonal delays and dendritic trunk/feedback. *Neural Networks* **138**, 126–139 (2021)
9. George, F.: On the computational power of spiking neural p systems with self-organization. *Scientific Letter* **6**(1), 1–16 (2016)
10. Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S., Lefticaru, R.: Kernel p systems: from modelling to verification and testing. *Theoretical Computer Science* **724**, 45–60 (2018)
11. Hinze, T., Happe, H., Henderson, A., Nicolescu, R.: Membrane computing with water. *Journal of Membrane Computing* **2**(3), 121–136 (2020)
12. Jiang, K., Pan, L.: Spiking neural p systems with anti-spikes working in sequential mode induced by maximum spike number. *Neurocomputing* **171**, 1674–1683 (2016)
13. Jiang, S., Fan, J., Ling, D., Yang, F., Wang, Y., Wu, T.: Cell-like spiking neural p systems with anti-spikes and membrane division/dissolution. In: *International Conference on Bio-Inspired Computing: Theories and Applications*. pp. 393–408. Springer (2019)
14. Kumar, D., Verma, N.K., Singh, N.: A review paper on deducting database in membrane computing. *Journal of Statistics and Management Systems* **21**(4), 667–673 (2018)
15. Liu, X., Li, Z., Liu, J., Liu, L., Zeng, X.: Implementation of arithmetic operations with time-free spiking neural p systems. *IEEE transactions on nanobioscience* **14**(6), 617–624 (2015)
16. Liu, X., Li, Z., Suo, J., Liu, J., Min, X.: A uniform solution to integer factorization using time-free spiking neural p system. *Neural Computing and Applications* **26**(5), 1241–1247 (2015)
17. Lv, Z., Bao, T., Zhou, N., Peng, H., Huang, X., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Spiking neural p systems with extended channel rules. *International Journal of Neural Systems* **31**(01), 2050049 (2021)

18. Pan, L., Orellana-Martín, D., Song, B., Pérez-Jiménez, M.J.: Cell-like p systems with polarizations and minimal rules. *Theoretical Computer Science* **816**, 1–18 (2020)
19. Pan, L., Păun, G., Pérez-Jiménez, M.J.: Spiking neural p systems with neuron division and budding. *Science China Information Sciences* **54**(8), 1596–1607 (2011)
20. Pan, L., Wu, T., Su, Y., Vasilakos, A.V.: Cell-like spiking neural p systems with request rules. *IEEE Transactions on Nanobioscience* **16**(6), 513–522 (2017)
21. Pan, L., Zeng, X., Zhang, X., Jiang, Y.: Spiking neural p systems with weighted synapses. *Neural Processing Letters* **35**(1), 13–27 (2012)
22. Pan, T., Shi, X., Zhang, Z., Xu, F.: A small universal spiking neural p system with communication on request. *Neurocomputing* **275**, 1622–1628 (2018)
23. Păun, A., Păun, G.: Small universal spiking neural p systems. *BioSystems* **90**(1), 48–60 (2007)
24. Peng, H., Li, B., Wang, J., Song, X., Wang, T., Valencia-Cabrera, L., Pérez-Hurtado, I., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Spiking neural p systems with inhibitory rules. *Knowledge-Based Systems* **188**, 105064 (2020)
25. Peng, H., Lv, Z., Li, B., Luo, X., Wang, J., Song, X., Wang, T., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Nonlinear spiking neural p systems. *International Journal of Neural Systems* **30**(10), 2050008 (2020)
26. Peng, H., Wang, J.: Coupled neural p systems. *IEEE Transactions on Neural Networks and Learning Systems* **30**(6), 1672–1682 (2018)
27. Peng, H., Wang, J., Pérez-Jiménez, M.J., Wang, H., Shao, J., Wang, T.: Fuzzy reasoning spiking neural p system for fault diagnosis. *Information Sciences* **235**, 106–116 (2013)
28. Ren, Q., Liu, X.: Delayed spiking neural p systems with scheduled rules. *Complexity* **2021**, 1–13 (2021)
29. Singh, G., Deep, K.: A new membrane algorithm using the rules of particle swarm optimization incorporated within the framework of cell-like p-systems to solve sudoku. *Applied Soft Computing* **45**, 27–39 (2016)
30. Song, B., Li, K., Orellana-Martín, D., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: Cell-like p systems with evolutionary symport/antiport rules and membrane creation. *Information and Computation* **275**, 104542 (2020)
31. Song, B., Zeng, X., Rodríguez-Patón, A.: Monodirectional tissue p systems with channel states. *Information Sciences* **546**, 206–219 (2021)
32. Song, T., Pan, L.: Spiking neural p systems with request rules. *Neurocomputing* **193**, 193–200 (2016)
33. Song, X., Peng, H., Wang, J., Ning, G., Sun, Z.: Small universal asynchronous spiking neural p systems with multiple channels. *Neurocomputing* **378**, 1–8 (2020)
34. Song, X., Valencia-Cabrera, L., Peng, H., Wang, J.: Spiking neural p systems with autapses. *Information Sciences* **570**, 383–402 (2021)
35. Sun, M., Qu, J.: Weighted spiking neural p systems with structural plasticity working in maximum spiking strategy (2016)
36. Wang, T., Wei, X., Wang, J., Huang, T., Peng, H., Song, X., Cabrera, L.V., Pérez-Jiménez, M.J.: A weighted corrective fuzzy reasoning spiking neural p system for fault diagnosis in power systems with variable topologies. *Engineering Applications of Artificial Intelligence* **92**, 103680 (2020)
37. Wu, T., Pan, L.: The computation power of spiking neural p systems with polarizations adopting sequential mode induced by minimum spike number. *Neurocomputing* **401**, 392–404 (2020)
38. Wu, T., Pan, L., Alhazov, A.: Computation power of asynchronous spiking neural p systems with polarizations. *Theoretical Computer Science* **777**, 474–489 (2019)

39. Wu, T., Pan, L., Yu, Q., Tan, K.C.: Numerical spiking neural p systems. *IEEE Transactions on Neural Networks and Learning Systems* **32**(6), 2443–2457 (2020)
40. Wu, T., Păun, A., Zhang, Z., Pan, L.: Spiking neural p systems with polarizations. *IEEE Transactions on Neural Networks and Learning Systems* **29**(8), 3349–3360 (2018)
41. Wu, T., Zhang, L., Pan, L.: Spiking neural p systems with target indications. *Theoretical Computer Science* **862**, 250–261 (2021)
42. Yang, Q., Li, B., Huang, Y., Peng, H., Wang, J.: Spiking neural p systems with structural plasticity and anti-spikes. *Theoretical Computer Science* **801**, 143–156 (2020)
43. Yang, Q., Lv, Z., Liu, L., Peng, H., Song, X., Wang, J.: Spiking neural p systems with multiple channels and polarizations. *Biosystems* **185**, 104020 (2019)
44. Zhang, G., Samdanielthompson, G., David, N.G., Nagar, A.K., Subramanian, K.: A bio-inspired model of picture array generating p system with restricted insertion rules. *Applied Sciences* **10**(22), 8306 (2020)
45. Zhao, Y., Liu, X., Wang, W.: Spiking neural p systems with neuron division and dissolution. *Plos One* **11**(9), e0162882 (2016)

An Improved Deep Echo State Network Inspired by Tissue-Like P System Forecasting For Non-stationary Time Series

Xiaojian Yang^{1,2}, Qian Liu^{1,2}, Xiyu Liu^{1,2}

¹Business school of Shandong Normal University, Jinan, China

²Academy of Management Science, Shandong Normal University, Jinan, China
xyliu@sdsu.edu.cn

Abstract. As a recurrent neural network, ESN has attracted wide attention because of its simple training process and unique reservoir structure, and has been applied to time series prediction and other fields. However, ESN also has some shortcomings, such as the optimization of reservoir and collinearity. Many researchers try to optimize the structure and performance of deep ESN by constructing deep ESN. However, with the increase of the number of network layers, the problem of low computing efficiency also follows. In this paper, we combined membrane computing and neural network to build an improved deep echo state network inspired by tissue-like P system. Through analysis and comparison with other classical models, we found that the model proposed in this paper has achieved great success both in predicting accuracy and operation efficiency.

Keywords: Deep Echo State Network; Membrane Computing; Time Series Prediction

1 Introduction

With the development of modern information technology and the rapid growth of data, the value of data is getting more and more attention. Decision-making in various fields depends on data processing and analysis. A large number of time series data also exist in various fields of life, such as the financial field, the transportation field, and the astronomical field. Time series analysis is the process of processing dynamic data. It needs to be analyzed on the basis of existing data to obtain the useful information contained in the data and realize the extraction of value [1,2]. For data with time series, there are usually difficulties such as large amount of data, high complexity, high storage cost and low calculation efficiency, so it is of great significance for the mining and analysis of time series data [3].

Recurrent Neural Network (RNN) is a traditional method for time series prediction [4,5]. This model is a network with a cyclic structure. Its working principle is to transport the information obtained from the network layer at the previous time to the network layer at the next time. The output of the hidden layer is determined by the sequence information of the past time. With the research on complex time series data, cyclic neural networks have also become a type of network specially used for processing time series data, but this type of network cannot handle long series data well. Because in the training process, the calculated gradient cannot be transmitted for a long time, which will cause the phenomenon of gradient disappearance and gradient explosion [6,7]. Continuous and in-depth development on the basis of deep learning can effectively solve some of the problems in time series data [8], but it will cause many parameters in the training process. The selection of these parameters usually requires the researcher to have good experience, which will directly affect the performance of the model.

In 1998, Academician Gheorghe Păun proposed membrane computing, a computing method inspired by nature [9,10]. Its development starts from the observation of cells, and this new computing model is built by abstracting from the structure and function of cells, also known as the P system. tissue-like P system is the abstract of living organisms, so contains a number of cells, cells have connected channels through which they can communicate directly, and cells without connecting channels can communicate indirectly through the environment. So tissue-like P systems are easier to implement information exchange [11,12].

Therefore, our motivation is to design a deep neural network based on tissue-like P systems, while achieving better performance than traditional deep neural networks. In this paper, we focus on the NARMA signal, as these are the most common time series in correlational research, yet verify our model performance.

The rest of this paper is organized as follows. Section 2 introduces the related work. . Section 3 describes an overview of the background works and proposes a deep echo state network (DESN) based on tissue-like P systems. The prediction performance of the model and the analysis of experimental results are presented in Section 4. Finally, some conclusions are given in Section 5.

2 Related Work

2.1 Echo State Network

As shown in fig.1, ESN is the structural basis of deep ESN. They have exactly the same reservoir structure and the same training mechanism. Echo State Network (ESN) is one of RC (Reservoir Computing) algorithms, which is developed on the basis of recurrent neural network. In order to solve the problems of large training resource consumption and long running time of cyclic neural network, ESN came into being [13,14].

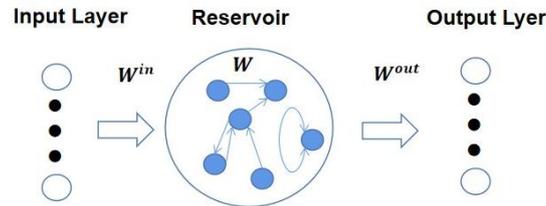


Fig. 1. The structure of the Echo State Network

The echo state network was proposed in 2001. It is a special type of RNN, and it is also composed of input layer, hidden layer and output layer. The difference between ESN and traditional neural networks is that it adds a randomly connected reserve pool to replace the original hidden layer. The connection state of neurons in the reserve pool is random, and the connection weight is fixed. This allows it to effectively reduce the amount of calculation during the training process, and to a certain extent avoid the phenomenon of local minima during the gradient descent process [15]. The reservoir accepts two directions of input, one from the input layer, and the other from the output of the previous state of the reservoir, where the state feedback weight is the same without training, and it is determined by the random initial state [16,17]. ESN uses randomly connected neurons in the reserve pool to generate a complex state space. The input data on the left is linearly combined with the state space to obtain the output data on the right.

Because the structure of the echo state network enriches the theory of traditional neural networks, and the parameter learning is simpler and faster, it has become an effective tool for studying time series data. Coulibaly et al. used the echo state network to predict the monthly average water level of four lakes in the United States as an example. In the hydrological time series prediction problem, it proved that the echo state network method is superior to the traditional recurrent neural network [18]. Regarding the incoming traffic load of the mobile network, Bianchi et al. used the echo state network to predict and obtained a better prediction effect [19]. Through the improvement of the echo state network, Liu et al. applied it to the production process of iron and steel enterprises, and they were also satisfied with the prediction results of the amount of blast furnace coal. A hybrid echo state network with complex network characteristics was proposed by Cui et al. The complex network theory is introduced on the basis of the traditional echo state network, which effectively improves the accuracy of time series prediction [20]. Najibi et al. proposed three new types of echo state networks. These three networks use K-means, PAM and Ward algorithms to construct the structure of the reserve pool. The prediction effect on the chaotic time series is significantly better than that of the traditional network [21].

4

2.2 tissue-like P Systems

P systems are distributed computational parallel models, inspired by the structure and functions of cells, tissues and organs. The P system contains communication rules to realize various functions. Communication between cells is realized by the exchange between objects. The execution of rules within cells meets the maximum parallelism, and each cell can operate independently, which makes P system with the maximum parallelism[22,23,24]. The original tissue-like P system was defined as:

$$\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, \text{syn}, i_0)$$

Where O represents finite non-empty alphabets of objects; $\text{syn} \subseteq \{1,2,\dots,m\} * \{1,2,\dots,m\}$, syn represents the communication channel between cell i and cell j ; $i_0 \in \{1,2,\dots,m\}$ represents that cell i is the output cell of the system, used to output the results of the computation; $\sigma_1, \sigma_2, \dots, \sigma_m$ is cells, m is the number of cells, and its specific form is defined as

$$\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i), 1 < i < m$$

Where Q_i is a finite set of states; $s_{i,0} \in Q_i$ represents the initial state of the cell; $w_{i,0} \in O$ represents the multiple sets of objects contained in cell i in the initial state; R_i is the set of rules inside the cell.

When the system implements rule P_i , w in cell i will be transformed and communicated under state s , and w will be transformed into x, y and z , and x will be left in cell i , y will be sent to the cell that has communication channel with cell i , and z will be transported to the environment. After completing this series of operations, the state of cell i will be transformed from s to s' . The transformation of the state means the completion of the computation. The tissue-like P system finishes the complete computation process through the continuous transformation of the state.

2.3 Time Series Prediction

Time series prediction models have evolved from early linear models, such as autoregressive moving average models, to better nonlinear models, such as neural network models. The development from linear regression modeling method with clear mathematical relations to black box nonlinear modeling method requires researchers to be able to adopt more effective theoretical methods, such as machine learning, fuzzy reasoning, heuristic, neural network and other artificial intelligence methods. There are obvious differences between various linear and nonlinear modeling methods. On the one hand, the difference lies in the different mathematical methods adopted by each prediction method. On the other hand, the difference lies in the significant differences in the theoretical methods used by different prediction methods. In the implementation of the algorithm, the computational resources required by different prediction methods are also very different. Different prediction methods have different mechanisms for extracting data features. The methods for extracting data features can be divided into two forms: explicit and implicit. Explicit feature extraction method is to build features by directly transforming finite length historical data at each time step. In contrast, the implicit feature extraction method is

to construct the internal dynamic features contained in the historical data through machine learning method. This method does not need to strictly consider the time relationship of the data in the time series. Different forecasting methods can be distinguished by forecasting model data characteristics and training methods. A variety of common time series forecasting methods will be introduced in the following sections.

3 Methods

3.1 Framework of Deep Echo State Network

Hinton et al. proposed an unsupervised greedy layer-wise training algorithm, opening the door of deep learning research. Deep ESN is the result of ESN combined with deep learning thought. Its essence is multi-layer artificial neural network, with simple input layer and output layer, and multiple reservoir structures as hidden layer. DESN has more reservoir structures and can map more complex time-series applications. Depth the echo state network (DESN) structure as shown in figure 2, the left is the input layer, its function is to the external data into the depth of the echo state network, the bottom is output layer, its function is the depth of the echo state network output the generated data to the external, is among multiple hidden layer, each layer is a dynamic structure of reservoir.

The DESN works like this: the input layer loads external data into the DESN, $u(n) \in \mathbb{R}^{K \times 1}$, K is the dimension of the input data, which determines the number of neurons in the input layer. External data enter DESN and enter the first reservoir after weighted by input weight $W_{in}^{(1)} \in \mathbb{R}^{N_1 \times K}$, where N_1 represents the number of neurons in the first reservoir. In the reservoir neurons of the first layer, the state value of the previous historical time point $x^{(1)}(n-1)$ inside the reservoir is weighted by reservoir weight $W^{(1)} \in \mathbb{R}^{N_1 \times N_1}$. After summing with the received weighted input, a new state value is formed from the activation function of the neuron, thus updating the state value of the first reservoir, denoting as $x^{(1)}(n) \in \mathbb{R}^{N_1 \times N_2}$. Like the first reservoir layer, the status of the second reservoir is updated from $x^{(2)}(n-1)$ to $x^{(2)}(n) \in \mathbb{R}^{N_2 \times 1}$. The model repeats the working process of the reservoir until the state value of the neurons in the last reservoir is updated. The state value of the last reservoir is denoted as $x^{(L)}(n) \in \mathbb{R}^{N_L \times 1}$, where L represents the maximum number of layers, also known as the depth. At this point, all the status value of reservoir neurons has updated, $N = N_1 + N_2 + \dots + N_L$ is total number of the neurons of DESN reservoirs. All state values are sorted together and denoted as $x(n) = [(x^{(1)}(n))^T, (x^{(2)}(n))^T, \dots, (x^{(L)}(n))^T]^T \in \mathbb{R}^{N \times 1}$, all of the status value weighted by the output weights $W^{out} \in \mathbb{R}^{M \times N}$ is feed to the output layer. The output layer outputs the final result to the network. M represents the dimension of output data, and is also the number of neurons in the output layer. So far, the flow of data in the network is finished.

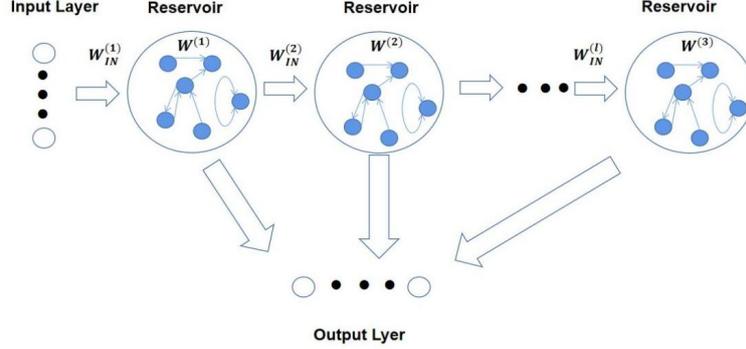


Fig. 2. The structure of the Deep Echo State Network

3.2 The tissue-like P System Based on Deep Echo State Networks

The membrane structure inside the class organization forms a network structure, and the system objects change their positions and states through communication rules and evolution rules. This paper applies the internal organization to the sonic state network, transfers the objects between the outer membranes through communication rules, and uses evolution rules to complete the state changes of the objects.

We constructed a tissue P system with 13 cells as shown in the Fig. 3, and its formal definition is:

$$\Pi = (O, \sigma_1, \dots, \sigma_{13}, ch, r_i, i_0)$$

where:

- (1) $O = \{x_1, x_2, \dots, x_n\}$ is a set of objects, where $x_j (j = 1, 2, \dots, n)$ is the j -th vector;
- (2) $\sigma_i (i = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$ represents the cell, its form is as follows: $\sigma_i = (Q_i, s_{i,0}^t, \omega_{i,0}, R_i)$, $1 \leq i \leq 13$, $Q_i = (s_{i,1}^t, s_{i,2}^t, \dots, s_{i,13}^t)$, where $t = (0, 1, \dots, t_{\max})$, $s_{i,0}^t (i = 1, 2, \dots, 13)$ represents the state of the i -th object in membrane i at time t , $\omega_{i,0} \in O^*$ represents the multiset of initial objects, and R_i is a finite set of rules, which represents the evolutionary rules in cell i . It will change the state of objects in the cell.
- (3) $ch = \{(1,2), (2,3), (3,4), (4,5), (5,6), (6,7), (7,8), (8,9), (9,10), (10,11), (2,12), (3,12), (4,12), (5,12), (6,12), (7,12), (8,12), (9,12), (10,12), (11,12), (12,13)\}$ represents the connecting channel between different cells. For example, cell 2 can only receive information from cell 1, while cell 12 can receive information from cells 2 to 11;
- (5) $r_i = (i, u/j, \lambda)$ is a communication rule. This rule indicates that the string u can be transmitted from the cell i to the cell j , and the cell j cannot transmit the cell i . Because λ is an empty string. In this system, all communication rules are one-way transmission rules.
- (6) $i_0 = 13$ indicates that cell 13 is the output cell of the entire system.

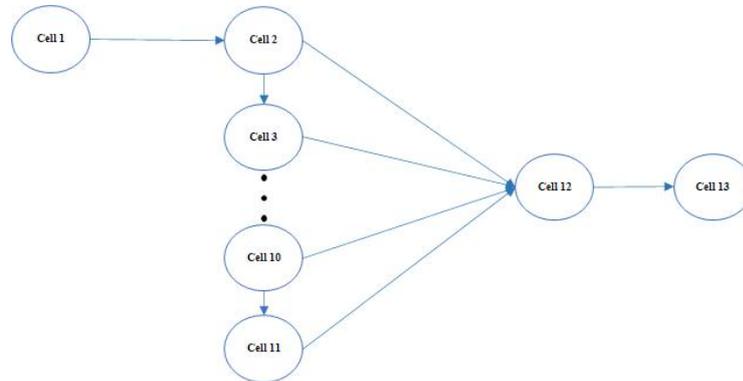


Fig. 3. The structure of the tissue-like P system

3.3 Operating Mechanism

Based on the organization P system, the objects of the original echo state network will change within the membrane structure. The rules in each membrane will be executed independently and will not affect each other, which will greatly improve the calculation efficiency.

(1) initialization

In order to perform the task of forecasting time series data, the system generates all initial objects in the input cell (cell 1), and each object represents a one-dimensional or multi-dimensional vector. The dimension of the time series data in this article is $w \times 1$, and then the objects are normalized in the input cell, mapped in the range of $[0-1]$, and these objects are transmitted to cell 2 through communication rules.

(2) computer system

The principle of extreme parallelism is adopted when the rules in the P system are executed, that is, all the rules that meet the conditions are executed in the system. The communication between the membrane and the membrane is carried out according to the communication rules between the membranes so that the objects in the membrane can move their positions according to this rule. In the beginning, the cell 2 receives the objects delivered by the cell, evolves within the cell 2 according to the evolutionary rules, and outputs the result to the cell 3 and the cell 12 at the same time. On the basis of satisfying the conditions, the membrane of each object will evolve according to the evolutionary rules in the membrane, and finally deliver the result to the next cell. The objects in the cell 2 to the cell 11 will all be transported to the cell 12, and eventually they will evolve according to the rules in the cell 12, and the evolved objects will be transported to the cell 13.

(3) termination condition and output

After the objects in the former cell 12 complete all the evolutionary rules, the cell 12 delivers the final result to the output cell (cell 13), the calculation process stops, and the membrane terminates the evolution. Finally, all objects in the output film are considered the final result.

4 Numerical Experiments and Results Analysis.

4.1 Comparative Methods

To evaluate the performance of the tissue-like p system based on deep echo state Networks, we compare some well-known RNN prediction models in deep learning: convolutional neural network (CNN), long short-term memory (LSTM), traditional ESN. In the past studies, these models have been achieved the great success and widely applied in time series prediction tasks.

4.2 Experimental Data And Experimental Environment

This paper selects the benchmark task commonly used in time series prediction, the NARMA signal. The Non-linear auto-regressive (NARMA) data set were originally proposed by Jaeger and included modeling of the following R-order system outputs:

$$y(n+1) = 0.3y(n) + 0.55y(n) \left[\sum_{i=0}^r y(n-i) \right] + 1.5x(n-r)x(n) + 0.1$$

The input $x(n)$ of the system is the noise randomly distributed between $[0, 1]$. The NARMA task requires memory of at least r past time steps, since the output of the system $y(n+1)$ is determined by the input and the outputs of the r time steps.

In order to avoid the influence of value range on the model, the input was pre-processed. The original time series data were linearly transformed before entering the data to the interval $[0,1]$. The normalization formula is

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

In this paper, the tensorflow open source platform is used as the deep learning platform, and Python 3.7 is used to write the experimental program. Meanwhile, some third-party libraries are used, such as Talib to calculate technical indicators and Keras to build the network structure. The experimental operating system is Windows 10.

4.3 Performance Assessment

This paper evaluates all prediction models in terms of model accuracy. For model accuracy, we choose root mean square error (RMSE) as the measurement standard. The calculation formula of ERMSE can be expressed as

$$RMSE = \sqrt{\sum_{t=1}^T (y_t - f_t)^2 / T}$$

where y_t and f_t are respectively the observed value and output value of the model at time t , and T is the number of data points. In this paper, the fitting and prediction

accuracy of the model is quantitatively evaluated by calculating RMSE values of the training set and the test set respectively. In this article, we do not expand the RMSE value of the training set.

4.4 Results

We expect to explore the performance of the proposed model for time series forecasting in the NARMA signal data set. As shown in Figure 4-7 and Table 1, for NARMA signal, the ESN with a single reservoir architecture achieves the same performance as LSTM. The RMSE of the ESN is slightly higher than that of the LSTM. As we increase the number of the reservoir, the tissue-like p system based on deep echo state networks showed significant improvements in predictive performance. The proposed model achieves much better RMSE than other models. It is worth noting that the complexity of NARMA time series is relatively low. In the above experiment, we prove that the model we proposed is more effective than other RNN in time series prediction. The proposed model not only has a great improvement in the prediction accuracy, but also has a certain advantage in the computation time compared with the traditional DESN.

Table 1. the prediction performance of models for NARMA signal .

Models	RMSE
LSTM	0.0219
Traditional ESN	0.0192
CNN	0.301
Proposed model	1.01e-05

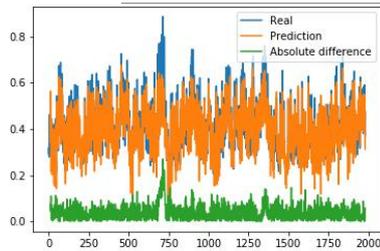


Fig. 4. The prediction result of CNN

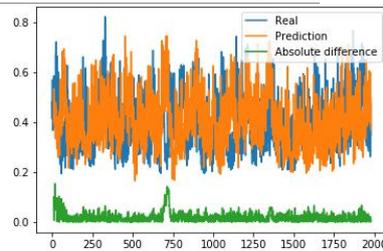


Fig. 5. The prediction result of the LSTM

10

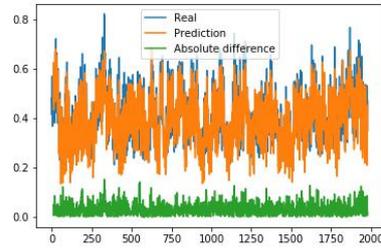
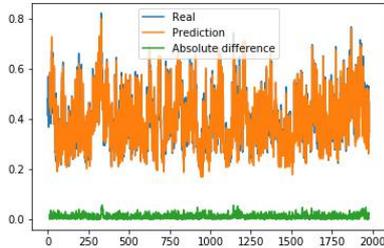


Fig. 6. The prediction result of ESN Fig. 7. The prediction result of the proposed model

5 Conclusions

In the field of time series forecasting, we expect to combined membrane computing and neural network to build a more computationally efficient model that can directly process time series. Although we have only made some preliminary improvements, some useful conclusions that are helpful to the future research have been given in this paper.

In this paper, the tissue-like p system based on deep echo state networks model was presented for modeling time series. The combination of ESN and deep learning improves the prediction ability of the model for complex time series. The application of the tissue-like p system framework overcomes the inherently low efficiency of deep neural network. Therefore, the model we proposed has some advantages.

This article focuses on use the tissue-like P system to implement the operation process of the neural network. However, P systems also have many applications in finding optimal solutions. We expect to find the application of P system in the parameter optimization of neural networks.

References

1. Cui Z , Chen W , Chen Y . Multi-Scale Convolutional Neural Networks for Time Series Classification[J]. 2016.
2. Omer Berat Sezer,Mehmet Ugur Gudelek,Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning : A systematic literature review: 2005–2019[J]. Applied Soft Computing Journal,2020,90.
3. V Olsavszky, Dosius M , Vladescu C , et al. Time Series Analysis and Forecasting with Automated Machine Learning on a National ICD-10 Database[J]. International Journal of Environmental Research and Public Health, 2020, 17(14):4979.
4. Jiaojiao Hu, Xiaofeng Wang, Ying Zhang, Depeng Zhang, Meng Zhang, Jianru Xue. Time Series Prediction Method Based on Variant LSTM Recurrent Neural Network[J]. Neural Processing Letters,2020,52(2).
5. Zhang, Huaguang, Wang, et al. A Comprehensive Review of Stability Analysis of Continuous-Time Recurrent Neural Networks.[J]. IEEE Transactions on Neural Networks & Learning Systems, 2014.

6. Weerakody Philip B., Wong Kok Wai, Wang Guanjin, Ela Wendell. A review of irregular time series data handling with gated recurrent neural networks[J]. *Neurocomputing*, 2021, 441.
7. Yara Rizk, Mariette Awad. On Extreme Learning Machines in Sequential and Time Series Prediction: A Non-Iterative and Approximate Training Algorithm for Recurrent Neural Networks[J]. *Neurocomputing*, 2018, 325.
8. Omer Berat Sezer, Mehmet Ugur Gudelek, Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning : A systematic literature review: 2005–2019[J]. *Applied Soft Computing Journal*, 2020, 90.
9. Bosheng Song, Kenli Li, David Orellana-Martín, Mario J. Pérez-Jiménez, Ignacio Pérez-Hurtado, A survey of nature-inspired computing: membrane computing, *ACM Computing Surveys*, 2021, 54(1), 1-31.
10. Bosheng Song, Kenli Li, David Orellana-Martín, Luis Valencia-Cabrera, Mario J. Pérez-Jiménez, Cell-like P systems with evolutionary symport/antiport rules and membrane creation, *Information and Computation*, 2020, 275, 104542.
11. Linqiang Pan, Mario J. Pérez-Jiménez. Computational complexity of tissue-like P systems[J]. *Journal of Complexity*, 2010, 26(3).
12. Bosheng Song, Xiangxiang Zeng, Min Jiang, Mario J. Pérez-Jiménez, Monodirectional tissue P systems with promoters, *IEEE Transactions on Cybernetics*, 2021, 51(1), 438-450.
13. Lukosevicius M , J Ae Ger H . Reservoir computing approaches to recurrent neural network training[J]. *Computer Science Review*, 2009, 3(3):127-149.
14. Jaeger H . The "Echo State" Approach to Analysing and Training Recurrent Neural Networks[J]. *überwachtes lernen*, 2001.
15. Bala A , Ismail I , Ibrahim R , et al. Applications of Metaheuristics in Reservoir Computing Techniques: A Review[J]. *IEEE Access*, 2018, 6:58012-58029.
16. Bo Y C , Wang P , Zhang X . An asynchronously deep reservoir computing for predicting chaotic time series[J]. *Applied Soft Computing*, 2020, 95:106530.
17. Lun S X , Yao X S , Qi H Y , et al. A novel model of leaky integrator echo state network for time-series prediction[J]. *Neurocomputing*, 2015, 159(jul.2):58-66.
18. COULIBALY P. Reservoir Computing Approach to Great Lakes Water Level Forecasting [J]. *Journal Of Hydrology*, 2010, 381 (1-2): 76-88.
19. BIANCHI F M, SCARDAPANE S, UNCINI A, et al. Prediction of Telephone Calls Load Using Echo State Network with Exogenous Variables [J]. *Neural Networks*, 2015, 71 204-213.
20. CUI H, LIU X, LI L. The Architecture of Dynamic Reservoir in the Echo State Network [J]. *Chaos*, 2012, 22 (3): 455.
21. NAJIBI E, ROSTAMI H. Scesn, Spesn, Swesn: Three Recurrent Neural Echo State Networks with Clustered Reservoirs for Prediction of Nonlinear and Chaotic Time Series [J]. *Applied Intelligence*, 2015, 43 (2): 460-472.
22. Yu W , Wang J , Peng H , et al. Fault Diagnosis of Power Systems Using Fuzzy Reasoning Spiking Neural P Systems with Interval-valued Fuzzy Numbers[J]. *Romanian journal of information science and technology*, 2017, 20(1):5-17.
23. Liu X , Zhao Y , Sun M . An Improved Apriori Algorithm Based on an Evolution-Communication tissue-like P System with Promoters and Inhibitors[J]. *Discrete Dynamics in Nature and Society*, 2017, (2017-02-19), 2017, 2017:1-11.
24. Bosheng Song, Kenli Li, Xiangxiang Zeng, Monodirectional evolutionary symport tissue P systems with promoters and cell division, *IEEE Transactions on Parallel and Distributed Systems*, 2021, DOI: 10.1109/TPDS.2021.3065397.

Fuzzy Tissue-like P Systems with Promoters and Their Application in Power Coordinated Control of Microgrid

Wenping Yu¹, Jieping Wu^{1,2} *, Yubo Wu¹, and Yufeng Chen¹

¹ School of Automation and Electrical Engineering, Chengdu Technological University, Chengdu 611730, P. R. China

² College of Electronics and Information Engineering, Sichuan University, Chengdu 610065, P. R. China

densitywellldone@hotmail.com

Abstract. A fuzzy tissue-like P system with promoters (PFTPS) for the power coordinated control of microgrid is proposed in this paper. Firstly, the promoters and a common language fuzzy system are integrated into the tissue-like P systems, so that large amount of uncertain and inaccurate information contained in the microgrid can be better characterized, and its detailed definition is given. Then, microgrid is one of the important means for effective utilization of renewable energy, which is of great significance to realize its stable operation. The power coordinated control of the microgrid aims to realize the stable and reliable operation of the internal microgrid and maintain the reasonable and effective power interaction between the microgrid and the external grid. So, the proposed PFTPS is applied to fulfill the power coordinated control of the microgrid, its modeling method and corresponding rules are developed, and then the corresponding energy control strategy is discussed, as well as the reasoning demonstration is carried out in this paper. Finally, MATLAB is used to simulate the power coordinated control strategy based on PFTPS. The simulation results show that the proposed strategy can realize the power coordination control of microgrid in a better state.

Keywords: Microgrid · Tissue-like P systems · Power coordination control · Promoters.

1 Introduction

As a new branch of natural computing, many research results which cover many disciplines or fields have emerged since membrane computing was proposed in 1998 [1]. For example, cell-like P systems can be used in the control design of mobile robots [2], tissue-like P systems can be used in the field of spectral clustering [3], and spiking neural P systems can be used in power system fault diagnosis [4]-[6]. In short, membrane computing is developing rapidly and has many achievements.

The various characteristics of the membrane computing, such as distribution, parallelism, non-determinism, and communication, etc., make it suitable for solving various application problems. In [7], cell-like P systems are used to handle the coordinated

* Corresponding author (J. Wu)

control and economic operation of the microgrid, and excellent control and optimization results are obtained. In [8], combined with fuzzy theory and cell-like P systems, the language fuzzy cell-like P systems are applied to the coordinated control of microgrid, which achieves reasonable power distribution and coordinated control requirements. In [9], a distributed fuzzy P system with promoter is proposed for energy control of multi-microgrid systems, and good results have been achieved. Therefore, membrane computing has a good application prospect in the field of control and optimization of microgrid systems.

Microgrid is a new network system, which integrates power generation system, energy storage system, load, and control system [10]. Because of its flexible installation and simple structure, it can adapt to the power supply in different geographical environment. Compared with large power grid, microgrid has many advantages, such as better adaptability, lower cost, and installation requirements. It can also be used to alleviate the problems of uneven distribution and unreasonable structure of energy, and realize the full utilization and sustainable development of new energy. Because microgrid has many characteristics, such as flexible grid structure, diverse power supply types, and complex control methods, etc., it makes the study of mathematical model of microgrid and optimal configuration of a variety of distributed generations (DG) very important. In order to provide users with continuous and reliable power supply, ensure good power quality, and realize the safe and stable operation of the system, it is necessary to carry out reasonable coordination and control of the microgrid, and realize the balance of active power by the cooperation of each unit of the microgrid.

So far, more and more scholars have invested in the research of energy management and stable operation of microgrid. In fact, the energy control of microgrid can be regarded as a control process integrating optimization and decision-making. The control process can be divided into centralized, hierarchical, and distributed in the current research [11]. In [12]-[14], centralized optimization is used to establish energy control models of multi microgrid, and the content on minimization of cost is considered. Due to the need for each control unit to complete the decision interactively, the Multi-agent System (MAS) is introduced in [15]-[17] to model and optimize the energy control of microgrid in the hierarchical and distributed optimization, at last, good results are achieved. In [18], under the trend of energy internet, the research on the demand response of regional energy grid is of great significance, which can absorb renewable energy, realize the balance between supply and demand, and enhance the flexibility and reliability of power grid. Therefore, around this topic, it can be explored by establishing a membrane computing models. The way of energy interaction and information transmission between cells and tissues composed of multiple cells can be applied to the formulation of energy control and the formulation of control scheme.

In view of the randomness and volatility of distributed generation, the information of the microgrid contains uncertainty and inaccuracy. In this paper, the proposed fuzzy tissue-like P system with promoters (PFTPS) is applied to the coordinated control of microgrid, which provides a new and feasible way to solve the problem of microgrid coordination control. Firstly, the promoters and a common language fuzzy system are integrated into the tissue-like P systems, so that large amount of uncertain and inaccurate information contained in the microgrid can be better treated, and its detailed

definition is given. Secondly, the proposed PFTPS is applied to the coordinated control of microgrid, in which the modeling method and corresponding rules are developed, the corresponding energy control strategy is discussed, and the reasoning demonstration is carried out. Finally, MATLAB is used to simulate coordinated control strategy based on PFTPS. The simulation results show that the proposed strategy can better realize the power coordinated control of microgrid.

The structure of this paper is as follows. In Chapter 2, the preliminaries are introduced. In Chapter 3, the specific definition of PFTPS is proposed, and PFTPS is modeled for microgrid. In addition, the coordinated control strategy based on PFTPS of microgrid is proposed. In Chapter 4, the correctness and effectiveness of the proposed method are further verified by simulation experiment based on MATLAB. In Chapter 5, the conclusion is drawn.

2 Preliminaries

2.1 Microgrid

There are two operating modes of microgrid, island mode and grid-connected mode. When the large power grid fails and cannot be recovered immediately, microgrid can turn into an island mode (microgrid disconnects from grid) to ensure uninterrupted power supply for important loads. And there are three coordinated control types of microgrid: master-slave control, peer-to-peer control, and hierarchical control [19]. At present, the main purpose of coordinated control of microgrid is to control the frequency, voltage, and active power of microgrid in proportion, so as to achieve the stable operation of microgrid.

Fig. 1 shows the structure of a simple AC microgrid. Based on this microgrid structure, this paper studies its membrane computing model. This microgrid includes fuel cell (FC), photovoltaic power generation unit (PV), important load (Load1), general load which presents flexible characteristics [20] (Load2), controllable gas turbine (GT) and energy storage unit (Storage). In addition, microgrid is connected to a distribution network via a common connection point (PCC) and a transformer (T). Combining the characteristics of each unit, the selected control units are as follows: GT, Storage, and Load2.

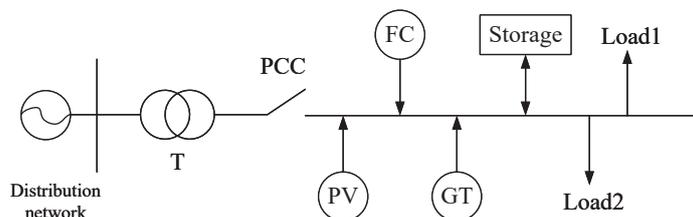


Fig. 1. A simple AC microgrid.

The operation of each unit of microgrid must follow certain restrictions, and must operate under restricted conditions. For example, GT and Load2 must work between the upper and lower limits of the output power. When the system power is too much, it will be cut off. For Storage, its overcharge or over discharge will cause damage, so

the upper and lower limits for charging Storage need to be set. The power of Storage can be shown by the state of charge (SOC). Therefore, in this paper, Storage will stop charging when the SOC is greater than 80%SOC; Storage will stop discharging when the SOC is less than 20%SOC.

2.2 Fuzzy tissue-like P system

As one of the important extensions of cell-like P system, the theory of tissue-like P system is developing rapidly, such as tissue-like P system with evolutionary symport/antiport rules [21], tissue-like P system with triggering ablation rules [22], and tissue-like P systems with promoters [23]-[24]. Compared with the cell-like P system, the tissue-like P system only has monolayer membranes, not multilayer membranes; during the computation, all cells work in parallel, and all of them contribute to the result. So, in this sense, the operation of a tissue-like P system is performed by multiple cells. In order to solve all kinds of fault problems in the process of power system operation, a fuzzy tissue-like P system (FTPS) is proposed in [25]. FTPS can deal with incomplete and uncertain information, which is very suitable for power system fault diagnosis.

Definition 1. A fuzzy tissue-like P system with degree m ($m \geq 1$) is as follows.

$$\Pi = (O, \sigma_1, \dots, \sigma_m, syn, i_0) \quad (1)$$

where:

- (1) O is a finite nonempty alphabet, whose objects are fuzzy multi-sets, represent the fuzzy quantity of the system inputs.
- (2) $\sigma_i, i \in \{1, 2, \dots, m\}$ represents a cell, the format is $\sigma_i = (\omega_{i,0}, R_i)$, $\omega_{i,0}$ represents the initial object value, which is a fuzzy number; R_i is a set of communication rules with weight, the format is $(i, x, j; \alpha)$, x is the object in cell i , α is the weight of the rule.
- (3) $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ represents the connection relationship between the units.
- (4) i_0 represents the output unit.

FTPS is composed of multiple units, when FTPS is stopped, the content of the output unit is regarded as the output of the entire system. In FTPS, cells usually work in parallel. In addition, the basic definition of fuzzy multi-sets [26] is as follows:

Definition 2. Assuming that $U = (u_1, u_2, \dots, u_n)$ is on the universe of discourse, then a fuzzy multi-set A on U can be defined as:

$$A = \{\{f_A^1(u_1), f_A^2(u_1), \dots, f_A^n(u_1)\}/u_1, \dots, \{f_A^1(u_n), f_A^2(u_n), \dots, f_A^n(u_n)\}/u_n\} \quad (2)$$

where, $\{f_A^1(u_i), f_A^2(u_i), \dots, f_A^n(u_i)\}/u_i$ is the membership of u_i , whose value is a multi-set on $[0, 1]$, and it represents the frequency of occurrence possibility of u_i is $\{f_A^1(u_i), f_A^2(u_i), \dots, f_A^n(u_i)\}$.

3 Application of PFTPS to power coordinated control of microgrid

In order to solve the coordinated control problem of microgrid and achieve its power balance, membrane computing needs to be improved to make itself suitable for building system models and dealing with logical reasoning problems. Therefore, combining promoter and fuzzy theory, an improved tissue-like P system PFTPS is proposed in this paper, which can better deal with the uncertainty and inaccuracy of microgrid information.

3.1 Fuzzy tissue-like P systems with promoters

Definition 3. A PFTPS with degree m ($m \geq 1$) is as follows.

$$\Pi = (O, Q, \sigma_1, \dots, \sigma_m, B_1, \dots, B_m, syn, i_0) \quad (3)$$

where:

- (1) O is a nonempty alphabet of objects which are fuzzy multi-sets.
- (2) $Q = \{a_i | i = 1, 2, \dots, m\}$, Q is a collection of a_i states. A PFTPS structure with degree m has m states, and m also means that PFTPS has m cells in total. The states of that PFTPS include input stage, middle stage, and output stage.
- (3) σ_i , $i \in \{1, 2, \dots, m\}$ represents the cell structure in PFTPS, the format is $\sigma_i = (\omega_{i,0}, B_i, R_i)$, $\omega_{i,0}$ represents the initial object value, which is a fuzzy number; B_i represents the promoter in σ_i ; R_i is a set of rules, there are two types of rules in middle stage: evolution rules and communication rules. The formats of evolution rule are $R_{iA} : (x_i b_i \rightarrow (y_j b_i, in_{n+1}); x_i \geq b_i)$ and $R_{iB} : (x_i b_i \rightarrow (y_j b_i); x_i < b_i)$; the format of communication rule is $((i, x_i b_i / y_j b_j, j); b_i > b_j)$, x is the object in cell i , y is the object in cell j , $j \in \{1, 2, \dots, m\}$, b is the promoter in cell i or j . In the input stage, the format of rule is $(x_i \rightarrow (x_i, in_{n+1}))$. In the output stage, the format of rule is $(x_i \rightarrow (x_i, out_{n+1}))$.
- (4) B_i , $i \in \{1, 2, \dots, m\}$ represents promoter, it is different from ordinary catalysts, and promoters can evolve independently.
- (5) $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$, with $(i, i) \notin syn$ for $i \in \{1, 2, \dots, m\}$ is the directed graph and represents the connection relationship between the units.
- (6) i_0 represents the output unit, $i_0 = 1$ means output in cell 1.

For the two formats of evolution rules in middle stage, the former indicates that when the promoter b_i and the object x_i exist and meet excitation condition $x_i \geq b_i$, R_{iA} will be activated, and the object x_i will be transformed into y_k and transported from membrane n to the next stage membrane $n + 1$. In contrast, when the promoter b_i and the object x_i exist and meet excitation condition $x_i < b_i$, R_{iB} will be activated, and the object x_i will be transformed into y_k and stay in membrane n . For the communication rules in middle stage, when the promoters b_i, b_j and the objects x_i, y_j exist in the two membranes and meet excitation condition $b_i > b_j$, R_i will be activated, and the object x_i in membrane i and the object y_j in membrane j will be exchanged. For the rules of input stage, when the object x_i exist, R_i will be activated, and the object x_i will be

transported into the next stage. For the rules of output stage, when the object x_i exist, R_i will be activated, and the object x_i will be transported into environment.

According to the literature [27] mentioned, the emergence of a chemical substance makes the occurrence of biochemical reactions possible, this chemical substance is different from the catalyst, it can evolve alone, and the evolution process can be carried out in parallel with the reaction. Here it can be called promoter. There are two types promoters are defined for the membranes in the system model: native promoter B^* and derivative promoter $B^\#$ [9]. The position of the native promoter in the model does not change, and its value is automatically evolved by the rules in the initial pattern of the membrane. The position of the native promoter in the model is unaltered, and its value is automatically evolved according to the rules in the initial pattern of the membrane.

As shown in Fig. 2, it is a simple PFTPS structure. The structure has 5 cells, which are divided into input stage, middle stage, and output stage. Each cell contains objects and rules, and a directed line segment represents a channel which can transmit signal. For signal transmission, one-way line segment can only carry out one-way signal transmission and two-way line segment can carry out two-way signal transmission.

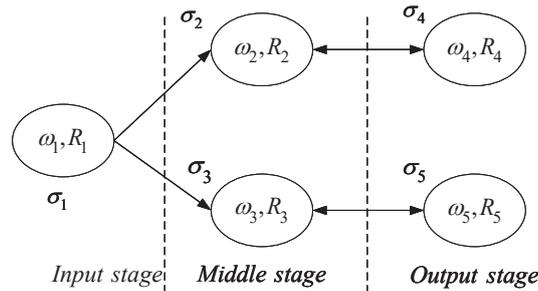


Fig. 2. A simple PFTPS structure.

A more complex coordinated control model of microgrid based on above definition and structure is proposed in next section. In PFTPS, multiple data of different control objects are input, and the data of these objects are output after calculation. The calculation process is realized through the parallel operation of each cell and multi-line data transmission in PFTPS.

3.2 Reasoning calculation

In order to test the correctness and feasibility of the constructed microgrid model based on PFTPS, the reasoning calculation is implemented by using the data which meets the system input object requirements and reflects the microgrid operating status. Now a PFTPS model of the above microgrid is established, as shown in Fig. 3. Three kinds of operation states data for microgrid are used to carry out the reasoning calculation, namely: (1) island mode can be realized; (2) grid-connected mode can be realized; (3) when f is less than the rated frequency. The reasoning results are shown in Table 1.

To make the reasoning process intelligible, a detailed example of the reasoning process is introduced with the operation state (1) as an example. A PFTPS structure Π with degree $m = 7$ is as follows.

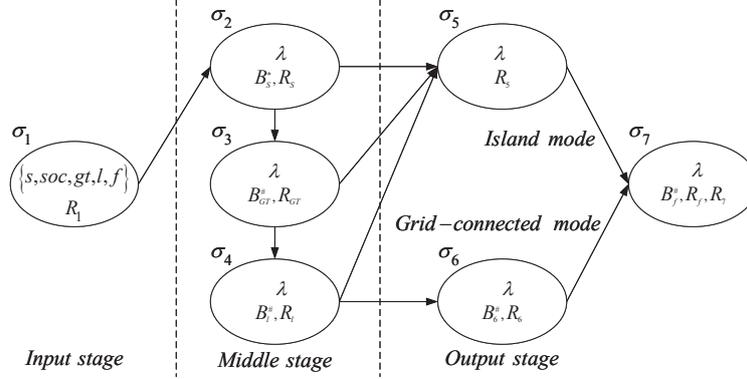


Fig. 3. Microgrid model based on PFTPS.

$$\Pi = (O, Q, \sigma_1, \dots, \sigma_7, B_1, \dots, B_7, syn, i_0) \quad (4)$$

where:

- (1) $O = \{s, soc, gt, l, f\}$;
- (2) $Q = \{a_i | i = 1, 2, \dots, 7\}$, a_1 is the initial state, a_2, a_3, a_4 are the intermediate states, and a_5, a_6, a_7 are the termination states;
- (3) $\sigma_1 = (\omega_1 = \{s, soc, gt, l, f\}, \lambda, R_1), \sigma_2 = (\omega_2 = \lambda, B_s^*, R_s), \sigma_3 = (\omega_3 = \lambda, B_{GT}^\#, R_{GT}), \sigma_4 = (\omega_4 = \lambda, B_l^\#, R_l), \sigma_5 = (\omega_5 = \lambda, \lambda, R_s), \sigma_6 = (\omega_6 = \lambda, B_s^\#, R_s), \sigma_7 = (\omega_7 = \lambda, B_f^\#, R_f \& R_7)$;
- (4) $syn \subseteq \{(1, 2), (2, 3), (2, 5), (3, 4), (3, 5), (4, 5), (4, 6), (5, 7), (6, 7)\}$;
- (5) $i_0 = 7$.

The detailed reasoning process is described as follows.

Step 1. Objects are included in σ_1 , there are $(s, 0.2), (soc, 0.5), (gt, 0.1), (l, 0.2), (f, 0.7)$, and $b_s = 0.5$. Rule R_1 meets the excitation conditions, hence the objects move into σ_2 . In σ_2 , there are $0.2 < 0.5 \& 0.5 < 0.8 \& 0.5 > 0$. Rule R_s meets the excitation conditions, hence $s = +0.2$, then $soc = 0.5 + 0.1 = 0.7$, f has changed and is still greater than 0, but the specific value cannot be judged, it can be set to ex , and the data is sent to σ_3 .

Step 2. In σ_3 , the promoter $B_{GT}^\#$ changes to $b_{GT} = b_s - 0.2 = 0.3, 0.1 < 0.3$, rule R_{GT} meets the excitation conditions, hence $gt = -0.1$, f changes and it is still greater than 0, but the specific value cannot be judged, it can be set to ex . Then, the data is transferred to σ_4 . In σ_4 , the promoter $B_l^\#$ changes to $b_l = b_{gt} - |-0.1| = 0.2, 0.2 = 0.2$, rule R_l meets the excitation conditions, hence $l = +0.2$, f has changed and it is still greater than 0, but the specific value cannot be judged, it can be set to ex . And the data is transferred to σ_5 to complete the islanding operation.

Step 3. In σ_5 , rule R_s only performs transfer operation. At this time, the power returns to normal, and f returns to normal, but the specific value cannot be judged, it can be set to ex , and the data is transferred to σ_7 .

Step 4. In σ_7 , f may have a little fluctuation, and fluctuate around 0. Rule R_f meets the excitation conditions, hence it is executed. When the data is greater than 0, the value

is positive; when the data is less than 0, and the value is negative. Then, rule R_7 is executed, and the calculated data $\omega_7 = \{(s, +0.2), (soc, 0.7), (gt, -0.1), (l, +0.2), (f, ex)\}$ is output to the environment at the end of the operation.

The meaning of output result is as follows: $(s, +0.2)$ represents the power of Storage increases 20%SOC; $(soc, 0.7)$ represents Storage is charged to 70%SOC; $(gt, -0.1)$ represents that GT reduces the power emitted by 0.1; $(l, +0.2)$ represents that load increases power demand by 0.2; (f, ex) means that the value of f is unknown and can be determined in a later link. This model achieves power balance in cell 4 and outputs the value of the controllable variable in cell 7. Currently, microgrid is operating in island mode.

Table 1. Reasoning results of three operation states.

Group	Input	Output
1	$(s, 0.2), (soc, 0.5), (gt, 0.1), (l, 0.2), (f, 0.7)$	$(s, +0.2), (soc, 0.7), (gt, 0.1), (l, +0.2), (f, ex)$
2	$(s, 0.2), (soc, 0.6), (gt, 0.3), (l, 0.2), (f, 0.6)$	$(s, 0.2), (soc, 0.4), (gt, +0.3), (l, 0.2), (f, ex)$
3	$(s, 0.3), (soc, 0.9), (gt, 0.5), (l, 0.5), (f, 0.2)$	$(s, 0.3), (soc, 0.6), (gt, +0.1), (l, 0), (f, ex)$

Similarly, in the second case, the reasoning result shows that microgrid still cannot maintain its own power balance after its internal control, the system needs to buy the power from distributed network to achieve its own power balance, the output reflects the adjustment measure at the next moment. In the third case, when f is less than the rated frequency, the reasoning result shows that microgrid also can realize its power balance. Therefore, the correctness and feasibility of the proposed theory are verified through the reasoning results.

3.3 Coordinated control strategy based on PFTPS of microgrid

This coordinated control strategy of microgrid mainly adjusts the controllable units separately by detecting the power and frequency of AC feeder. The control strategy based on PFTPS of microgrid is shown in Fig. 4.

Firstly, the system power is compared to determine the power shortage or excess. Then, the output state of the controllable units is determined by comparing with the adjustable capacity of the controllable units one by one. Finally, if microgrid still cannot meet the power balance after self-regulation, the power interaction is carried out to the distribution network to achieve power balance.

In this scheme, PCC represents whether the microgrid is connected to the external power grid, if $PCC = 0$, the microgrid is not connected with the external power grid and enters the island mode, the power balance of the microgrid is met after the self-regulation of the microgrid; if $PCC = 1$, the microgrid relates to the external power grid and enters the grid-connected mode, and the microgrid will interact with the distribution grid for energy. $\Delta P1$ represents the initial power shortage value of the system. P_{GT} represents the adjustable power of the GT. $\Delta P2$ represents the power shortage value of the system after GT adjustment.

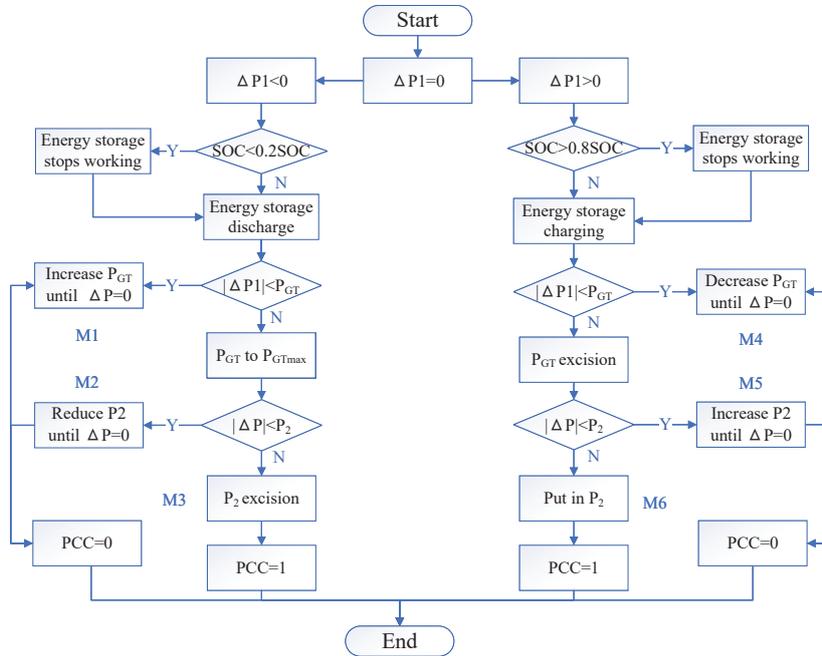


Fig. 4. Control strategy based on PFTPS of microgrid.

4 Simulation analysis of coordinated control for microgrid

In this section, simulation analysis will be performed based on the derivation test in the previous section to further determine the correctness and feasibility of the model.

4.1 Simulation of traditional coordination control method

This section will simulate the microgrid system with traditional control strategy. In this strategy, controllable energy unit is Storage. Because the energy storage unit can stabilize and buffer the fluctuation caused by PV and load fluctuation, the energy storage unit is used to adjust part of the power to achieve the improvement of power quality. Besides, PV and FC are controlled by constant frequency and constant voltage mode (PQ control mode), GT does not participate in regulation in the traditional control strategy.

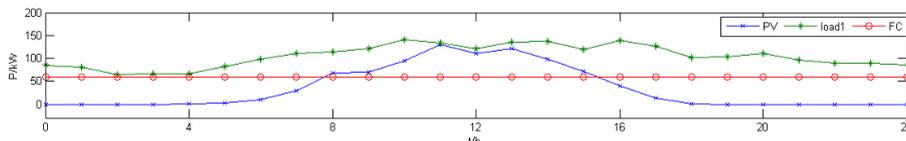


Fig. 5. Power change curve of each unit.

The simulation results are shown in the above curves, the power change curves of each unit are shown in Fig. 5, the electricity curve of Storage is shown in Fig. 6, and the power curve of microgrid and the frequency curve of AC feeder are shown in Fig. 7

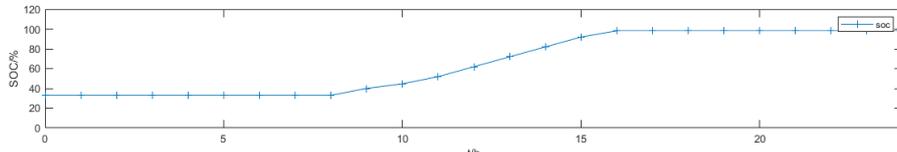


Fig. 6. Electricity curve of Storage.

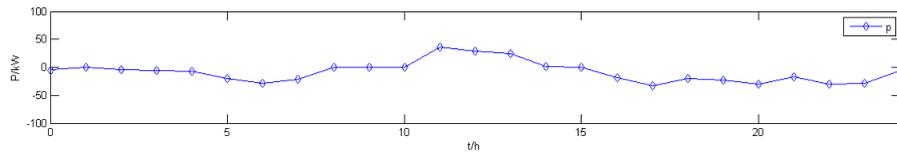


Fig. 7. Power curve of microgrid.

and Fig. 8. From Fig. 6, we can see the Storage is not connected during 0 – 7h, so there is no charge and discharge of energy storage at this time. At the same time, the system power can be basically balanced. From Fig. 5, it can be seen that the power generation of PV unit is large, and the excess power of the system is obvious in 8 – 16h. Therefore, the Storage is connected by charging, the value is positive, and the SOC shows an upward trend. In 16 – 24h, the SOC reaches full state, and the overcharge of the Storage will affect its service life. From Fig. 7 and Fig. 8, it can be seen that part of the power imbalance can be alleviated in the system with Storage during 2 – 4h and 8 – 10h, which can reduce the fluctuation of system power to a certain extent. However, the adjustment capacity of a single storage unit is limited, and it still has large fluctuations in most of the time, so the controllable unit needs further investment.

4.2 Simulation of coordinated control based on PFTPS in microgrid

In this section, the control model of microgrid based on PFTPS is simulated. The method of combining energy management and SOC power state is adopted by the model. Not only the excess and deficiency of power in the model are considered, but also the limitation of SOC power state is considered. Then the power balance is realized by judging the common control mode one by one. Fig. 9-Fig. 13 are the simulation results. Fig. 9 shows the power change curve of each controllable unit, Fig. 10 shows the power change curve of the SOC, Fig. 11 shows the power curve before control and power interaction curve with distributed network, Fig. 12 shows the power curve of microgrid after control, and Fig. 13 shows the frequency curve of AC feeder after control.

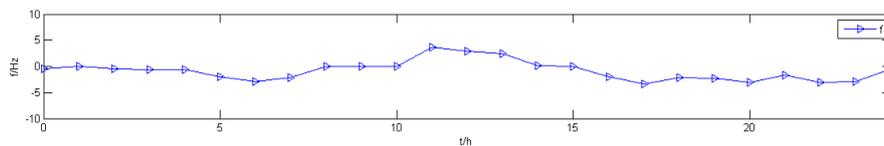


Fig. 8. Frequency curve of AC feeder.

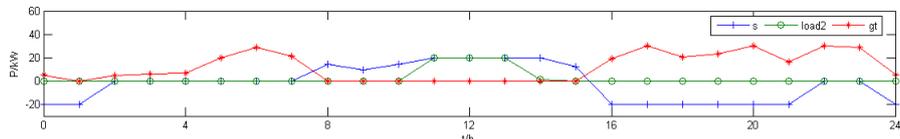


Fig. 9. Power change curve of each controllable unit.

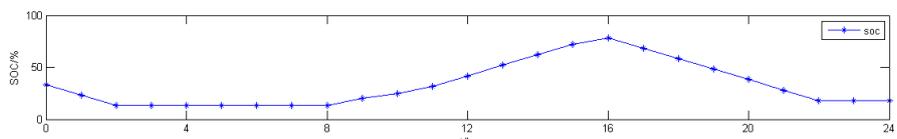


Fig. 10. Power change curve of the SOC.

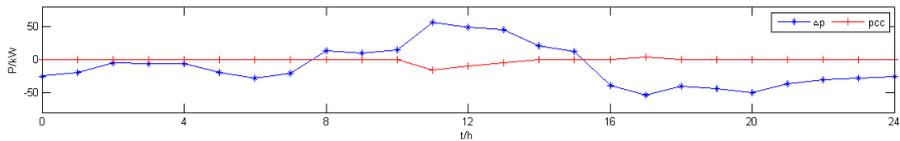


Fig. 11. Power curve before control and power interaction curve with distributed network.

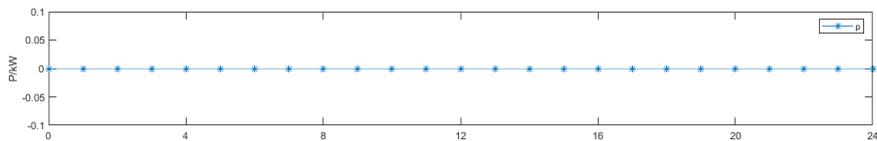


Fig. 12. Power curve of microgrid after control.

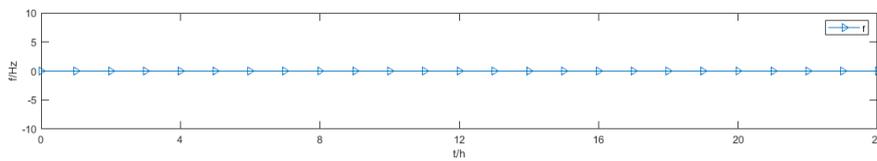


Fig. 13. Frequency curve of AC feeder after control.

We can see that 0 – 7h is at night, PV does not generate power and the system power consumption is excessive. At this time, GT and Storage are put into operation to keep the power balance of microgrid. And during 0 – 2h, the SOC is greater than 20%, so the Storage can be discharged. When the SOC is lower than 20% and cannot be discharged at 3h, the charging and discharging of Storage ends, and the power balance is met only by GT. In 8-11h, the illumination conditions become stronger and the power emitted by PV increases gradually. At this time, the SOC is at the lower limit, and the excess power of the system charges the Storage. In 11-13h, the power generation of PV in this section is the strongest in the whole day, and the excess power of the system is large. At this time, GT is cut off, and Storage and Load2 are input to absorb excess power. However, the power balance cannot be met at this time. Therefore, as shown in Fig. 11, the microgrid has energy interaction with the distributed network and sells power to the distributed network. During 13 – 15h, the light intensity gradually reduces, the power generation of PV decreases, and the Storage is still in a charging state. Until 16h, the Storage reaches the upper limit of 80%, and Storage ends charging. During 16–21h, the output of PV gradually decreases to 0, the power demand of Load1 does not decrease significantly, and the input power needs to be increased. At this time, GT and Storage are put into operation, the Storage is discharged, and the general load is cut off at the same time. However, at 17h, the microgrid still can not meet the load demand through its own regulation, so the microgrid buys power from the distribution network to meet the power balance. In 21 – 24h, the power required by the load decreases, while the power generation of the system increases to a certain extent. The SOC of the Storage is less than 20%, and the discharge is stopped so the system meets the power balance. The simulation results show that the coordinated control strategy based on PFTPS can balance the power of the microgrid and achieve the expected purpose, which verifies that the method is feasible and reasonable.

In summary, the following conclusions can be drawn through these simulation results. The power and frequency of microgrid using traditional control methods fluctuate greatly, and the power quality cannot meet the requirements. However, the coordinated control method based on PFTPS can reduce the fluctuation of power and frequency, and improve the power quality. At the same time, because the proposed method considers the upper and lower limits of Storage, the problems of overcharge and over discharge can be avoided, and its service life can be prolonged. In addition, compared with the traditional control methods, the coordinated control method based on PFTPS can reduce the interaction frequency between the microgrid and the distributed network, so the impact on the distributed network is reduced, and the stability of the grid is promoted.

In addition, the communication between microgrid units plays a very important role in the coordinated control strategy of the system. So compared with the cell-like P systems in [7], the language fuzzy cell-like P systems in [8], distributed fuzzy P systems with promoters in [9], the proposed PFTPS is based on tissue-like P systems, it has its own communication channels, which can better simulate the communication between microgrid units, to deal with all kinds of uncertain and inaccurate information of microgrid.

5 Conclusions

The structure of microgrid is diverse and the control mode is complex. Some impact may be brought by intermittent and random distributed generation when it is connected to the grid. In order to provide users with continuous and reliable power supply, ensure the good power quality and realize the safe and stable operation of the system, a fuzzy tissue-like P system with promoters is proposed and applied to the coordinated control of microgrid. In addition, the internal rules and operation process are discussed; Then, the coordinated control model of microgrid based on PFTPS is constructed, and its coordinated control strategy is given. The correctness of the theory is verified by reasoning. Finally, the power balance within the microgrid and the interaction between the microgrid and the distributed network is realized through experiments, and the frequency stability of the system is maintained.

Acknowledgment

This work was partially supported by the School level key project of Chengdu Technological University (No. 2019ZR002) and the School level key project of Chengdu Technological University (No. 2019ZR003).

References

1. Zhang, G., Prez-Jimnez, M. J., Riscos-Nñez, A., et al: Membrane Computing Models: Implementations. Springer, Singapore (2021)
2. Buiu, C., Vasile, C., Arsene, O.: Development of Membrane Controllers for Mobile Robots. *Information Sciences*. **187**, 33–51 (2012)
3. Yin, X., Liu, X.: An Improved Spectral Clustering Based on Tissue-like P System. *Bio-Inspired Computing: Theories and Applications*, Springer Berlin Heidelberg (2021)
4. Wang, T., Zhang, G., Zhao, J., et al: Fault Diagnosis of Electric Power Systems based on Fuzzy Reasoning Spiking Neural P Systems. *IEEE Transactions on Power Systems*. **30**(3), 1182–1194 (2015)
5. Wang, J., Peng, H., Yu, W., et al: Interval-valued Fuzzy Spiking Neural P Systems for Fault Diagnosis of Power Transmission Networks. *Engineering Applications of Artificial Intelligence*, **82**(6), 102–109 (2019)
6. Liu, Y., Chen, Y., Paul, P., et al: A Review of Power System Fault Diagnosis with Spiking Neural P Systems. *Applied Sciences*, **11**(10), 4376 (2021)
7. Luo, J., Wang, J., Shi, P., et al: Micro-grid Economic Operation using Genetic Algorithm based on P Systems. *ICIC Express Letters*. **9**(2), 609–618 (2015)
8. Wang, J., Chen, K., Li, M., et al: Cell-Like Fuzzy P System and Its Application in Energy Management of Micro-Grid. *Journal of Computational and Theoretical Nanoscience*. **13**(6), 3643–3651 (2016)
9. Yu, W., Wang, J., Tao, W., et al: Distributed Fuzzy P Systems with Promoters and Their Application in Power Balance of Multi-microgrids. *International Conference on Bio-inspired Computing: Theories and Applications*. pp. 329–342. Springer, Heidelberg (2017)
10. Chen, L., Niu, Y.: Two-stage Game Framework for Energy Management in Islanded Multi-microgrid System. *IET Generation Transmission & Distribution*, **14**(23), 5439–5446 (2020)

11. Peng, Y. I., Hong, Y. G.: Distributed Cooperative Optimization and Its Applications. *Scientia Sinica Mathematica*. **46**(10), 1547–1564 (2016)
12. Liu, Q., Wang, S., Zhao, Q., et al: Effects of Dynamic Topology Reconfiguration for Optimal Operation in Multi-Microgrid System. *IOP Conference Series Earth and Environmental Science*, **645**, 012044 (2021)
13. Wang, C., Zhang, G., Chen, S., et al: Bilevel Energy Optimization for Grid-connected AC Multimicrogrids. *International Journal of Electrical Power & Energy Systems*. **130**(1), 106934 (2021)
14. Sun, Y., Cai, Z., Zhang, Z., et al: Coordinated Energy Scheduling of a Distributed Multi-Microgrid System Based on Multi-Agent Decisions. *Energies*, **13**(16), 4077 (2020)
15. Gomes, L., Vale, Z. A., Corchado, J. M.: Multi-Agent Microgrid Management System for Single-Board Computers: A Case Study on Peer-to-Peer Energy Trading. *IEEE Access*, **99**, 1–1 (2020)
16. Harmouch, F. Z., Krami, N., Hmina, N.: A Multiagent based Decentralized Energy Management System for Power Exchange Minimization in Microgrid Cluster. *Sustainable Cities and Society*. **40**, 416–427 (2018)
17. Zheng, Y., Song, Y., Hill, D. J., et al: Multiagent System based Microgrid Energy Management via Asynchronous Consensus ADMM. *IEEE Transactions on Energy Conversion*. **33**(2), 886–888 (2018)
18. Huang, K., Qian, A. I., Zhang, Y., et al: Challenges and Prospects of Regional Energy Network Demand Response Based on Energy Cell/Tissue Architecture. *Power System Technology*. **43**(9), 3149–3160 (2019)
19. Ghaderyan, D., Pereira, F. L., Aguiar, A. P.: A Fully Distributed Method for Distributed Multiagent System in a Microgrid. *Energy Reports*. **7**(5), 2294–2301 (2021)
20. Hungerford, Z., Bruce, A., Macgill, L.: The Value of Flexible Load in Power Systems with High Renewable Energy Penetration. *Energy*. **188**(1), 115960.1–115960.12 (2019)
21. Luo, Y., Guo, P., Jiang, Y., et al: Timed Homeostasis Tissue-Like P Systems with Evolutional Symport/Antiport Rules. *IEEE Access*, **8**, 1–1 (2020)
22. Sang, X., Liu, X., Zhang, Z., et al: Improved Biogeography-Based Optimization Algorithm by Hierarchical Tissue-Like P System with Triggering Ablation Rules. *Mathematical Problems in Engineering* (2021)
23. Song, B., Pan, L.: The Computational Power of Tissue-like P Systems with Promoters. *Theoretical Computer Science*. **641**, 43–52 (2016)
24. Yan, S., Xue, J., Liu, X.: An Improved Quicksort Algorithm Based on Tissue-Like P Systems with Promoters. Springer, Cham (2019)
25. Peng, H., Wang, J., Shi, P., et al: Fault Diagnosis of Power Systems using Fuzzy Tissue-like P Systems. *Integrated Computer Aided Engineering*. **24**(4), 401–411 (2017)
26. Miyamoto, S.: Multisets and Fuzzy Multisets. *Soft Computing and Human-centered Machines*. Springer, Japan (2000)
27. Păun, G.: Membrane Computing: An Introduction. *Theoretical Computer Science*. **287**(1), 73–100 (2002)

Time-free RSSNP systems with synaptic co-transmission

Luping Zhang and Fei Xu*

Key Laboratory of Image Processing and Intelligent Control of Education Ministry of China, School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan 430074, China
lpzhang@hust.edu.cn, fxu@hust.edu.cn

Abstract. Time-free spiking neural P systems with rules on synapses (RSSNP systems) are a class of spiking neural P systems with rules on synapses, whose computation results are independent of the execution time of rules. In this work, the idea of synaptic co-transmission, inspired by the related biological observation, is introduced in time-free RSSNP systems, where a family of sets of co-transmission synapses is given, and spikes pass through synapses which are in the same set of co-transmission synapses taking the same execution time. The computation power of time-free RSSNP systems with synaptic co-transmission is investigated. It is proved that time-free RSSNP systems with synaptic co-transmission are Turing universal as both number accepting devices and number generating devices.

Keywords: Bio-inspired computing, Membrane computing, Spiking neural P system, Time-free solution

1 Introduction

Each neuron contains abundant information even in tiny space, and it can be used as a container and a processor of information [15]. Neural networks formed by neurons provide an interesting subject for the analysis and modeling of the information processes of various neural networks, such as artificial neural networks [11], silicon neurons models [45], spiking neural models [10].

Spiking neural P systems are such computing models using the computing paradigm of membrane computing which abstracts computation notions from the biological organizations and reactions in living cells to execute computations in the distributive and parallel manner [4, 26, 32, 33, 47, 50, 51]. A great number of variants of spiking neural P systems have been introduced so far: some of them incorporate some ideas from neuroscience such as neurobiological features of the potential-dependent dynamic feature of neuron populations [28, 44], the request-response communications between neurons [26, 46], the replication behaviors [25]; some of them incorporate concepts of maths, such as antimatter [23], generalization of rules [14]; some of them incorporate notions of computer science [1, 13].

* Corresponding author

More and more variants of spiking neural P systems have been used to solve practical problems [8, 49].

Spiking neural P systems with rules on synapses (RSSNP systems) is a variant of spiking neural P systems [40]. RSSNP systems can be represented by directed graphs, where a spike denoted by the symbol a is the basic unit of information and pieces of information are encoded by the multi-set of spikes, such as a^k ; a neuron denoted by a node is the carrier of information; a synapse between neurons denoted by a directed arc with spiking rules as well as forgetting rules is the one-way bridge for compiling and delivering information. RSSNP systems are founded on the assumption of the massively parallel application of rules: a maximal set of enabled rules are obligatory to be applied, and the execution of each rule is strictly restricted in one computation step. In literature, the computation power as well as the efficiency of RSSNP systems are extensively investigated, e.g., [5, 39–42].

Timed RSSNP systems are an extension of RSSNP systems which work in the time mode, i.e., the application of different rules in these systems could take different integer durations [6]. Theoretically, the timed mode which abandons the global synchronization in the prototype of P systems seems to be hard to be managed, yet certain important computing properties are available by investigating computing models in membrane computing working in the timed mode [2, 7, 27, 35]. For instance, the spiking neural P systems with extended rules [27], the P systems with active membranes [37], tissue-like P systems with evolutionary symport/antiport rules [19], P systems with active membranes using promoters [18], whose computation power are proved to be robust independent of the execution time of rules. Because the robust, some timed P systems are used to solve computation-hard problems, such as those in [18, 19, 35, 37]. To our knowledge, there is no literature in the realm of timed RSSNP systems.

In the fields of computer science, a large number of new computing models are continuously introduced, and their important properties, such as computation power and space capacity, describing the ability of the systems. However, many of them have no physical realization, and one reason for the fact is that small changes in the implementation could affect the ability to compute or solve a problem. The motivation of this work is to fill the gap of studying the timed RSSNP systems to search some element (besides extended rules [27], active membranes [37], evolutionary rules [19], promoters [18] mentioned before) benefit for the robust computation power, to help the design and implementation of the corresponding simulator, and to solve some application problems which can be converted to the corresponding computations.

The idea of synaptic co-transmission sparks from the neurobiological observation that neurotransmitters are co-transmitted to the target neurons at the same neuronal event [17, 20]. In this work, a family of sets of co-transmission synapses are introduced in timed RSSNP systems using standard rules, where spikes are delivered through each of the synapses in the same set of co-transmission synapses during the same period. Namely, if a spike is delivered through a synapse which is in a set of co-transmission synapses by using an enabled rule

on the synapse, there is a spike delivered through an arbitrary synapse in the same set by using the corresponding rule on synapse during the same execution time; otherwise, no spike can be delivered through any synapse in the set.

It is proved that timed (even time-free) RSSNP systems with synaptic co-transmission are universal using only standard firing rules (without extended rules, forgetting rules, or delay), as they exert two functions: generating numbers, accepting numbers. The result shows that synaptic co-transmission is a benefit mechanism or the robustness of computing models working in the timed mode, besides extended rules [27], active membranes [37], evolutionary rules [19], promoters [18]. In addition, this work also provides an universal time-free RSSNP system with synaptic co-transmission consisting of 101 neurons and 89 sets of co-transmission synapses to accept numbers. The results show that the synaptic co-transmission is benefit for the robustness of universality of timed RSSNP systems in the sense that computation results are independent of the execution time of rules.

The following sections are organized as follows. The definition of timed RSSNP systems with synaptic co-transmission is given in Section 2. The generating numbers power of timed RSSNP systems with synaptic co-transmission is investigated in Section 3. The accepting numbers power of timed RSSNP systems with synaptic co-transmission is proved to be universal, and an universal RSSNP system with synaptic co-transmission for accepting numbers is given in Section 4. The conclusions and some hints for the future work are given in the final section.

2 Timed RSSNP Systems with Synaptic Co-transmission

In this section, the formal definition of timed RSSNP systems with synaptic co-transmission is given. For the primitive definition of RSSNP systems, please refer to [30, 40].

An alphabet is a nonempty set of symbols. For an alphabet O , the set of all finite strings of symbols over O including the empty string λ is denoted by O^* , and the one without the empty string λ is denoted by O^+ . The set of natural numbers is denoted by N , and the family of number sets that Turing machine can compute is denoted by NRE . For more related notations and notions in the theory of formal language, please refer to [31].

Definition 1. *A timed RSSNP system with synaptic co-transmission is a construct of the form*

$$\Pi(e) = (O, \sigma_1, \sigma_2, \dots, \sigma_m, \text{syn}, S, R, e, i_{out}),$$

where

- $O = \{a\}$ is a singleton alphabet;
- $\sigma_i = (n_i)$ ($1 \leq i \leq m$) is a neuron labeled σ_i initially consisting of n_i spikes;
- $\text{syn} = \{(i, j) \mid i, j = 1, 2, \dots, m, \text{env}; i \neq j\}$ is the set of synapses, where each $(i, j) \in \text{syn}$ is a synapse connecting from neuron i to neuron j ;

- $S = \{S_1, \dots, S_k\} \subseteq \mathcal{P}(syn)$ is the family sets of co-transmission synapses;
- $R = \cup_{i \neq j} R_{(i,j)}$ ($i, j = 1, 2, \dots, m, env$), where $R_{(i,j)}$ is the set of rules on the synapse (i, j) of the following two forms:
 - spiking rule: $E/a^c \rightarrow a$ ($c \geq 1$), where E is a regular expression over $\{a\}$;
 - forgetting rule: $a^s \rightarrow \lambda$, where $\{a^s\} \cap L(E) = \emptyset$ for any $E/a^c \rightarrow a \in R_{(i,j)}$;
 for a set of synapses $S' \subseteq syn$, we denote $R(S') = \cup_{(i,j) \in S'} R_{(i,j)}$;
- $e : R \rightarrow N - \{0\}$ is the mapping for execution time of the rules on synapses; for each rule on synapse, the mapping specifies a positive number (an execution time) to it; for two rules in the same set of co-transmission synapses, they have the same execution time, that is, $e(r') = e(r'')$, if $r', r'' \in R(S_i)$ and $S_i \in S$;
- i_{out} indicates the output neuron.

A spiking rule is applied as follows. Let $\sigma_i, \sigma_j \in \Pi(e)$ be two neurons, and (i, j) be a synapse with the rule on synapse $((i, j), r : E/a^c \rightarrow a)$. If the rule on synapse $((i, j), r : E/a^c \rightarrow a)$ is enabled as neuron σ_i contains k spikes meeting the condition $a^k \in L(E)$, then the rule can be applied by consuming c spikes from neuron σ_i and sending one spike to neuron σ_j . The produced spike is sent to the environment if neuron $\sigma_{i_{out}}$ applies the spiking rule on the synapse (i_{out}, env) . If there are two (or more) enabled rules on the same synapse (i, j) at a given time, only one enabled rule can be nondeterministically chosen to be applied at that time.

If the forgetting rule on synapse $((i, j), r : a^s \rightarrow \lambda)$ is enabled as neuron σ_i contains s spikes, then the enabled forgetting rule is applied by removing the s spikes from neuron σ_i .

It is possible that there could be two or more types of spike consumption for one neuron at a give time. For instance, the enabled rules $E_1/a^{c_1} \rightarrow a \in R_{(i,j)}$ and $E_2/a^{c_2} \rightarrow a \in R_{(i,k)}$ ($j \neq k$) emerging from the same neuron σ_i contradict each other for different numbers of spike consumption $c_1 \neq c_2$. In such case, the rules of only one type of spike consumption are non-deterministically chosen to be applied at that time.

The execution time of each rule is specified by the mapping e . It is supposed that there is an external clock which marks the time units (steps) of equal length, and the execution time of each rule is an integral multiple of one step [27]. If the rule $((i, j), r)$ with $e(r) \geq 1$ is applied at step $t + 1$, then any other rule on the synapse (i, j) can not be used at step $t + 1, t + 2, \dots, t + e(r) - 1, t + e(r)$, and neuron σ_j can not receive the produced spike from neuron σ_i until step $t + e(r)$.

In this work, a family of sets of co-transmission synapses $S = \{S_1, \dots, S_k\} \subseteq \mathcal{P}(syn)$ is introduced in timed RSSNP systems. For a given set of co-transmission synapses S_i ($S_i \in S$), the set of rules on synapses which are in S_i is denoted by $R(S') = \cup_{(i,j) \in S'} R_{(i,j)}$. A rule on synapse $r \in R(S_i)$ can not be applied until each rule on synapse is enabled; if the application of $r \in R(S_i)$ is started at step $t + 1$ and ended at step $t + e(r)$, any rule r' with the condition $r' \in R(S_i)$ is applied during the same period.

The configuration of a timed RSSNP system with synaptic co-transmission at a given time is described not only by the number of spikes in each neuron but also the left number of time-step when enabled rules on the synapses can be used. A configuration of a timed RSSNP system is of the form $(\eta_1/t_1, \eta_2/t_2, \dots, \eta_m/t_m)$ where η_i ($1 \leq i \leq m$) is the current number of spikes in neuron σ_i , and the spikes in neuron σ_i can be consumed after $t_i \geq 0$ steps.

The initial configuration is given by $C_0 = (n_1/0, n_2/0, \dots, n_m/0)$. By using the spiking rules and forgetting rules, one configuration of a system is transitioned to a successor $C_t \Rightarrow C_{t+1}$. A sequence of transitions starting from the initial configuration forms a computation. A halting computation $C_0 \Rightarrow C_1 \Rightarrow \dots \Rightarrow C_h$ is associated with the halting configuration C_h where there no rule on any synapse can be applied.

The result of a computation is defined as the total number of spikes emit to the environment by the output neuron $\sigma_{i_{out}}$ in the halting computation. The set of numbers generated by a computing model M is denoted by $N(M)$, and the set of numbers generated/accepted by a timed RSSNP system is denoted by $N(\Pi(e))$.

Time-free RSSNP systems which are robust against the different execution time of rules are a sub-type of timed RSSNP systems. Namely, a time-free RSSNP system $\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn, e, i_{out})$ produces the same set of natural numbers $N(\Pi(e_1)) = N(\Pi(e_2))$ for different mappings $e_1 \neq e_2$. The set of natural numbers generated by time-free system Π is denoted by $N(\Pi)$.

The family of sets of numbers generated by timed RSSNP systems with synaptic co-transmission of degree m is denoted by $N_\alpha TRSSNP_m^{csyn}$, where $\alpha \in \{gen, acc\}$ indicates the systems act as number generating (resp., accepting) devices, and m can be replaced by $*$ when the degree is not specified.

3 Computation Power of Generating Numbers

In this section, the universal results of timed RSSNP systems with synaptic co-transmission are generated by simulating universal register machines, when systems are used to generate numbers.

Formally, an universal register machine with m registers storing non-negative numbers is of the form $M = (m, H, I)$, where $H = \{l_0, l_1, \dots, l_h\}$ is the set of instruction labels with the starting label l_0 and the halting label l_h , respectively; I is the finite set of instructions with unique labels from H , where each element of H is associated with an unique label in I . The details of labeled instructions are of the following forms:

- ADD instructions of the form $l_i : (ADD(p), l_j, l_k)$; the system nondeterministically goes to either the instruction labeled l_j or l_k after an accumulation of one to register r ;
- SUB instructions of the form $l_i : (SUB(p), l_j, l_k)$; the system goes to instruction labeled l_j after a consumption of one from register p if register p is nonempty, or directly go to instruction labeled l_k otherwise;
- $l_h : (HALT)$; the system halts when it goes to the halting instruction label l_h .

Theorem 1. $N_{gen}TRSSNP_*^{csyn} = NRE$.

Proof. Let $M = (m, H, I)$ be a register machine defined above. In the following manner, machine M generate number n : it starts from instruction labeled l_0 with all registers storing value 0; proceeds according to the values in registers and the indicators; machine M eventually reaches instruction labeled l_h when the number n stored in the first register. As usual, the initial instruction l_0 is an ADD instruction, and register 1 as the output register participates no operations in any SUB instruction. The proof of the Theorem 1 is based on the fact that non-deterministic register machine M can generate the family NRE [21], and the convention that number zero is ignored when one compares the computational power of two computing devices [13].

To simulate register machine M mentioned above, a timed RSSNP system with synaptic co-transmission $\Pi(e)$ is constructed, where each register p ($p = 1, 2, \dots, m$) is associated with an unique neuron σ_p , and the value n_p in the register p is associated with $2n_p$ spikes in neuron σ_p . Moreover, each ADD (resp., SUB) instruction is associated with an ADD (resp., SUB) module shown in Fig. 1 (resp., shown in Fig. 2), and the halt instruction is associated with the Halt module shown in Fig. 3.

In each module, the neurons $\sigma_{l_i}, \sigma_{l_j}, \sigma_{l_k}$ ($0 \leq i, j, k \leq h$) are associated with the instructions labeled l_i, l_j, l_k , respectively. The simulation of instructions labeled l_i begins when neuron σ_{l_i} contains exact one spike, and the neurons of the form $\sigma_{l_i^j}$ in the same module are auxiliary neurons. In what follows, the simulation of each type instruction is clearly illustrated by corresponding modules.

Simulation of ADD instruction.

The ADD module in Fig. 1 consisting of three sets of co-transmission synapses is constructed to simulate ADD instruction $l_i : (ADD(p), l_j, l_k)$.

Assume that a spike is delivered to neuron σ_{l_i} in an ADD module at step t , so two rules on synapses $((l_i, l_i^1), r_{i_1} : a \rightarrow a), ((l_i, l_i^2), r_{i_2} : a \rightarrow a)$ ($r_{i_1}, r_{i_2} \in R(S_{i_1}')$) emerging from neuron σ_{l_i} are enabled. The application of the enabled rules means that one spike produced by σ_{l_i} is distributed to each of the neurons $\sigma_{l_i^1}, \sigma_{l_i^2}$, and all of the rules on synapses $((l_i^1, l_i^3), r_{i_3} : a \rightarrow a), ((l_i^2, l_i^3), r_{i_4} : a \rightarrow a), ((l_i^1, p), r_{i_5} : a \rightarrow a), ((l_i^2, p), r_{i_6} : a \rightarrow a)$ ($r_{i_3}, r_{i_4}, r_{i_5}, r_{i_6} \in R(S_{i_2}')$) are enabled at step $t + e(r_{i_1})$. Hence, two spikes are distributed to each of the neurons σ_p and $\sigma_{l_i^3}$ at step $t + e(r_{i_1}) + e(r_{i_3})$, by using each of the rules in the set $R(S_{i_2}')$. Then, one of the rule on synapse (l_i^3, l_i^4) or the rule on synapse (l_i^3, l_k) is non-deterministically chosen as they take different types of spike consumption numbers.

If the rule on synapse $((l_i^3, l_k), r_{i_7} : a^2 \rightarrow a)$ is chosen to be applied, one spike from neuron $\sigma_{l_i^3}$ is delivered to σ_k at the end of step $t + e(r_{i_1}) + e(r_{i_3}) + e(r_{i_7})$.

If the rule on synapse $((l_i^3, l_i^4), r_{i_8} : a^2/a \rightarrow a)$ is chosen to be applied, one spike is delivered to neuron $\sigma_{l_i^4}$ at the end of step $t + e(r_{i_1}) + e(r_{i_3}) + e(r_{i_8})$. Then, the rules on synapse $((l_i^4, l_j), r_{i_9} : a \rightarrow a)$ and $((l_i^3, g), r_{i_{10}} : a \rightarrow a)$ are applied in the same execution time $e(r_{i_9})$ as they are in the same set $R(S_{i_3}')$ with one spike distributed to neuron σ_j .

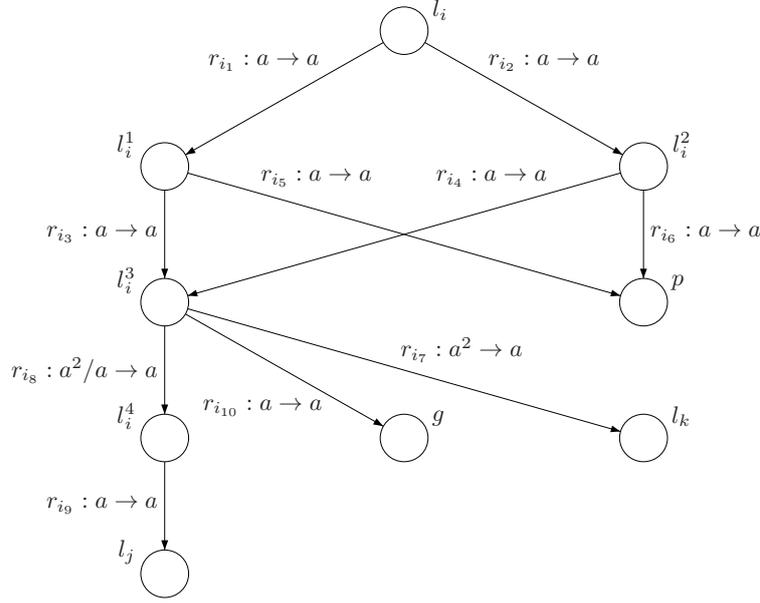


Fig. 1: ADD-Module with three sets of co-transmission synapses: $S_{i'_1} = \{(l_i, l_i^1), (l_i, l_i^2)\}$, $S_{i'_2} = \{(l_i^1, l_i^3), (l_i^1, p), (l_i^2, l_i^3), (l_i^2, p)\}$, $S_{i'_3} = \{(l_i^4, l_j), (l_i^3, g)\}$.

The activation of the ADD module provides two spikes to σ_p , and one spike to either σ_{l_j} or σ_{l_k} , nondeterministically. The simulation of the ADD instruction $l_i : (ADD(p), l_j, l_k)$ is correct.

Simulation of SUB instruction.

Let B_p be the set of SUB instruction labels associated with register p ($1 \leq p \leq m$), and $|B_p|$ be the number of labels in the set. For instance, $|B_1| = 0$ as register 1 participates no SUB instruction.

The SUB module in Fig. 2 consisting of five sets of co-transmission synapses is constructed to simulate SUB instruction $l_i : (SUB(p), l_j, l_k)$.

Assume that a spike is delivered to neuron σ_{l_i} at step t signifying the starting of the simulation of the SUB instruction $l_i : (SUB(p), l_j, l_k)$. All the rules on synapses $((l_i, p), r_{i_1} : a \rightarrow a)$, $((l_i, l_i^1), r_{i_2} : a \rightarrow a)$ ($r_{i_1}, r_{i_2} \in R(S_{i'_1})$) are applied, and both of the neurons σ_p and $\sigma_{l_i^1}$ receive one spike at step $t + e(r_{i_1})$.

If $2n_p > 0$ spikes are contained in neuron σ_p , all the rules on synapses $((p, l_i^2), r_{i_3} : a(aa)^+/a^3 \rightarrow a)$, $((l_i^1, l_i^2), r_{i_5} : a \rightarrow a)$, $((l_i^1, l_i^3), r_{i_6} : a \rightarrow a)$ in the set $R(S_{i'_2})$ are enabled at step $t + e(r_{i_1})$. By applying the enabled rules, two spikes are delivered to neuron $\sigma_{l_i^3}$ and one spike to neuron $\sigma_{l_i^2}$ at step $t + e(r_{i_1}) + e(r_{i_3})$; besides, one spike is delivered to each of neurons $\sigma_{l_u^2}$, $\sigma_{l_u^3}$ as register p is shared by SUB instruction $l_u : (SUB(p), l_v, l_w)$. At step $t + e(r_{i_1}) + e(r_{i_3}) + e(r_{i_7})$, the application of all the rules in $R(S_{i'_4})$ is completed. One spike is delivered to

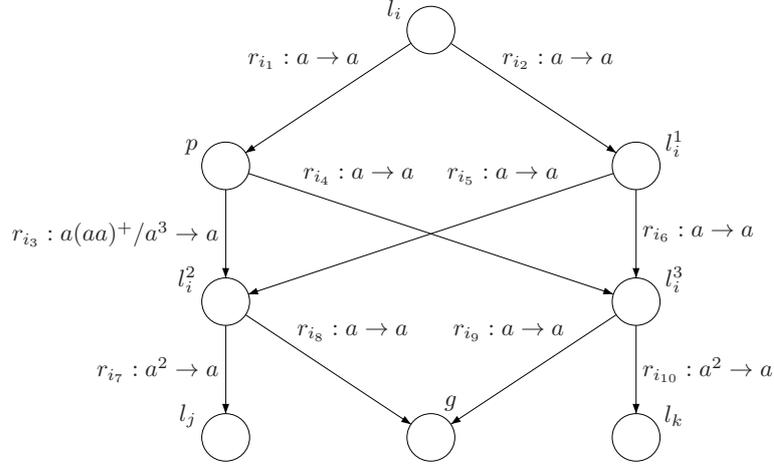


Fig. 2: SUB-Module with five sets of co-transmission synapses: $S_{i'_1} = \{(l_i, p), (l_i, l_i^1)\}$, $S_{i'_2} = \{(p, l_i^2), (l_i^1, l_i^2), (l_i^1, l_i^3)\}$, $S_{i'_3} = \{(p, l_i^3), (l_i^1, l_i^2), (l_i^1, l_i^3)\}$, $S_{i'_4} = \{(l_i^2, l_j), (l_i^3, g), (l_i^2, g), (l_i^3, g), \dots\}$, $S_{i'_5} = \{(l_i^3, l_k), (l_i^2, g), (l_i^2, g), (l_i^3, g), \dots\}$, with $l_u \in B_p$.

neuron σ_{l_j} , and all the unnecessary spikes, such as spikes in the neuron $\sigma_{l_i^2}$, $\sigma_{l_i^3}$, $\sigma_{l_i^3}$, are removed across synapses in $S_{i'_4}$ causing no interference.

If no spike is contained neuron σ_p , all the rules on synapses $((p, l_i^3), r_{i_4}: a \rightarrow a)$, $((l_i^1, l_i^2), r_{i_5}: a \rightarrow a)$, $((l_i^1, l_i^3), r_{i_6}: a \rightarrow a)$ in the set $R(S_{i'_3})$ are enabled at step $t + e(r_{i_1})$, and the execution of the rules are completed at step $t + e(r_{i_1}) + e(r_{i_4})$. Then, all the rules in the set $R(S_{i'_5})$ can be applied during the next $e(r_{i_8})$ steps. At step $t + e(r_{i_1}) + e(r_{i_4}) + e(r_{i_8})$, one spike is delivered to neuron σ_{l_k} , and all the unnecessary spikes are removed deleting the undesirable interference.

By the work of SUB module, the number of spikes in neuron σ_p which is non-empty is decreased by two, and a spike is delivered to neuron σ_{l_j} ; or a spike is delivered to neuron σ_{l_k} keeping neuron σ_p empty. The simulation of the SUB instruction $l_i: (SUB(p), l_j, l_k)$ is correctly completed by the corresponding SUB module.

Simulation of Halt instruction.

The Halt Module shown in Fig. 3 with no set of co-transmission synapses, corresponding to the halting instruction $l_h: HALT$, is designed for halting the system and outputting the computation results.

If a spike is delivered to neuron σ_{l_h} at step t when $2n_1$ spikes are contained in the output neuron σ_1 , the rule on synapse $((l_h, 1), r_{h_1}: a \rightarrow a)$ can be applied during the next $e(r_{h_1})$ steps with another one spike delivered to neuron σ_1 . From step $t + e(r_{h_1}) + 1$ to step $t + e(r_{h_1}) + n_1 e(r_{h_2})$, the rule on synapse $((1, env), r_{h_2}: a(aa)^+/a^2 \rightarrow a)$ is applied for n_1 times emitting n_1 spikes to the environment. Then, the system halts with no rules to be used, and the number

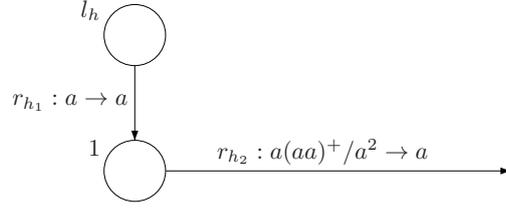


Fig. 3: HALT-Module.

of spikes emitted out by output neuron σ_1 equals to the number generated by register machine M .

From the work of $\Pi(e)$, $N(M) = N(\Pi(e))$. \square

Theorem 2. *Time-free RSSNP systems with synaptic co-transmission working as number generating devices are Turing universal.*

Proof. Let $\Pi(e') = (O, \sigma_1, \sigma_2, \dots, \sigma_m, \text{syn}, S, R, e', 1)$ be an arbitrary timed RSSNP system with synaptic co-transmission, where the topology remains the same with the system $\Pi(e)$ in Theorem 1, and the mapping e' is different from the mapping e in $\Pi(e)$. From the analysis of Theorem 1, $N(M) = N(\Pi(e'))$, and hence the proof is completed. \square

4 Computation Power of Accepting Numbers

In this section, it is proved that RSSNP system with synaptic co-transmission can accept any recursively enumerable set of numbers, and an universal timed RSSNP system with synaptic co-transmission is constructed to accept numbers.

Theorem 3. $N_{acc}TRSSNP_*^{csyn} = NRE$.

Proof. It is known that register machines can accept any recursively enumerable set of numbers in the following manner [16]. Natural numbers x is introduced in the first register (such as register 1); then, the register machine starts a computation executing the instruction l_0 , and continues to execute deterministic ADD instructions of the form $l_i : (ADD(p), l_j)$ and SUB instructions mentioned above as indicated by the labels; if the register machine halts at instruction l_h , the number x is accepted by the register machine.

Let $M_a = (m, H, I)$ be such a register machine mentioned above. To simulate machine M_a , the timed RSSNP system with synaptic co-transmission $\Pi_a(e)$ is constructed, where the IN module (in Fig. 4) is used to introduce the number x ; ADD modules (in Fig. 5) are used to carry out the simulation of deterministic ADD instructions; SUB modules are the same with those in Theorem 1; the neuron σ_{l_h} has the synapse (l_h, env) with the rule $a \rightarrow a$ on the synapse. The work of IN module helps neuron σ_1 accumulate $2x$ spikes, and then helps neuron σ_{l_0} contain one spike by which the system starts the simulation of acceptance.

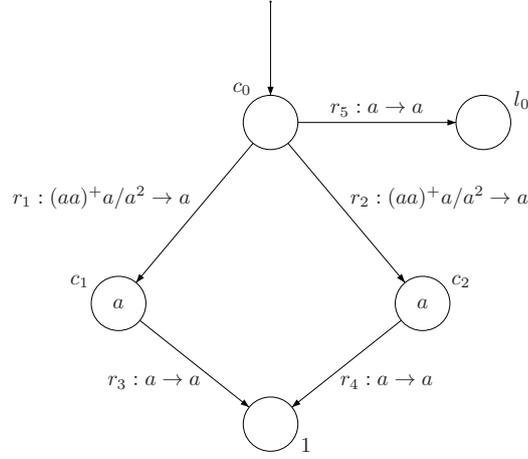


Fig. 4: IN-Module with three sets of co-transmission synapses: $S_1 = \{(c_0, c_1), (c_0, c_2), (c_1, 1), (c_2, 1)\}$.

Initially, each of the neurons $\sigma_{c_1}, \sigma_{c_2}$ contains a spike, while all the other neurons are empty. The enabled rules on synapses $((c_1, 1), r_3 : a^+/a \rightarrow a)$, $((c_2, 1), r_4 : a^+/a \rightarrow a)$ can not be applied because there is not any enabled rule on the synapse $(c_1, 1)$ or the synapse $(c_2, 1)$ which are in the same set of synapse co-transmission S_1 . The system $\Pi_a(e)$ begins to introduce the number x when neuron σ_{c_0} receives the temporal spikes $0^y 32^{x-1} 0^z$ ($y, z \geq 1$): the moment when neuron σ_{c_0} receives two spikes (resp., three spikes) is encoded by number 2 (resp., 3), and the moment when the neuron σ_{c_0} receives nothing is encoded by number 0).

When three spikes are delivered to the neuron σ_{c_0} at step 1, the enabled rules on synapse $((c_0, c_1), r_1 : (aa)^+a/a^2 \rightarrow a)$, $((c_0, c_2), r_2 : (aa)^+a/a^2 \rightarrow a)$, $((c_1, 1), r_3 : a^+/a \rightarrow a)$, $((c_2, 1), r_4 : a^+/a \rightarrow a)$ are applied during the first $e(r_1)$ steps. At the end of step $e(r_1)$ ($e(r_1) \leq 1$), two spikes are accumulated in neuron σ_1 , and a spike is delivered to each of the neurons $\sigma_{c_1}, \sigma_{c_2}$ by which the rules r_3, r_4 are enabled again. The rules $r_1, r_2 \in R(S_1)$ can not be applied before the rules r_3, r_4 are enabled, though the rules $r_1, r_2 \in R(S_1)$ are enabled at step 2 when two spikes are delivered to neuron σ_{c_0} . Because of the restriction of the set of synapse co-transmission S_1 , the rules in $R(S_1)$ can be concurrently applied for x times. At the end of step $x * e(r_1)$, $2x$ spikes are accumulated in neuron σ_1 , and one spike is left in each of the neurons $\sigma_{c_0}, \sigma_{c_1}, \sigma_{c_2}$. Although the rules r_3, r_4 are enabled at step $x * e(r_1) + 1$, they can not be applied for the rules r_1, r_2 which are in the same set $R(S_1)$ are not enabled. From step $x * e(r_1) + 1$ to step $x * e(r_1) + e(r_5)$, the rule on synapse $((c_0, l_0), r_5 : a \rightarrow a)$ is applied, and a spike is delivered to neuron σ_{l_0} indicating the starting of the following computation.

The ADD module for deterministic ADD instruction $l_i : (ADD(p), l_j)$ is shown in Fig.5. When a spike is delivered to neuron σ_{l_i} at step t , the rules

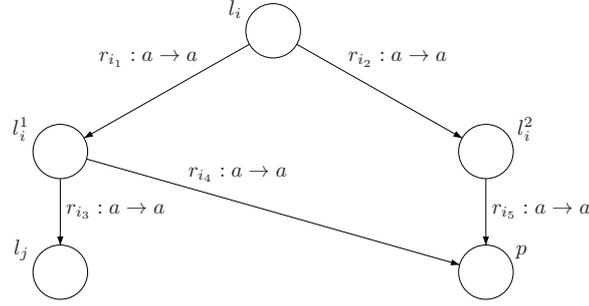


Fig. 5: ADD-Module with two sets of co-transmission synapses $S_{i_1} = \{(l_i, l_i^1), (l_i, l_i^2)\}$, $S_{i_2} = \{(l_i^1, p), (l_i^2, p), (l_i^1, l_j)\}$.

on synapses $((l_i, l_i^1), r_{i_1} : a \rightarrow a)$, $((l_i, l_i^2), r_{i_2} : a \rightarrow a)$ ($r_{i_1}, r_{i_2} \in R(S_{i_1})$) are enabled, and the application of the enabled rules means that a spike is delivered to each of the neurons $\sigma_{l_i^1}$ and $\sigma_{l_i^2}$ with the rules on synapses $((l_i^1, l_j), r_{i_3} : a \rightarrow a)$, $((l_i^1, p), r_{i_4} : a \rightarrow a)$, $((l_i^2, p), r_{i_5} : a \rightarrow a)$ ($r_{i_3}, r_{i_4}, r_{i_5} \in R(S_{i_2})$) enabled at step $t + e(r_{i_3})$. Then, the three enabled rules are applied, and two spikes are delivered to neuron σ_p and one spike to neuron σ_{l_j} . It is clear that the simulation of $l_i : (ADD(p), l_j)$ is correct.

The simulation of SUB modules which are the same with that in Theorem 1 has been verified.

If the rule on synapse $((l_h, env), a \rightarrow a)$ is enabled when a spike is delivered to neuron σ_{l_h} at some time, one spike is emitted to the environment after the application of the rule on synapse $((l_h, env), a \rightarrow a)$ is completed. Then, system halts with no rule to be applied signifying that the number x is accepted. \square

We have the following Theorem 4 by expanding Theorem 3.

Theorem 4. *Time-free RSSNP systems with synaptic co-transmission working as accepting number devices are universal.*

An universal time-free RSSNP systems with synaptic co-transmission is constructed on the basis of Theorem 4 to accept numbers.

Corollary 1. $N_{acc}ARSSNP_{101}^{csyn} = NRE$.

Proof. The well-known universal register machine M_c with 8 registers and 23 instructions is given in Fig. 6. The register 0 storing the computation results in M_c is associated with a SUB instruction $(l_{19} : (SUB(0), l_0, l_{18}))$ which breaks away from the assumption (used in the Theorem 3) that output register associated with no SUB instruction, so some modifications are made on here.

A complementary register labeled 8 which is associated with no SUB instruction is added in M_c ; two complementary instructions $l_{-1} : (ADD(2), l_{-1}, l_{-2})$, $l_{-2} : (ADD(8), l_0)$ are added; the instruction $l_{18} : (SUB(4), l_0, l_h)$ is substituted for $l_{18} : (SUB(4), l_0, l_{22})$.

$l_0 : (SUB(1), l_1, l_2)$	$l_1 : (ADD(7), l_0)$
$l_2 : (ADD(6), l_3)$	$l_3 : (SUB(5), l_2, l_4)$
$l_4 : (SUB(6), l_5, l_3)$	$l_5 : (ADD(5), l_6)$
$l_6 : (SUB(7), l_7, l_8)$	$l_7 : (ADD(1), l_4)$
$l_8 : (SUB(6), l_9, l_6)$	$l_9 : (ADD(6), l_{10})$
$l_{10} : (SUB(4), l_0, l_{11})$	$l_{11} : (SUB(5), l_{12}, l_{13})$
$l_{12} : (SUB(1), l_1, l_2)$	$l_{13} : (SUB(2), l_{18}, l_{19})$
$l_{14} : (SUB(5), l_{16}, l_{17})$	$l_{15} : (SUB(3), l_{18}, l_{20})$
$l_{16} : (ADD(4), l_{11})$	$l_{17} : (ADD(2), l_{21})$
$l_{18} : (SUB(4), l_0, l_h)$	$l_{19} : (SUB(0), l_0, l_{18})$
$l_{20} : (ADD(0), l_0)$	$l_{21} : (ADD(3), l_{18})$
$l_h : HALT$	

Fig. 6: Universal register machine with 8 registers and 23 instructions.

The modified universal register machine M_s takes register 8 as the output register associated with no SUB instruction. Machine M_s consists of nine registers, ten deterministic ADD instructions, a non-deterministic ADD instruction, thirteen SUB instructions, and a Halt instruction. An time-free RSSNP with synaptic co-transmission $\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn, S, R, e', l_h)$ is designed to simulate machine M_s .

- 9 neurons for the registers,
- 3 neurons and a set of co-transmission synapses for the inputs,
- 1 garbage neuron for removing unnecessary spikes,
- 25 neurons for the instruction labels,
- $2 \times 10 + 4$ auxiliary neurons and $2 \times 10 + 3$ sets of co-transmission synapses for the ADD instructions,
- 3×13 auxiliary neurons and 5×13 sets of co-transmission synapses for the SUB instructions.

The universal time-free RSSNP systems with synaptic co-transmission is composed of 101 neurons and 89 sets of co-transmission synapses to accept numbers. \square

5 Discussions and Conclusions

In general, the biological phenomenon that neurotransmitters are transmitted independently is incorporated in variants of spiking neural P systems. Here, the idea of co-transmission is introduced in RSSNP systems (a variant of spiking neural P systems) named timed RSSNP systems with synaptic co-transmission, where a family of co-transmission synapses sets is given; a spike is delivered

across a synapse that in the co-transmission synapses set if and only if one spike is delivered across each of the synapses which are in the same set.

In this work, the computational power of timed RSSNP systems with synaptic co-transmission is investigated. Timed RSSNP systems with synaptic co-transmission are proven to be universal for generating numbers (Theorem 1) as well as accepting numbers (Theorem 3), using only standard firing rules without forgetting rules or delay time. In addition, the universal results can be extended to time-free RSSNP systems with synaptic co-transmission (Theorem 2, Theorem 4), and an universal timed RSSNP systems with synaptic co-transmission is constructed to accept numbers (Corollary 1). The results demonstrate that synaptic co-transmission is such an element that make the computing power of the timed RSSNP systems robust.

Many new computing models have no physical realization, and one reason is that small changes in the implementation could affect the ability to compute or solve a problem. Here, synaptic co-transmission is used to make the computation power of timed RSSNP systems robust, and hence it can be used to design and implementation of the corresponding simulator mentioned in [43], and to efficiently solve some application problems (such as robot controller design, fault diagnosis of power systems [3, 9, 12]) which can be converted to computations in timed RSSNP system with synaptic co-transmission.

The universal timed RSSNP system with synaptic co-transmission (in Corollary 1) could be further investigated, such as constructing homogeneously universal systems like [48], minimizing the number of neurons (resp., synapses, rules) of the universal systems [22, 29, 36], restricting the numbers of synapses in sets of synapse co-transmission that are in universal systems, searching the normal form of universal time-free RSSNP systems [24, 38], and so forth. The role of synaptic co-transmission could be further investigated, such as investigating the computation power of RSSNP systems (or other computing models in membrane systems, e.g., tissue P systems [33, 34]) with synaptic co-transmission working in other mode (e.g., the clock-free mode mentioned in [2]).

Acknowledgements

The work is supported by National Natural Science Foundation of China (62072201) and China Postdoctoral Science Foundation (2020M672359).

References

1. Adorna, H.N.: Computing with SN P systems with I/O mode. *Journal of Membrane Computing* **2**(4), 230–245 (2020)
2. Alhazov, A., Freund, R., Ivanov, S., Pan, L., Song, B.: Time-freeness and clock-freeness and related concepts in P systems. *Theoretical Computer Science* **805**, 127–143 (2020)
3. Buiu, C., Florea, A.G.: Membrane computing models and robot controller design, current results and challenges. *Journal of Membrane Computing* **1**(4), 262–269 (2019)

4. Cabarle, F.G.C., Adorna, H.N., Jiang, M., Zeng, X.: Spiking neural P systems with scheduled synapses. *IEEE Transactions on Nanobioscience* **16**(8), 792–801 (2017)
5. Cabarle, F.G.C., de la Cruz, R.T.A., Cailipan, D.P.P., Zhang, D., Liu, X., Zeng, X.: On solutions and representations of spiking neural P systems with rules on synapses. *Information Sciences* **501**, 30–49 (2019)
6. Cavaliere, M., Sburlan, D.: Time-independent P systems. In: *International Workshop on Membrane Computing*. pp. 239–258. Springer (2004)
7. Cavaliere, M., Sburlan, D.: Time and synchronization in membrane systems. *Fundamenta Informaticae* **64**(1-4), 65–77 (2005)
8. Ciobanu, G., Pérez-Jiménez, M.J., Păun, Gh.: *Applications of Membrane Computing*. Berlin:Springer (2006)
9. Fan, S., Paul, P., Wu, T., Rong, H., Zhang, G.: On applications of spiking neural P systems. *Applied Sciences* **10**(20), 7011 (2020)
10. Ghosh-Dastidar, S., Adeli, H.: Spiking neural networks. *International Journal of Neural Systems* **19**(04), 295–308 (2009)
11. Hopfield, J.J.: Artificial neural networks. *IEEE Circuits and Devices Magazine* **4**(5), 3–10 (1988)
12. Huang, Y., Wang, T., Wang, J., Peng, H.: Reliability evaluation of distribution network based on fuzzy spiking neural P system with self-synapse. *Journal of Membrane Computing* **3**(1), 51–62 (2021)
13. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. *Fundamenta Informaticae* **71**(2, 3), 279–308 (2006)
14. Jiang, Y., Su, Y., Luo, F.: An improved universal spiking neural P system with generalized use of rules. *Journal of Membrane Computing* **1**(4), 270–278 (2019)
15. Koch, C., Segev, I.: The role of single neurons in information processing. *Nature Neuroscience* **3**(11), 1171–1177 (2000)
16. Korec, I.: Small universal register machines. *Theoretical Computer Science* **168**(2), 267–301 (1996)
17. Lehotzky, A., Oláh, J., Fekete, J.T., Szénási, T., Szabó, E., Gyórfy, B., Várady, G., Ovadi, J.: Co-transmission of alpha-synuclein and TPPP/p25 inhibits their proteolytic degradation in human cell models. *Frontiers in molecular biosciences* **8**, 421 (2021)
18. Luo, Y., Guo, P., Jiang, Y., Zhang, Y.: Timed homeostasis tissue-like P systems with evolutionary symport/antiport rules. *IEEE Access* **8**, 131414–131424 (2020)
19. Luo, Y., Tan, H., Zhang, Y., Jiang, Y.: The computational power of timed P systems with active membranes using promoters. *Mathematical Structures in Computer Science* **29**(5), 663–680 (2019)
20. Maher, B.J., Westbrook, G.L.: Co-transmission of dopamine and GABA in periglomerular cells. *Journal of neurophysiology* **99**(3), 1559–1564 (2008)
21. Minsky, M.L.: *Finite and Infinite Machines*. Englewood Cliffs: Prentice-Hall (1967)
22. Neary, T.: Three small universal spiking neural P systems. *Theoretical Computer Science* **567**, 2–20 (2015)
23. Pan, L., Păun, Gh.: Spiking neural P systems with anti-spikes. *International Journal of Computers Communications & control* **4**(3), 273–282 (2009)
24. Pan, L., Păun, Gh.: Spiking neural P systems: an improved normal form. *Theoretical Computer Science* **411**(6), 906–918 (2010)
25. Pan, L., Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems with neuron division and budding. *Science China Information Sciences* **54**(8), 1596 (2011)
26. Pan, L., Păun, Gh., Zhang, G., Neri, F.: Spiking neural P systems with communication on request. *International Journal of Neural Systems* **27**(08), 1750042 (2017)

27. Pan, L., Zeng, X., Zhang, X.: Time-free spiking neural P systems. *Neural Computation* **23**(5), 1320–1342 (2011)
28. Pan, L., Zeng, X., Zhang, X., Jiang, Y.: Spiking neural P systems with weighted synapses. *Neural Processing Letters* **35**(1), 13–27 (2012)
29. Păun, A., Păun, Gh.: Small universal spiking neural P systems. *BioSystems* **90**(1), 48–60 (2007)
30. Păun, Gh.: *Membrane Computing: An Introduction*. Springer Science & Business Media (2012)
31. Rozenberg, G., Salomaa, A.: *Handbook of formal languages: volume 3 beyond words*. Springer Science & Business Media (2012)
32. Song, B., Li, K., Orellana-Martín, D., Pérez-Jiménez, M.J., Pérez-Hurtado, I.: A survey of nature-inspired computing: membrane computing. *ACM Computing Surveys* **54**(1), 1–31 (2021)
33. Song, B., Li, K., Orellana-Martín, D., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: Cell-like P systems with evolutionary symport/antiport rules and membrane creation. *Information and Computation* **275**, 104542 (2020)
34. Song, B., Li, K., Zeng, X.: Monodirectional evolutionary symport tissue P systems with promoters and cell division. *IEEE Transactions on Parallel and Distributed Systems*, DOI: 10.1109/TPDS.2021.3065397 (2021)
35. Song, B., Song, T., Pan, L.: A time-free uniform solution to subset sum problem by tissue P systems with cell division. *Mathematical Structures in Computer Science* **27**(1), 17–32 (2017)
36. Song, B., Zeng, X., Jiang, M., Pérez-Jiménez, M.J.: Monodirectional tissue P systems with promoters. *IEEE Transactions on Cybernetics* **51**(1), 438–450 (2021)
37. Song, B., Pan, L.: Computational efficiency and universality of timed p systems with active membranes. *Theoretical Computer Science* **567**, 74–86 (2015)
38. Song, T., Pan, L.: A normal form of spiking neural P systems with structural plasticity. *International Journal of Swarm Intelligence* **1**(4), 344–357 (2015)
39. Song, T., Pan, L.: Spiking neural P systems with rules on synapses working in maximum spiking strategy. *IEEE Transactions on Nanobioscience* **14**(4), 465–477 (2015)
40. Song, T., Pan, L., Păun, Gh.: Spiking neural P systems with rules on synapses. *Theoretical Computer Science* **529**, 82–95 (2014)
41. Song, T., Xu, J., Pan, L.: On the universality and non-universality of spiking neural P systems with rules on synapses. *IEEE Transactions on NanoBioscience* **14**(8), 960–966 (2015)
42. Song, T., Zou, Q., Liu, X., Zeng, X.: Asynchronous spiking neural P systems with rules on synapses. *Neurocomputing* **151**, 1439–1445 (2015)
43. Valencia-Cabrera, L., Pérez-Hurtado, I., Martínez-del Amor, M.Á.: Simulation challenges in membrane computing. *Journal of Membrane Computing* pp. 1–11 (2020)
44. Wang, J., Hoogeboom, H.J., Pan, L., Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems with weights. *Neural Computation* **22**(10), 2615–2646 (2010)
45. Wijekoon, J.H.B., Dudek, P.: Compact silicon neuron circuit with spiking and bursting behaviour. *Neural Networks* **21**(2-3), 524–534 (2008)
46. Wu, T., Bîlbîe, F., Păun, A., Pan, L., Neri, F.: Simplified and yet turing universal spiking neural P systems with communication on request. *International Journal of Neural Systems* **28**(08), 1850013 (2018)
47. Wu, T., Păun, A., Zhang, Z., Pan, L.: Spiking neural P systems with polarizations. *IEEE Transactions on Neural Networks and Learning Systems* **29**(8), 3349–3360 (2017)

48. Zeng, X., Zhang, X., Pan, L.: Homogeneous spiking neural P systems. *Fundamenta Informaticae* **97**(1-2), 275–294 (2009)
49. Zhang, G., Pérez-Jiménez, M.J., Gheorghe, M.: *Real-life Applications with Membrane Computing*. Berlin:Springer (2017)
50. Zhang, G., Rong, H., Neri, F., Pérez-Jiménez, M.J.: An optimization spiking neural P system for approximately solving combinatorial optimization problems. *International Journal of Neural Systems* **24**(05), 1440006 (2014)
51. Zhang, X., Pan, L., Păun, A.: On the universality of axon P systems. *IEEE Transactions on Neural Networks and Learning Systems* **26**(11), 2816–2829 (2015)

Regular Papers: Nature-Inspired Computing

Face Illumination Normalization based on Generative Adversarial Network^{*}

Dequan Guo¹, Shenggui Ling^{2,3}, Tianxiang Li⁴, and Haoyuan Ma¹

¹ Chengdu University of Information Technology, Chengdu, Sichuan, 610225, China
guodq@cuit.edu.cn; 2101555266@qq.com

² National Key Laboratory of Fundamental Science on Synthetic Vision, Sichuan University, Chengdu, Sichuan, 610065, China

³ Neijiang Vocational and Technical College, Neijiang, Sichuan, 641000, China
Shenggui Ling, 2017326040006@stu.scu.edu.cn

⁴ Sichuan Changjiang Vocational College, Chengdu, Sichuan, 610106, China
litianxiang@ultrawise.com.cn

Abstract. Face recognition technology has been widely used in the field of artificial intelligence. The technology needs to be carried out normally under the appropriate light, however, there is not ideal light, even poor-lighted for the face recognition device, and with the head in deflect angle. The poor-lighted under various head poses will influence the face recognition significantly. To address the issue, we present a novel and practical architecture based on deep fully convolutional neural network and generative adversarial networks for illumination normalization of facial images. The proposed method is termed as illumination normalization generative adversarial network. Compared with previous methods based on deep learning, our approach does not require identity and illumination label as input. We preserve identities of faces by an elaborately-designed generator together with content loss. Moreover, the framework of our scheme is simpler than previous methods based on deep learning. It can address the illumination of frontal and non-frontal face. In order to fairly evaluate the proposed method against state-of-the-art models, the peak signal to noise ratio is employed to estimate the performance of illumination normalization algorithm. Experimental results show that the proposed method achieves favorable normalization results against previous models under various head poses and illumination challenges.

Keywords: Face recognition · illumination normalization · generative adversarial networks · loss function.

^{*} We would like to thank Dr. Ferrante Neri (University of Nottingham) and Dr. Paul Liu (Stork Healthcare) to give us some advice on scientific research. This work was supported in part by the National Natural Science Foundation of China under Grant 61806028, Grant 61672437 and Grant 61702428, Sichuan Science and Technology Program under Grant 2018GZ0245, Grant 21ZDYF2484, Grant 2021YFN0104, Grant 18ZDYF3269, Grant 21GJHZ0061, Grant 21ZDYF2907, Grant 21ZDYF3629, Grant 21ZDYF0418, Grant 21YYJC1827, Grant 21ZDYF3537, Grant 2019YJ0356, and the Chinese Scholarship Council under Grant 202008510036.

1 Introduction

It is well known that illumination is an important factor to perform computer vision tasks. As is shown in Fig. 1, some reasons, such as the excessive exposure and the lack of exposure of the camera, the intensity and direction of the light, could make the lighting conditions complicated. Face appearances can change dramatically due to illumination variations. This fact will cause: the variations between the images of the same face since illumination are almost always larger than image variations due to changes in face identities. Zhang et al. [1] proposed a recursive copy and paste Generative Adversarial Network (Re-CPGAN) to recover authentic high-resolution face images while compensating for non-uniform illumination. A depth algorithm was proposed for a single-lens occluded portrait to estimate the actual portrait distance for different poses, angles of view and obscuration [2]. A network analysis approach was carried out, depicting sub-communities of nodes related to face [3]. There are some challenges in illumination [4]. Therefore, illumination normalization of face is essential and valuable. In this study, we focus on the task of illumination normalization of facial images under different illumination conditions and a variety of head poses.



Fig. 1. Some poor-lighted faces under various head poses.

As a pioneering work, Faisal et al. [5] combine Phongs lighting model and a 3D face model to normalize illumination of color face. Unfortunately, due to the requirement for 3D point clouds and a large amount of computation, this method has limited practical application. With the developments of hardware and neural networks, illumination normalization is gradually evolved from traditional ways to deep learning-based techniques. So far, there are a few methods to process illumination with deep learning, for which there remain two challenges: 1) The key challenge is identity preservation. 2) It is difficult to deal with the illumination of color face. Ma et al. [6] first used generative adversarial nets

to process illumination of facial images. Han et al. [7] put forward asymmetric joint generative adversarial networks (GAN) to normalize facial illumination with lighting labels. Therefore, in view of the shortcomings of the methods, we propose a new method to normalize illumination of color facial image without any identity label and illumination label as input. Moreover, our method that contains a generator, a discriminator and one feature extractor is simpler than the previous methods based on deep learning.

Inspired by the success of GANs on image denoising, image synthesis and transfer learning, we reformulate the illumination normalization problem like the way of dealing with the tasks above. Our goal is to learn a GAN mapping from any poor-lighted images to well-lighted images, and the latter is called standard illumination cases in this study. In summary, our main contributions are listed as follows:

1. A new scheme is proposed for the illumination normalization of color face images. Unlike previous deep learning methods, we reduce the number of discriminators and not utilize reconstruction computation, which improves the computational speed.
2. We use content loss and elaborately designed generator for preserving identity. Experimental results demonstrate that the combination of content and L1 loss makes our method achieve good performance. The proposed method can not only process the illumination of frontal face but also the non-frontal face.
3. Though our model is trained on the faces under well-controlled lighting variations, it generalizes to face images with less-controlled lighting variations well, meanwhile preserving its identity effectively.

The reminder of the paper is organized as follows. Section 2 describes related work on illumination normalization. Section 3 describes the proposed Illumination Normalization GAN (IN-GAN) in details. Experimental results, evaluation and comparisons are included in Section 4. Section 5 describes validation for identity preserving. Finally, conclusions are drawn in Section 6.

2 Related work

2.1 Illumination Normalization

2.1.1 Traditional methods To deal with the illumination variation problem, numerous works have been put over the past decades. In 1987, Pizer et al. [8] proposed adaptive histogram equalization to enhance image contrast. Afterward, many researchers extended the histogram equalization algorithm. For instance, Shan et al. [9] propose region-based histogram equalization to deal with illumination. Xie et al. [10] proposed block-based histogram equalization for illumination processing. Orientated local histogram equalization that compensates illumination while encoding rich information on the edge orientations is presented by Lee et al. [11]. In 1999, Shashua and Riklin-Raviv proposed the

quotient image method [12] that provided an invariant approach to deal with the illumination variation. Afterward, many researchers extended the quotient image algorithm. Shan et al. [9] developed gamma intensity correction for normalizing the overall image intensity at the given illumination level by introducing an intensity mapping and quotient image relighting. Wang et al. [13] put self-quotient image. Chen et al. [14] came up with TV-based quotient image model for illumination normalization. Srisuk et al. [15] proposed Gabor quotient image to extend from the self-quotient image by which the 2D Gabor filter is applied instead of weighted Gaussian filter. An et al. [16] proposed a decomposed image under L1 and L2 norm constraint, then obtained illumination invariant large-scale part by region-based histogram equalization and got illumination invariant small-scale part by self-quotient image. The absolute error (L1 Loss) is mean absolute error (MAE), which refers to the mean of the predicted value of the model and the true value. MAE gradient is the same in most cases, which means that even for small loss values, the gradient is large. This is not good for convergence of the function and learning the model. However, no matter what kind of input value, there is a stable gradient, will not lead to the gradient explosion problem, with a relatively robust solution.

Adini et al. [4] proposed logarithmic transformation, directional gray-scale derivation, and Laplacian of Gaussian for illumination normalization. Single-scale retinex was put by Jobson et al. [17] for processing illumination. Fitzgibbon et al. [18] proposed Gaussian high-pass to process illumination. Local normalization technology proposed by Xie et al [19]. It could effectively eliminate the adverse effect of uneven illumination while keeping the local statistical properties of the processed image the same as in the corresponding image under normal illumination condition. Chen et al. [20] came up with a lighting normalization method based on the generic intrinsic illumination subspace, which was used as a bootstrap subspace for novel images. Du et al. [21] presented wavelet-based illumination normalization. Chen et al. [22] proposed logarithmic total variation for processing illumination. Chen et al. [23] put a new method named logarithmic discrete cosine transformation for illumination compensation and normalization. Tan and Triggs [24] processed illumination by combination of gamma correction, difference of Gaussian filtering, masking, and contrast equalization, which was called TT in literature [25]. Fan et al. [26] proposed a method named homomorphic filtering-based illumination normalization. The filters key component was a difference of Gaussian. Wang et al. [27] came up with illumination normalization based on Webers Law. Zhao et al. [28] proposed a selflighting ratio to suppress illumination differences in the frequency domain. A linear representation-based face illumination normalization method was put forward by Li et al. [29]. BimaSenaBayu et al. [30] proposed an adaptive contrast ratio based on fuzzy by considering two models of individual face as input, appearance estimation model and shadow coefficient model. Goel et al. [31] put forward an approach for illumination normalization based on discrete wavelet transformation and discrete cosine transformation. Discrete wavelet transformation was performed on the image and discrete cosine transformation was employed on low frequency sub

band. Then low frequency discrete cosine transformation coefficients were modified to suppress the illumination variations. Vishwakarma [32] proposed a fuzzy filter applied over the low-frequency discrete cosine transformation coefficients method for illumination normalization. With the development of 3D technologies, physical lighting models became a mainstream. Zhao et al. [33] decomposed lighting effect by ambient, diffuse, and specular lighting maps and estimated the albedo for face images with drastic lighting conditions. Tu et al. [34] presented a new and efficient method for illumination normalization with an energy minimization framework. Ahmad et al. [35] used independent component analysis and filtering to process illumination. Zhang et al. [36] presented a novel patch-based dictionary learning framework for face illumination normalization. Zheng et al. [37] proposed a local texture enhanced illumination normalization method based on fusion of difference of Gaussian filters and difference of bilateral filters. Zhang et al. [38] first combined Phongs model and lambertian model, then generated the Chromaticity Intrinsic Image (CII) in a log chromaticity space that was robust to illumination variations. The largest matching area was helpful to perform lighting normalization, occlusion de-emphasis and finally face recognition [39].

2.1.2 Deep learning-based methods The essence of deep learning is to solve a function to realize the mapping from input to output. Ma et al. [6] used GAN to process illumination of color faces. Though their method can generate vivid and well-lighted facial images based on illumination label. However, reconstruction and discriminators were used, it took more time to complete its computation. Han et al. [7] put forward asymmetric joint GANs to normalize face illumination. Their method contained two GANs, one is to normalize illumination, the other is to maintain personalized facial structures. Moreover, their method needs lighting labels.

2.2 GANS and their applications

The GAN [40] brings extraordinary vitality to the image generation, even expand to Fourier series [41]. With the help of the combination of GAN and CNN, DCGAN [42] makes a great leap in the ability of image generation. By specifying the input conditions, conditional GAN [43] can generate the specific target photos. At the same time, GAN leads to a series of breakthroughs in image inpainting [44], super-resolution [45], style transfer [46], image translation [47] [48] and so forth. In the field of face recognition, the training data is expanded by using GAN to generate some face photos [49], different expressions [50], different age faces [49], etc. For the application of generating frontal photos for FR, TP GAN [51], DR-GAN [52] and others synthesize large pose face images into frontal photos, and obtain good FR results.

3 Proposed method

3.1 Overall framework

In this section, we detail the proposed IN-GAN for illumination normalization. Fig. 2 shows the block diagram of IN-GAN, which takes a set of poor-lighted face images and corresponding standard illumination images as input and outputs a set of well-lighted face images in an end-to-end way. As is illustrated in Fig. 2, the core of our approach consists of a generator, a discriminator, and a feature extractor. Our generator is composed of an encoder network and a decoder network. The discriminator is employed to judge its input whether real images or fake images generated by the generator. The feature extractor is to extract face features for every face image. In the testing phase, we just use a generator to transform poor-lighted face images into well-lighted images. We utilize 3 loss items: adversarial loss, content loss and L1 loss. In traditional GAN, D network and G network are updated with simple cross entropy loss, while Least Squares GAN is updated with Least Squares loss. There are two advantages of choosing the least square Loss for updating: 1. Outlier Fake samples far away from the data set can be punished more strictly, so that the generated pictures are closer to the real data (and the images are clearer at the same time); 2. The least square ensures that the outlier sample has a larger penalty, which solves the original problem of insufficient (unstable) GAN training: However, the disadvantages are also obvious. adversarial loss excessive punishment of outliers may lead to a decrease in the diversity of sample generation, and the generated sample is likely to be a simple imitation and minor modification of the real sample. But adversarial loss is particularly good for face changes, which are mainly subtle changes. The content loss is to compare the L1 loss of the feature map from the intermediate convolution layer of VGG16.

3.2 Generator and discriminator architecture

The generator of our IN-GAN is inspired by the components of Pix2Pix [53] and U-net [54]. Our generator consists of 11 convolutional layers and 11 deconvolutional layers, each of which is equipped with a LeakyReLU as activation. Details of the generator are shown in Fig. 3. As is shown in Fig. 3, the input size of the generator is designed to be a 128×128 color image. The output resolution of our generator is 128×128 pixels in size. The dotted lines in Fig. 3 are skip connections that are conducive to feature retention. In the middle 6 convolutional layers, we utilize dropout to avoid overfitting and special deconvolutional and convolutional layers at the end of the generator for enhancing the synthetic ability of our model. For this special design, which further enhances the ability of feature retention. Because InstanceNorm [55] has the characteristics of preventing instance-specific mean and covariance shift simplifying the learning process, we utilize InstanceNorm after each convolutional layer. Moreover, experiments demonstrate that InstanceNorm makes our model converge fast, which means that our model can obtain higher recognition rate, good visual effect, and fine

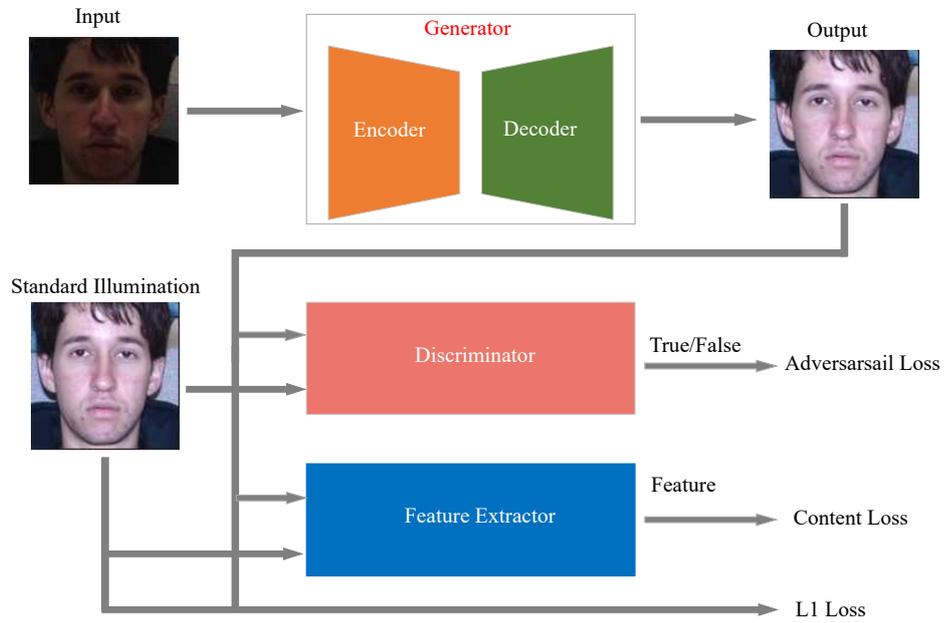


Fig. 2. Our overall generative adversarial network framework.

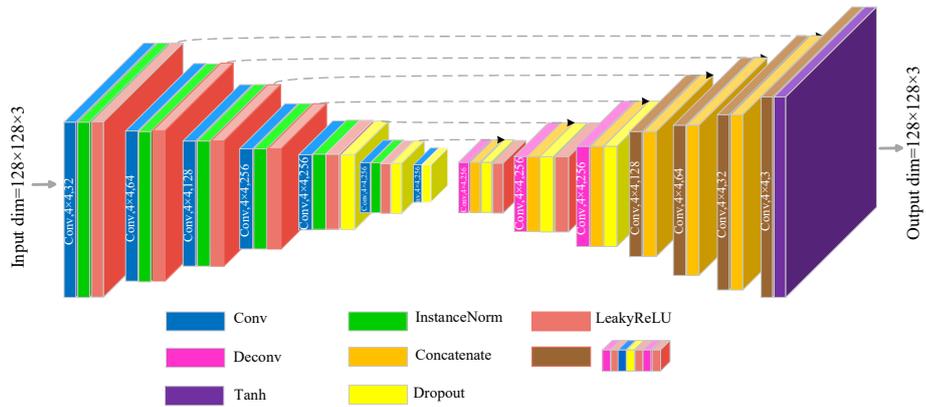


Fig. 3. The detailed structure of our generator.

illumination normalization results after about 14 epochs. InstanceNorm can be computed by:

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \varepsilon}}, \quad (1)$$

$$\mu_{ti} = \frac{1}{HW} \sum_{l=1}^W \sum_{m=1}^H x_{tilm}, \quad (2)$$

$$\sigma_{ti}^2 = \frac{1}{HW} \sum_{l=1}^W \sum_{m=1}^H (x_{tilm} - \mu_{ti})^2, \quad (3)$$

where $x \in \mathbf{R}^{T \times C \times W \times H}$ is an input tensor containing a batch of T images. Let x_{tijk} mean its $tijk$ -th element, where k and j are span spatial dimensions, i denotes the feature channel (color channel if the input is a RGB image), t is the index of the image in the batch. The discriminator of our IN-GAN is inspired by the compoents of Pix2Pix [51] and consists of 5 convolutional layers, each of which is equipped with a LeakyReLU as activation. Details of the discriminator are shown in Fig. 4. As is illustrated in Fig. 4, the input size of the discriminator is designed to be a 128×128 color image. We use InstanceNorm after each convolutional layers.

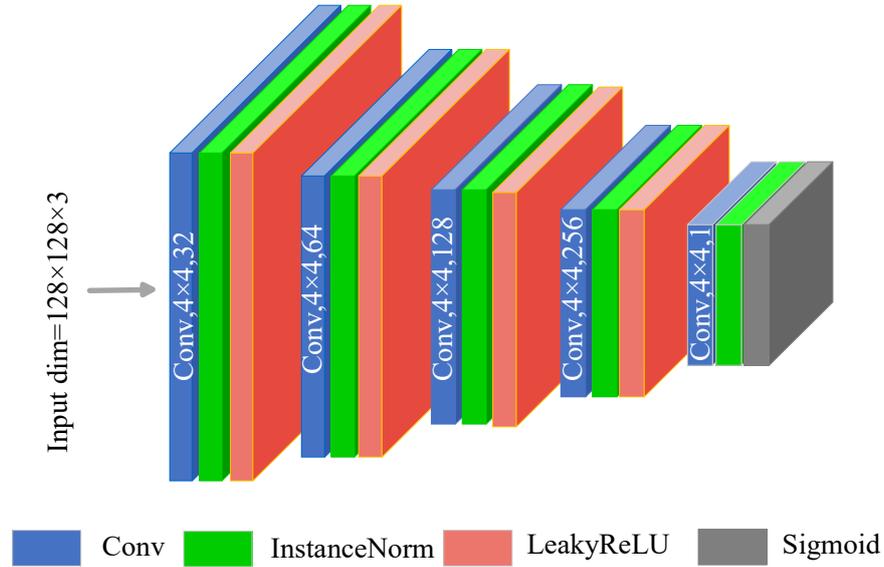


Fig. 4. The detailed structure of our discriminator.

3.3 Objective

Our method uses one discriminator D and a generator G , which constitutes a set of adversarial training processes respectively and optimizes their min-max problems. Our full objective is:

$$\begin{aligned} L(G, D) &= L_{adversarial}(G, D) \\ &+ \lambda_1 \times L_{content}(G) \\ &+ \lambda_2 \times L_{l1}(G), \end{aligned} \quad (4)$$

among them, λ_1, λ_2 are weight parameters respectively, $L_{adversarial}$, $L_{content}$ and L_{l1} are as follows:

$$\begin{aligned} L_{adversarial}(G, D) &= E_x[\log D(x)] \\ &+ E_{G(x)}[\log(1 - D(G(x)))], \end{aligned} \quad (5)$$

$$L_{content}(G) = \|F(y) - F(G(x))\|_1, \quad (6)$$

$$L_{l1}(G) = \|y - G(x)\|_1, \quad (7)$$

where x denotes input image, whereas y is target image (standard illumination), F means feature extractor such as VGG-19 [56], ResNet-50 [57] for extracting feature. In this study, we use ResNet-50 trained on VGGFace2 [58].

4 Experimental Results

4.1 Datasets

As far as the dataset, we choose Multi-PIE [59] that has 15 poses ranging from -90° to $+90^\circ$. Each pose includes 20 illuminations and up to 6 expressions. All the dataset has 337 identities. We select $0^\circ, -15^\circ, -30^\circ, -45^\circ$ faces, 20 illuminations and natural expression, without glasses from session 1 of Multi-PIE as our dataset. We detect and crop faces from the dataset with single shot scale-invariant face detector (S3FD) [60] and resize to 128×128 as our training and test set. The 07 illumination faces are chosen as standard face (standard illumination) and the rest are selected as poor-lighted facial images. The total identity number is 129, 30 identities are chosen as our test dataset and the rest 99 identities as our training dataset. There are a lot of dataset to refer, such as CMU Face Pose, Illumination, and Expression (PIE) Database (http://www.ri.cmu.edu/projects/project_418.html), AR Face Database (http://cobweb.ecn.purdue.edu/~aleix/a_leix_face_DB.html), BioID Face Database (<http://www.bioid.com/downloads/facedb/index.php>), Caltech Computational Vision Group Archive (Cars, Motorcycles, Airplanes, Faces, Leaves, Background) (<http://www.vision.caltech.edu/html-files/archive.html>), Carnegie Mellon Image Database (motion, stereo, face) (<http://vasc.ri.cmu.edu/idb/>).

As to the test dataset, we organize 4 settings. The setting 1 contains only frontal facial image, which has 30 identities and includes 19 illuminations. In regard to settings 2, it has the same identities as setting 1 but with large head poses such as $[-15^\circ, -30^\circ, -45^\circ]$ under 19 illuminations. With respect to setting 3, we choose -15° facial images and convert RGB to gray. In regard to setting 4, we obtain some poor-lighted faces using search engines from the Internet and Face Recognition Grand Challenge(FRGC) database [61]. For the purpose of verifying the performance of our algorithm, we also added people who wore glasses and faces under large head poses to our test dataset. All the faces of setting 4 are under less-controlled lighting variations.

4.2 Implementation Details

As to the encoder, we use LeakyReLU with a slope of 0.2 for activation first. In regard to the decoder, we also use LeakyReLU with a slope of 0.2. For gradient descent, we use Adam [62] optimizer, and choose a learning rate of 0.0002 with momentum parameters $\beta_1 = 0.5$, $\beta_2 = 0.999$, $\text{weightdecay} = 0.0001$. One 1080Ti graphics cards with batch size of 16 is used during training and 20 epochs has completed within about 25 minutes. Besides, during the training period, we do not use any data enhancement methods. By setting different values for λ_1 and λ_2 , we get 3 combinations of loss items. We set $\lambda_1 = 1.0$, $\lambda_2 = 0$ to train a model with content loss item only. We choose $\lambda_1 = 0.0$, $\lambda_2 = 1.0$ to train a model with L1 loss item only. When $\lambda_1 = 1.0$ and $\lambda_2 = 0.1$, we train a model with content and L1 loss items.

4.3 Metrics

At present, most literatures evaluate the performance of illumination normalization algorithms from two aspects: one is to compare recognition rate, the other is to illustrate some face images before and after processing by various methods. We make use of cosine similarity of feature vectors for face recognition. For the purpose of more comprehensively estimating the performance of various illumination normalization algorithms, except for comparing recognition rate and illustrating examples, we also adopt peak signal to noise ratio (PSNR) to evaluate the performance of various illumination normalization methods. Cosine similarity and PSNR are briefly introduced as follows:

1. Cosine similarity is defined as:

$$\cos(\theta) = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}, \quad (8)$$

where A and B are the feature vectors obtained from ResNet-50 [55] trained on VGGFace2 [56], A_i denotes i -th element of vector A , B_i denotes i -th element of vector B .

2. PSNR is defined as:

$$PSNR = 10 \times \log_{10} \left[\frac{(2^n - 1)^2}{MSE} \right], \quad (9)$$

where n is the maximum bits of each pixel, MSE means mean square error of two images, which can be computed by:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2, \quad (10)$$

where m and n denote the width and height of image, I and K are two gray images that are equal in width and height.

4.4 Quantitative Evaluation

In this section, we evaluate the performance of other existed methods and our algorithm from two aspects: one is face recognition rate, the other is peak signal to noise ratio (PSNR). In terms of recognition rate, we evaluate the performance of Directional Gray-scale Derivation X (DGDx) [4], Directional Gray-scale Derivation Y (DGDy) [4], Gaussian High-pass (GHP) [18], Histogram Equalization (HE) [8], Logarithmic Discrete Cosine Transform(LDCT) [23], Local Normalization Technology (LN) [19], Laplacian of Gaussian (LoG) [4], Logarithmic Transform (LT) [4], Logarithmic Total Variation (LTV) [22], Self Quotient Image (SQI) [13], Single-scale Retinex (SSR) [17], TT [24] and ours. In case of PSNR, we only evaluate GHP, GIC, HE, LN, LT, SQI, SSR, TT and ours. All the aforementioned methods have higher recognition rate and better visual effect than the rest when we use them to process frontal faces.

As is shown in Table. 1, the recognition rate of GIC, HE, LN, SQI, SSR and ours is 100% when the probe images are frontal faces under various illuminations processed by a variety of methods. GHP, LT, LTV and TT also obtain high recognition rate. All of them are greater than 92%. In general, all the aforementioned methods have the ability to preserve details or identity information in some sense. But GIC, HE, LN, SQI, SSR, and ours obtain the maximal recognition rate. In qualitative comparisons section, we illustrate contrast face images of the aforesaid algorithms.

Table 1. Recognition rate on setting 1

Method	DGDx	DGDy	GHP	GIC	HE	LDCT	LN	LoG	LT	LTV	SQI	SSR	TT	Ours
Recognition Rate	30.88%	21.75%	92.81%	100%	100%	70.00%	100%	2.98%	99.47%	96.32%	100%	100%	94.91%	100%

In Table. 2, we illustrate the PSNR of GHP, GIC, HE, LN, LT, SQI, SSR, TT and ours under 19 illuminations. As is shown in the Table. 2, we can conclude

that the PSNR of original images is very different, but after illumination normalization, GHP, GIC, HE, LN, LT, SQI, SSR, TT and ours narrow the difference of PSNR and enhance the quality of poor-lighted images. It is obvious that our method is the best of all the illumination normalization algorithms illustrated in Table. 2 and our approach archives the best result of illumination normalization.

4.5 Qualitative Comparisons

Table 2. PSNR of original images and images processed by various illumination normalization methods

Methods Illumination	Original	GHP [18]	GIC [9]	HE [8]	LN [19]	LT [19]	SQI [13]	SSR [17]	TT [24]	Ours
00	7.17	10.12	14.85	13.27	10.12	15.90	13.63	14.48	13.29	20.57
01	9.17	11.28	13.12	11.73	9.30	14.12	13.72	13.33	13.11	20.49
02	10.10	11.70	13.51	12.13	10.35	14.69	14.11	13.62	13.47	21.11
03	10.36	11.65	13.63	12.39	11.10	14.78	14.14	13.67	13.62	20.82
04	11.59	11.63	14.03	12.86	12.19	14.82	14.34	13.90	13.78	20.92
05	14.27	11.81	14.37	13.45	13.76	14.44	14.57	13.61	13.94	20.54
06	15.84	11.51	14.72	14.42	14.97	14.80	14.57	13.53	14.02	20.43
08	16.62	12.04	14.67	14.16	15.75	13.79	15.25	13.19	14.11	20.77
09	13.12	12.28	13.68	13.27	13.98	14.14	15.22	13.62	14.05	20.83
10	11.27	12.04	13.55	12.66	12.82	14.23	14.85	13.82	13.79	21.06
11	10.72	12.30	13.06	12.15	11.78	14.05	14.70	13.57	13.68	20.78
12	9.74	11.83	13.20	11.89	10.71	14.28	14.08	13.58	13.38	20.90
13	9.59	11.58	12.59	11.46	9.68	13.82	13.89	13.32	13.08	20.55
14	10.58	11.84	14.04	12.99	12.20	15.12	14.39	14.21	13.82	21.06
15	12.79	12.15	14.56	13.55	13.47	15.18	14.78	14.17	13.99	21.22
16	13.86	11.37	15.51	14.33	13.82	16.42	14.58	14.22	13.78	21.05
17	12.19	12.37	14.26	13.30	13.49	15.04	15.20	14.03	13.98	21.29
18	10.54	11.98	14.17	12.89	12.59	15.01	14.93	14.12	13.83	21.11
19	7.28	10.13	14.89	13.20	10.15	15.72	13.65	14.45	13.29	20.59
Average	11.41	11.67	14.02	12.95	12.22	14.75	14.45	13.81	13.68	20.85

Because most of the previous methods focused on the illumination normalization of gray faces. For the sake of fairness, we also conduct the comparative experiment on gray faces. According to Fig. 5, the faces from 2 to 7 columns are -15° , from 8 to 13 are -30° , from 14 to 19 are -45° . The first row is the original poor-lighted facial images. The second and third rows are the results of GHP and GIC separately. The fourth row is obtained from HE method. Next, we show the experimental results of LN, LT, SQI, SSR, TT, DGD_x, DGD_y, LDCT, LoG, and LTV. The final row is the output results of our method.

As can be observed from the Fig. 5, in general, these methods such as SQI, GIC, HE, LN, SSR, LT and ours achieve good performance, whereas the other methods preserve only part of the details and have poor visual effect. Although

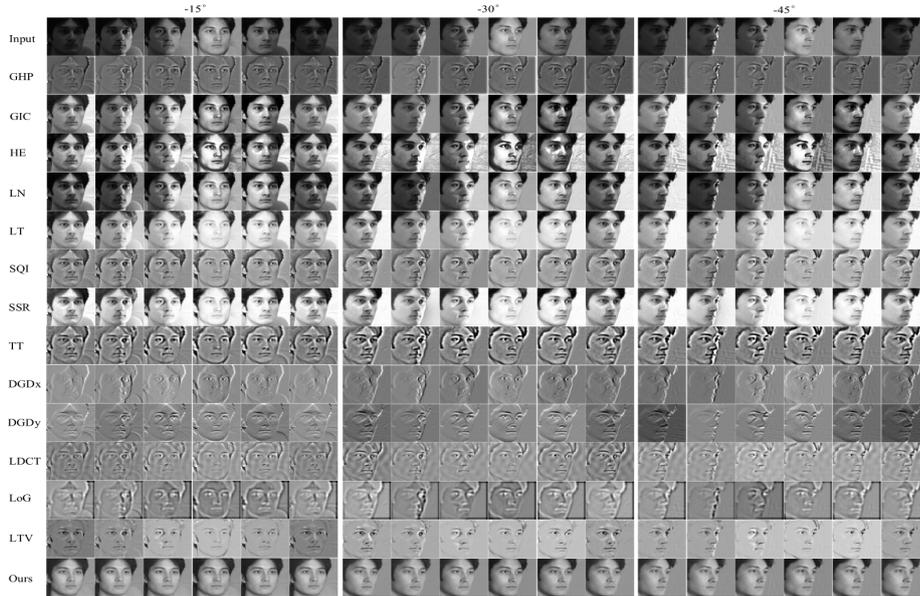


Fig. 5. Some gray faces under various illuminations and large head poses before and after processing by various methods

SQI, GIC, HE, LN, SSR, LT achieve good results, they have some defects. For instance, though the SQI method has a high recognition rate, its visual effect is a little bad. From the fourth row of Fig. 5, we can see that HE method has a lot of noise and does not handle shadows effectively. Although LN method improves the illumination of the original facial images, it can not deal with the shadows effectively. From the third, ninth, eleventh and fifteenth image of the third row of Fig. 5, we can see that GIC can not handle light and cast shadow effectively. There is a large difference in light intensity on the whole face area. Though SSR method has good visual effect, some images are oversaturated and can not process cast shadow effectively. Although the LT method can deal with the illumination of each facial image effectively and has fine visual effect, it can not process cast shadow effectively and some images are oversaturated. It is obvious in the final row of Fig. 5 that our method can not only deal with the illumination of each image but also keep the corresponding identity of the images under various illuminations effectively. In summary, GIC, SSR, LT and ours perform better than other illumination normalization algorithms. If we evaluate it from three aspects: recognition rate, illumination normalization effect and visual effect, our method is the best of all the aforementioned approaches.

There is a large difference in light intensity on the whole face region. Though SSR method is with good visual effect, some images are oversaturated and cannot process cast shadow effectively. Although the LT method can deal with the illumination of each facial image effectively and with fine visual effect, it cannot

process cast shadow effectively and some images are oversaturated. It is obvious in the final row in Fig. 5. Our method can not only deal with the illumination of each image but also keep the corresponding identity of the images under various illuminations effectively. In summary, GIC, SSR, LT and ours perform better than other illumination normalization algorithms. From three aspects: recognition rate, illumination normalization effect and visual effect, our method is the best of all the approaches. From the previous parts, the methods such as GHP, GIC, HE, LN, LT, SQI, SSR, TT and ours have better performance than others. To verify their performance under less-controlled lighting variations, another set of experiments illustrate the comparison results are shown in Fig. 6. It can be concluded that GIC, HE, LN, LT, SSR and ours have better visual effect than others. From the third image of the third row, GIC cannot process the light. From the third image of the fourth row, HE encounters the same problem and there is some noise. In line 5 of Fig. 6, it is obvious that all the face images still have severe shadows after processing by LN. LT and SSR are the best if only consider from the aspect of visual effect. But some images are oversaturated. Our algorithm is the best of all the approaches from two aspects: illumination normalization and visual effect.



Fig. 6. Some gray faces under less-controlled illuminations by various methods.

4.6 Ablation studies

Since our method is without illumination label and identity label, we compare it only to algorithms that do not require any label. Because there is no label free algorithm in deep learning, we only illustrate some results of our method. According to Fig. 7, the first row is the original facial images under various illuminations. The second row is the synthetic facial images using our approach training with the content loss item only. The third row is the synthetic faces using our method trained with the L1 loss item only. The final row of Fig. 7 is the result obtained from our algorithm trained with the content and L1 loss items. According to Fig. 7, our method can not only make the poor illumination become well-lighted and unanimous but also preserves the identity information of the original poor-lighted facial images effectively. Since content and L1 loss items can preserve identity well, the model trained with the combination of content and L1 loss items can not only keep identity effectively but also obtain better visual effect.



Fig. 7. Some frontal color faces under various loss items by our method.

In order to further verify the performance of our algorithm, we perform illumination normalization experiment on the non-frontal facial images under various lighting conditions. As is shown in Fig. 8, we can see that the first and third rows are -15, -30 and -45 faces under various illuminations, the second row is the output of the first row. The fourth row is the corresponding synthetic results of the third row. It is obvious that our method can not only process illumination of color faces under large-poses and poor-lighted conditions but also keep corresponding identities effectively.

In practical application, it is basically less-controlled illumination. Therefore, it is necessary to verify the performance of our algorithm under the less-controlled lighting variations. According to Fig. 9, the first row is the original input with poorly lighted faces and the second row is the corresponding output. It is apparent that the input images are not aligned and under various less-controlled lighting variations. The results of Fig. 9 indicate that our method



Fig. 8. Some color faces under large head poses and various illuminations before and after processing by our method.

can not only process the illumination of color face under less-controlled lighting variations with various head poses but also keep its identity effectively. Though no one wore glasses in our training set, our method still has the synthetic ability to generate glasses, which demonstrates our approach has a strong ability of feature retention.



Fig. 9. Verify our algorithm to process the illumination of color faces.

As is illustrated in the Fig. 10, the first row is the original images, and the next 3 rows are the corresponding output of epoch 1, 5 and 13. According to Fig. 10, we can conclude that our method trained with content loss, L1 loss, the combination of content and L1 loss has good feature retention ability. When we combine content and L1 loss, our method converges faster than L1 loss and content loss used separately. Although content loss also makes our algorithm converge fast, its visual effect is not as good as the combination of content and L1 loss. It is apparent in the Fig. 10 that our method has the advantages such as rapid convergence, good feature retention, favorable illumination normalization results.

To evaluate our method quantitatively in Fig. 11 and Fig. 12. They are under 3 loss items from epoch 1 to 14. Fig. 11 illustrates the recognition rate of frontal faces under 3 loss items from epoch 1 to 14. From Fig. 11, it is known that the combination of content and L1 loss makes our model converge fast and obtain higher recognition rate. Fig. 12 shows the recognition rate of -15° faces under



Fig. 10. Verify our algorithm to process the illumination of color faces.

3 loss items from epoch 1 to 14, which demonstrates that our method can get better performance after combining content loss and L1 loss than using content loss or L1 loss separately. From Fig. 11 and 12, It can be concluded that our method converges fast and obtains high recognition rate after combining content loss and L1 loss.

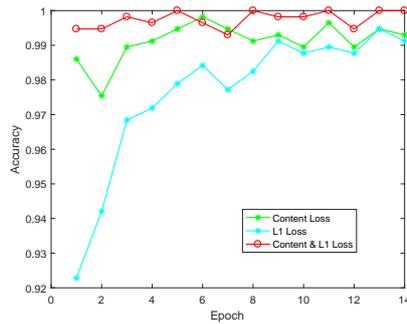


Fig. 11. Recognition rate of frontal faces.

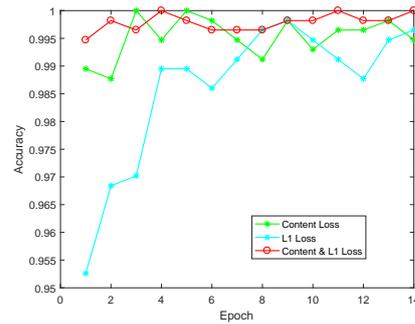


Fig. 12. Recognition rate of -15°

5 Validation for Identity Preserving

Table 3. Recognition rate of faces under various head poses and lighting variations after doing illumination normalization by our method.

angle	0°	-15°	-30°	-45°
Recognition Rate	100%	100%	100%	97.54%

In this section, we discuss our algorithm's ability to keep identity information by conducting face recognition experiment. Table. 3 shows the recognition results of our algorithm. In line 2 of Table. 3, it is the recognition results of color faces under 0° , -15° , -30° , -45° . It is obvious that our method under content and L1 loss items can obtain high recognition rate. As is illustrated in Table. 3, though there is no any input such as identity label, illumination label, our method still preserves the corresponding identities of the original faces effectively.

6 Conclusion

In this study, we put a novel and practical deep fully convolutional neural network architecture for illumination normalization of color face termed IN-GAN. Our method can process not only the illumination of color face images but also the illumination of gray face images. Furthermore, all the existed methods mainly focus on processing the illumination of frontal or near frontal face. Our scheme can not only process the illumination of frontal face but also the non-frontal face. Moreover, our method can normalize illumination of face image, and retain identity information effectively. Finally, though our trained model on faces under well-controlled lighting variations, it can process face under less-controlled lighting variations and preserve identity information effectively. In our further researches, other features, and geometric structure [63] need to be considered. We find that the number of layers and types of connection are important. The authors showed that the six-convolution layer and three fully-connected layer CNN, nine-layers in total, achieved better performance in sensitivity, specificity, accuracy, and precision [64]. In [65], the performance of parametric rectified linear unit is better than ordinary ReLU. They also verify that batch normalization overcomes the internal covariate shift and dropout got over the overfitting. we shall try different layer CNN model, and various types of connection in the future. There is no gradient in ReLU, when values are less than zero. However, there is a small gradient in LeakyReLU, while values are less than zero. Therefore, LeakyRelu is selected in our experiment ReLU can get good results. Although our illumination normalization algorithm achieves preferable results from qualitative and quantitative comparisons. Compared with other algorithms, our algorithm has gained advantages. There is a lot of future work here worth continuing to study:

1. To improve our network structure for preserving more texture details.
2. To train a feature extractor and classifier for the facial images after normalizing illumination by our method.
3. To process illumination normalization of other image types, such as landscape and medical images.
4. To the preprocessing stage of other visual analysis tasks, such as facial landmark detection and face alignment.

Conflict of interest

We promise that this manuscript is the authors original work and has not been published nor has it been submitted simultaneously elsewhere. All authors have checked the manuscript and have agreed to the submission.

References

1. Zhang Y, Tsang I, Luo Y, Hu C, Lu X and Yu X: Recursive copy and paste GAN: face hallucination from shaded thumbnails. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp 1-1. . <https://doi.org/10.1109/TPAMI.2021.3061312>. (2021)
2. Tsai Y, Hsu L, Hsieh Y, Lin S: The real-time depth estimation for an occluded person based on a single image and openpose method. *Mathematics*, 8(8), 1333; <https://doi.org/10.3390/math8081333>. (2020)
3. Moret-Tatay C, Baixauli-Fortea I, Grau Sevilla M D, Irigaray, Tatiana Q: Can You Identify These Celebrities? A network analysis on differences between word and face recognition. *Mathematics*, 8(5), 699; <https://doi.org/10.3390/math8050699>. (2020)
4. Adini Y, Moses Y, and Ullman S: Face recognition: The problem of compensating for changes in illumination direction. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19, pp 721732. (1997)
5. Al-Osaimi F R, Bennamoun M, and Mian A S: Illumination normalization for color face images, *International Symposium on Visual Computing, Advances in Visual Computing* pp 90-101. (2006)
6. Ma W, Xie X, Yin C, and Lai J H: Face image illumination processing based on generative adversarial nets. *2018 24th International Conference on Pattern Recognition (ICPR)*, pp 25582563. (2018)
7. Han X, Yang H, Xing G, and Liu Y: Asymmetric joint gans for normalizing face illumination from a single image. *IEEE Transactions on Multimedia*, vol. 22, no. 6, pp 1619-1633, doi: 10.1109/TMM.2019.2945197. (2020)
8. Pizer S M, Amburn E P, Austin J D, Cromartie R, and Zuiderveld K: Adaptive histogram equalization and its variations. *Computer Vision Graphics & Image Processing*, 39(3), pp. 355-368. (1987)
9. Shan, S, Gao, W, Cao, B, and Zhao, D: Illumination normalization for robust face recognition against varying lighting conditions. *2003 IEEE International SOI Conference. Proceedings (Cat. No.03CH37443)*, pp. 157164. (2003)
10. Xie X and Lam K-M: Face recognition under varying illumination based on a 2d face shape model. *Pattern Recognition*, 38, pp. 221230. (2005)
11. Lee P H, Wu S W and Hung Y P: Illumination compensation using oriented local histogram equalization and its application to face recognition. *IEEE Transactions on Image Processing*, 21, pp. 42804289. (2012)
12. Shashua A and Riklin-Raviv T: The quotient image: Class-based rerendering and recognition with varying illuminations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23, pp. 129139. (1999)
13. Wang H, Li S Z and Wang Y: Generalized quotient image. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2, pp III. (2004)
14. Chen T, Yin W, Zhou X S, Comaniciu D, and Huang T S: Illumination normalization for face recognition and uneven background correction using total variation-based image models. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2, pp 532539 (2005)

15. Srisuk S and Petpon A: A gabor quotient image for face recognition under varying illumination. *ISVC Advances in Visual Computing*, pp 511-520 (2008)
16. An G, Wu J and Ruan Q: An illumination normalization model for face recognition under varied lighting conditions. *Pattern Recognition Letters*, 31, pp 1056-1067 (2010)
17. Jobson D, Rahman Z and Woodell, G: Properties and performance of a center/surround retinex. *IEEE Transactions on Image Processing*, 6, no. 3, pp 451-462 (1997)
18. Fitzgibbon A W and Zisserman A: On affine invariant clustering and automatic cast listing in movies. *Computer Vision*, pp 304-320 (2002)
19. Xie X and Lam K-M (2006) An efficient illumination normalization method for face recognition. *Pattern Recognition Letters*, 27, pp 609-617
20. Chen C-P and Chen C-S: Lighting normalization with generic intrinsic illumination subspace for face recognition. *Tenth IEEE International Conference on Computer Vision*, 2, pp 1089-1096 (2005)
21. Du S and Ward R K: Wavelet-based illumination normalization for face recognition. *IEEE International Conference on Image Processing*, 2, pp II954 (2005)
22. Chen T, Yin W, Zhou X S, Comaniciu, D, and Huang, T S: Total variation models for variable lighting face recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(9), 1519-1524 (2006)
23. Chen W, Er M J, and Wu S: Illumination compensation and normalization for robust face recognition using discrete cosine transform in logarithm domain. *IEEE Transactions on Systems Man and Cybernetics Part B Cybernetics*, 36(2), pp 458-466 (2006)
24. Tan X, and Triggs B (2010) Enhanced local texture feature sets for face recognition under difficult lighting conditions. *IEEE Transactions on Image Processing*, 19, pp 1635-1650
25. Han H, Shan S X Chen and Gao W: A comparative study on illumination preprocessing in face recognition. *Pattern Recognition*, 46, pp 1691-1699 (2013)
26. Fan C-N and Zhang F-Y: Homomorphic filtering-based illumination normalization method for face recognition. *Pattern Recognition Letters*, 32, pp 1468-1479 (2011)
27. Wang B, Li W, Yang W and Liao Q: Illumination normalization based on webers law with application to face recognition. *IEEE Signal Processing Letters*, 18, pp 462-465 (2011)
28. Zhao X, Shah S K and Kakadiaris I A: Illumination normalization using self-lighting ratios for 3d2d face recognition. *ECCV Workshops. European Conference on Computer Vision, Workshops and Demonstrations*, pp 220-229 (2012)
29. Li Y, Meng L and Feng J: Lighting coefficients transfer based face illumination normalization. *Chinese Conference on Pattern Recognition*, pp 268-275 (2012)
30. BimaSenaBayu D and Miura J: Fuzzy-based illumination normalization for face recognition. *2013 IEEE Workshop on Advanced Robotics and its Social Impacts*, pp 131-136 (2013)
31. Goel T, Nehra V and Vishwakarma V P: Illumination normalization using down-scaling of low-frequency dct coefficients in dwt domain for face recognition. *2013 Sixth International Conference on Contemporary Computing*, pp 295-300 (2013)
32. Vishwakarma V P: Illumination normalization using fuzzy filter in dct domain for face recognition. *International Journal of Machine Learning and Cybernetics*, 6, pp 1734 (2015)
33. Zhao X, Evangelopoulos G, Chu, D Shah, S K, and Kakadiaris, I A: Minimizing illumination differences for 3d to 2d face recognition using lighting maps. *IEEE Transactions on Cybernetics*, 44, pp 725-736 (2014)

34. Tu X, Yang F, Xie M, and Ma Z: Illumination normalization for face recognition using energy minimization framework. *IEICE Transactions*, 100-D, pp 13761379 (2017)
35. Ahmad F, Khan A, Islam, I U, Uzair, M, and Ullah, H: Illumination normalization using independent component analysis and filtering. *The Imaging Science Journal*, 65(5), 308-315 (2017)
36. Zhang, Y, Wang, L, Guan X, and Wei, H (2018) Illumination normalization for face recognition via jointly optimized dictionary-learning and sparse representation. *IEEE Access*, 6, pp 6663266640
37. Zheng C, Wu S, Xu W, and Xie S: Illumination normalization via merging locally enhanced textures for robust face recognition. *arXiv preprint*, arXiv:1905 03904 (2019)
38. Zhang W, Zhao X, Morvan J M, and Chen L: Improving shadow suppression for illumination robust face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(3), 611-624 (2019)
39. McLaughlin N, Ming J, and Crookes D: Largest matching areas for illumination and occlusion robust face recognition. *IEEE Transactions on Cybernetics*, vol. 47, no. 3, pp 796-808, doi: 10.1109/TCYB.2016.2529300. (2017)
40. Goodfellow I J Pouget-Abadie J, Mirza M, Xu B: Generative adversarial nets. *Advances in Neural Information Processing Systems*, 27, pp 26722680 (2014)
41. Gonzalez-Prieto , Mozo A, Talavera E, Gomez-Canaval S: Dynamics of fourier modes in torus generative adversarial networks. *Mathematics*, 9, 325. <https://doi.org/10.3390/math9040325>. (2021)
42. Radford A, Metz L, and Chintala S: Unsupervised representation learning with deep convolutional generative adversarial networks. *International conference on learning representations*, arXiv preprint:1511.06434. (2016)
43. Mirza M and Osindero S: Conditional generative adversarial nets, arXiv, preprint: arXiv:1411.1784 (2014)
44. Demir U and Lnal G B: Patch-based image inpainting with generative adversarial networks. arXiv, preprint: arXiv:1803.07422 (2018)
45. Wang X, Yu K, Wu S, Gu J, Liu Y (2018) Esrgan: Enhanced super-resolution generative adversarial networks. *Proceedings of the European Conference on Computer Vision (ECCV)*, pp 6379
46. Yang F W, Lin H J, Yen S-H and Wang C-H: A study on the convolutional neural algorithm of image style transfer. *International Journal of Pattern Recognition and Artificial Intelligence*, 33, no. 5, pp 1954020 (2019)
47. Lin J, Xia Y, Qin T, Chen Z, and Liu T Y: Conditional image-to-image translation. *2018 IEEE Conference on Computer Vision and Pattern Recognition*, pp 55245532 (2018)
48. Zhu J-Y, Park T, Isola P, and Efros A A: Unpaired image-to-image translation using cycle-consistent adversarial networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp 22422251 (2017)
49. Shrivastava A, Pfister T, Tuzel O, Susskind J, and Webb R: Learning from simulated and unsupervised images through adversarial training. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp 22422251 (2017)
50. Pumarola A, Agudo A, Martinez A M, Sanfeliu A, and Moreno-Noguer F: Ganimation: anatomically-aware facial animation from a single image. *European Conference on Computer Vision 8*, pp 835851 (2018)
51. Huang R, Zhang S, Li T, and He R: Beyond face rotation: global and local perception GAN for photorealistic and identity preserving frontal view synthesis. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp 24582467 (2017)

52. Tran L, Yin X and Liu X: Disentangled representation learning gan for pose-invariant face recognition. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 12831292 (2017)
53. Isola P, Zhu J-Y, Zhou T and Efros, A A: Image-to-image translation with conditional adversarial networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 59675976 (2017)
54. Ronneberger O, Fischer P and Brox, T: U-net: Convolutional networks for biomedical image segmentation. Medical image computing and computer assisted intervention, pp 234241 (2015)
55. Ulyanov D, Vedaldi A and Lempitsky, V S Instance normalization: The missing ingredient for fast stylization. ArXiv 2016, abs/1607.08022. (2016)
56. Simonyan K and Zisserman A: Very deep convolutional networks for large-scale image recognition. arXiv, preprint: arXiv:1409.1556 (2014)
57. He K, Zhang X, Ren S and Sun, J: Deep residual learning for image recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 770778 (2016)
58. Cao Q, Shen L Xie W, Parkhi O M and Zisserman, A: Vggface2: A dataset for recognizing faces across pose and age. 13th IEEE International Conference on Automatic Face and Gesture Recognition, pp 6774 (2018)
59. Gross R, Matthews I, Cohn J, Kanade T and Baker, S: Multi-pie, Image and Vision Computing, 28, no. 5, pp 807813 (2010)
60. Zhang S, Zhu X, Lei Z, Shi H, Wang X, and Li S Z: S3fd: Single shot scale-invariant face detector. 2017 IEEE International Conference on Computer Vision (ICCV), pp 192201 (2017)
61. Phillips P J, Flynn P J, Scruggs T, Bowyer K W, and Worek W: Overview of the face recognition grand challenge. IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (CVPR), 1, pp 947954 (2005)
62. Kingma D P and Ba, J L: Adam: A method for stochastic optimization, international conference on learning representations. Computer science, <https://arxiv.org/abs/1412.6980> (2015)
63. Pareja-Corcho J, Betancur-Acosta O, Posada J, Tammara A, Ruiz-Salguero O, Cadavid C: Reconfigurable 3D CAD feature recognition supporting confluent n-dimensional topologies and geometric filters for prismatic and curved models. Mathematics, 8(8), 1356; <https://doi.org/10.3390/math8081356> (2020)
64. Wang S H, Sun J D, Mehmood I, Pan C C, Chen Y, Zhang Y D: Cerebral micro-bleeding identification based on a nine-layer convolutional neural network with stochastic pooling. Concurrency and computation: practice and experience, pp e5130.1-e5130.16 (2020)
65. Wang S H, Muhammad K, Hong J Sangaiah A K, and Zhang Y D: Alcoholism identification via convolutional neural network based on parametric ReLU, dropout, and batch normalization. Neural Computing and Applications, 32, pp 665680 (2020)

Attention-based deep residual network for ghost imaging

Mingshan Liu and Yunyun Niu*

School of Information Engineering, China University of Geosciences in Beijing,
Beijing 100083, China yniu@cugb.edu.cn

Abstract. Although ghost imaging using deep learning (GIDL) reconstruction technology can obtain higher quality images than traditional physical methods, one of the main problems is that the image reconstruction process cannot focus on the image itself in a noisy environment, which makes it more difficult to further improve the clarity and quality of the reconstructed images. Here we propose a deep residual transposed network based on an attention mechanism (DAT-Net) to deal with this problem. The experimental results show that the DAT-Net can focus on the image itself and obtain higher image quality in a noised image reconstruction process. The proposed network model is better than the current best model in multiple evaluation indicators. With the support of the attention mechanism, this method also has great practical application value for image reconstruction, image enhancement, denoising of single-pixel computational imaging, and ghost imaging.

Keywords: ghost imaging · deep residual network · attention-based model

Introduction

Ghost imaging (GI) is a new type of imaging technology that uses two-photon composite detection to recover the spatial information of the object to be measured [1]. It was initially achieved by the quantum entanglement of the entangled light source [2], but later studies have shown that a quantum light source is not necessary [3] [4]. The traditional ghost imaging system is achieved by two beams of light working together to achieve image reconstruction, the two beams of light test and reference [2]. In subsequent physical experiments, the reference light source was proved to be unnecessary in reconstructing the image [4] [5]. The part of the reference light source can be directly completed by enhancing the random phase spatial light modulator of the test light source, and then according to the test beam carried the image features complete the image reconstruction process. Since GI appeared, it has been successfully applied in many fields, such as three-dimensional imaging [6] [7], optical watermarking [8] [9] and so on.

The rapid development of deep learning has promoted a new upsurge in the development of the computer field. For example, there are extraordinary

* Corresponding author

effects in image recognition [10], natural language processing [11] [12], and target recognition [13]. Also, deep learning also has a good contribution to the inverse problem of optical imaging [14], making it possible to combine deep learning and ghost imaging [15] [16]. Because the conventional image reconstruction process often takes much time to obtain high-resolution images with as little noise as possible [17], such rigorous experimental conditions make ghost imaging not in real life when it is first discovered. Therefore, this paper uses direct simulation data to train to complete the construction of neural networks and the comparison of related network models. The combination of the two, through experimental verification, the image effect obtained by the image reconstruction technology based on deep learning can also be like the image effect of ghost imaging on compressive sensing (CSGI) based on the Fourier integral transform [18], in addition to effectively shortening the experiment required time to improve the quality of reconstructed images. The method of applying deep learning to ghost imaging (GIDL) was proposed by Lyu et al [15]. They used an end-to-end method [15] [19], even at a lower Nyquist sampling rate ($\beta = \alpha/\eta$, α is the number of experiments for one-dimensional bucket measurement, and η represents the pixel size of the image. The image used in this experiment is 28×28 , $\eta = 784$). The same can achieve high-quality image reconstruction. However, a problem that must be considered is that under a lower sampling rate, a certain amount of noise interference will inevitably be introduced. In response to this problem, in DDA-Net [19], the target image is directly extracted from the one-dimensional target image under the condition of under-Nyquist sampling rate, thereby reducing the corresponding noise intake; in DRU-Net [20], the upper and lower the sampled depth residual network module is used to obtain the reconstruction process of the noisy image, which improves the quality of the reconstructed image to a certain extent.

However, in the image reconstruction network models that have been proposed so far, although denoising can be carried out to a certain extent, the above model will be filtered to a certain extent when denoising, both the image and noise, making the image saturation not prominent enough. The model is unable to focus on the image itself to be retained so that some features in the original image will be treated as noise filtering in the reconstructed image, and high-quality images cannot be obtained. In response to the problems that arise, we propose a deep residual transposed network based on the attention mechanism (DAT-Net). This network is mainly composed of deep residual network blocks [21], transposed convolution and attention modules [22], making the network model. At the same time denoising, more attention can be paid to the image itself, while removing the noise, the characteristics of the image itself are also preserved as much as possible to obtain a reconstructed image closer to the original image.

Methods

This part of the experiment is directly simulated by an experimental computer [16], so two-dimensional image information is selected. Figure 1 is a schematic diagram of a CGI device. It can be seen from the schematic diagram that when the test object is irradiated by natural light, it will enter the current spatial light modulator (SLM) [1] [5]. The light carrying image information will be divided into two directions, and part of the image will be obtained through the signal detector. The characteristic signal is finally combined with the reference beam to obtain a reconstructed image. The following formula can be used to express this process:

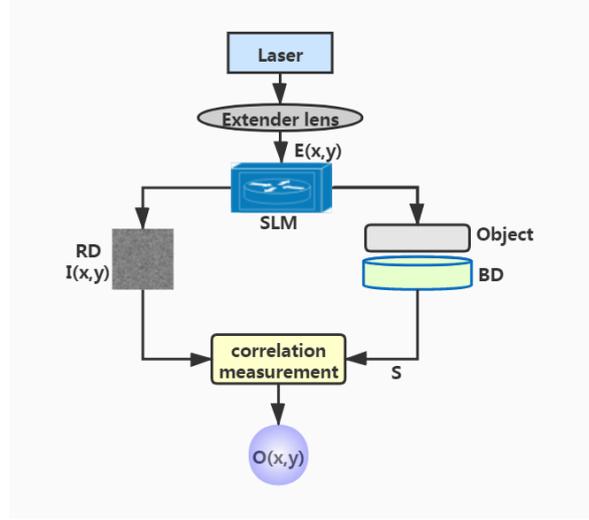


Fig. 1. CGI simulation diagram

$$S(x, y) = \xi \left(\sum_{n=1}^N f(\theta) E_m(x, y) \right) \quad (1)$$

where the subscript m represents the average carried information calculated by the bucket measurement after multiple calculations. N represents the size of the pixel. S represents the average information carried by the bucket measurement. $\xi(\cdot)$ represents the average value of the sum of all the information obtained after summing each image. θ is the parameter matrix in the nonlinear function.

According to the image information calculated above, the calculation function of the reconstructed image with a certain noise can be obtained:

$$T_{GI}(x, y) = \xi(S(x, y)I_m(x, y)) - \xi(S(x, y)) \cdot \xi(I_m(x, y)) \quad (2)$$

where $I(x, y)$ represents random patterns. By adding a random model composed of certain noise to the original image, the resolution of the original image will be destroyed. But this situation is most in line with the real world. It has been proven that GIDL can use simulated data for simulation experiments [15] [19]. While using a deep learning model for image reconstruction, a large enough image data set must be used to train the model. It will take a lot of time to obtain noised images in actual physical experiments. Therefore, we directly used MNIST dataset to divide the image dataset into a training set and a test set. The image with noise was used as the training set, while the original image was used as label data. The reconstructed image was compared with the original image to calculate the loss function. To make the model use better generalization ability, the pixel value of each picture can be normalized using the following function [20].

$$T_{GI}(x, y) = \frac{T_{GI}(x, y) - T_{GImin}(x, y)}{T_{GImax}(x, y) - T_{GImin}(x, y)} \quad (3)$$

Mean square error (MSE) [23] is defined as Equation (4).

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \{I_{real}(x, y) - I_{rec}(x, y)\}^2 \quad (4)$$

m and n respectively represent the width and height of the image, and I_{real} and I_{rec} represent the pixel values at the corresponding positions of the original image and the reconstructed image. In other words, after the pixel points at the corresponding positions of the two images are subtracted, the results are finally added up. The smaller the MSE value is, the more similar the two images are. A more convincing evaluation index, The Structural Similarity Index (SSIM) [24] [25], is often used to make a more objective evaluation of the reconstructed image.

$$SSIM(x, y) = \frac{(2\gamma_x\gamma_y + \nu_1)(2\delta_{xy} + \nu_2)}{(\gamma_x^2 + \gamma_y^2 + \nu_1)(\delta_x^2 + \delta_y^2 + \nu_2)} \quad (5)$$

where

$$\gamma_x = \frac{1}{H * W} \sum_{i=1}^H \sum_{j=1}^W X(i, j) \quad (6)$$

$$\delta_x^2 = \frac{1}{H * W - 1} \sum_{i=1}^H \sum_{j=1}^W (X(i, j) - \gamma_x)^2 \quad (7)$$

$$\delta_{xy} = \frac{1}{H * W - 1} \sum_{i=1}^H \sum_{j=1}^W ((X(i, j) - \gamma_x)(Y(i, j) - \gamma_y)) \quad (8)$$

Compared with MSE, SSIM can better reveal the similarity between two images. The index value range is [-1, 1]. From the formula, it can be concluded

that SSIM = -1 indicates the two images are completely different, while SSIM = 1 means that the two images are identical. Therefore, the closer the SSIM value is to 1, the better the reconstruction effect of the model. γ_x represents the mean value in the x direction in the H*W area of the image, δ_x is the variance in the x direction in the H*W area, ν_1 and ν_2 represent the average pixel intensity of the two pictures, and δ_{xy} represents the x and y directions co-variance value.

There is another thing that cannot be ignored is Peak signal-to-noise ratio (PSNR) [25] [26], which is defined in Equation (9). In the field of image reconstruction, SSIM is the most commonly used and widely used evaluation index for researchers, which is based on the error between corresponding pixels in the image, or the corresponding evaluation of the image quality that is sensitive to errors. Because the calculation method does not consider the visual characteristics of the human eyes, the evaluation results are often inconsistent with human subjective feelings. (Human eyes are more sensitive to the low spatial frequency and the difference in brightness and contrast. The perception of a certain area will also be affected by its surrounding neighboring areas, etc.)

$$PSNR = 20 * \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \quad (9)$$

MAX_I is the maximum sampling value representing the colour of an image point. If each sampling point is represented by 8-bit binary, the maximum sampling value is 255. If the MSE is smaller, the PSNR will be larger and the quality of reconstructed image will be better.

Before the DAT-Net network was proposed, many scholars have put forward their GIDL models and verified the validity of their respective models [27] [28]. For example, using an image reconstruction model based on a residual network of up-and-down sampling [20], the process functions used to reconstruct the image and the associated parameter calculations can probably be represented as follows.

$$R_{up} = \begin{cases} T(x, y) - T_{GI}(x, y), & T(x, y) > T_{GI}(x, y) \\ 0, & T(x, y) \leq T_{GI}(x, y) \end{cases} \quad (10)$$

$$R_{down} = \begin{cases} T_{GI}(x, y) - T(x, y), & T_{GI}(x, y) > T(x, y) \\ 0, & T_{GI}(x, y) \leq T(x, y) \end{cases} \quad (11)$$

The process function of rebuilding the image can be revealed in Equation (12).

$$O(x, y) = T_{GI}(x, y) + R_{up}T_{GI}(x, y) - R_{down}T_{GI}(x, y) \quad (12)$$

where R_{up} and R_{down} respectively represent the image information obtained by up-and-down sampling, and $O(x, y)$ represents the reconstructed image.

When reproducing the DRU-Net, a shortcoming was found that the blur degree of the reconstructed image obtained is relatively large. The details of the reconstructed image are not processed well enough. The main reason is that this

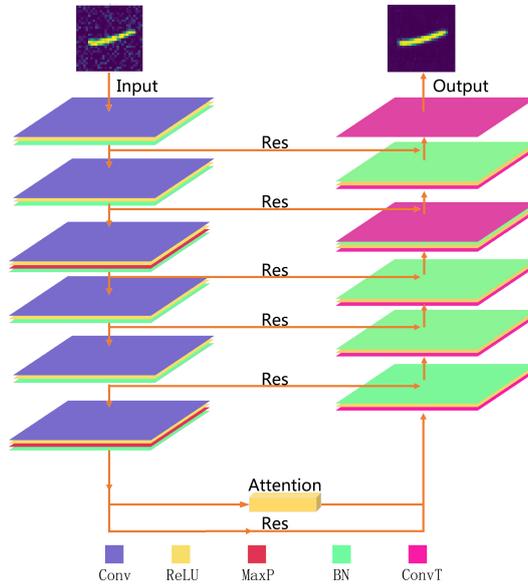


Fig. 2. The detailed structure diagram of the proposed DAT-Net. The left part is to process the image information of the light signal with noise, the bottom part is the attention mechanism for weighted feature extraction, and the right part uses transposed convolution to reconstruct the image. Res, which represents the output of the upper layer is directly input to the layer pointed by the arrow, which is the realization of the residual network. Attention, which represents the realization of the attention mechanism. The vertical arrow represents the image passing through the network layer in turn. Conv, convolution, ReLU, activation function, MaxP, maximum pooling (2x2), ConvT, transposed convolution.

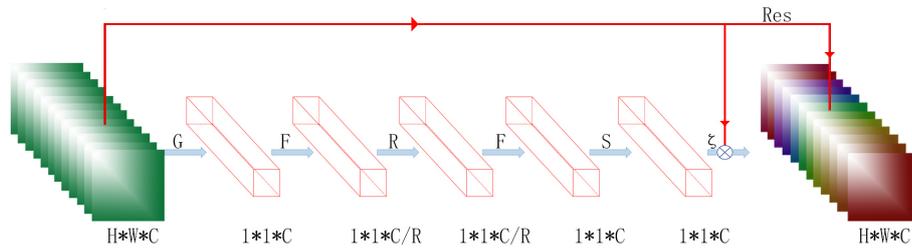


Fig. 3. Detailed structure diagram of the attention module. The feature map on the right is the feature map on the left after the attention weight weighting and residual calculation superimposed. G, global average pooling, F, fully connected layer, R, ReLU function, S, sigmoid function.

kind of models cannot focus on the image. Some information about the image is filtered, and the reconstructed image is not clear enough.

From the attention mechanism in the translation model, we can get enlightenment to solve the problem [29] [30]. After using the attention mechanism in the translation model, compared with the previous model, the model will focus on the current words to be translated, and the weight of the current words is greater. According to this idea, based on the residual network model, we make corresponding improvements to the model, add the channel attention mechanism and make the model focuses on the image itself during the training and reconstructing processes. The purpose is to make the weight coefficient of the image features larger and to make the weight coefficient of the noise smaller. In addition, the use of transposed convolution layers instead of up and down sampling layers can better filter noise. The reconstructed network model is proposed as shown in Figure 2. The architectural idea of the whole model comes from two models Res-Net [21] and SE-layer [31]. On the left side of the network layer, the characteristics of image fusion are extracted, and the attention module is applied to the high-level feature extraction [31] [32]. With the support of the residual network block [33], image characteristics can be learned thoroughly. At that time, more attention is placed on the image itself. The network on the right is built using multi-layer transposed convolution to achieve image reconstruction. In the process of GIDL, the corresponding function formula can be calculated as follows:

$$Res = \begin{cases} T(x, y) - T_{GI}(x, y), & T(x, y) > T_{GI}(x, y) \\ 0, & T(x, y) \leq T_{GI}(x, y) \end{cases} \quad (13)$$

z_a is the initial weight of feature layer obtained by global average pooling operation of feature map u_c . It is calculated by using Equation (15). The weight coefficient on different channels is denoted by w_a , $w_a \in (0, 1)$. w_a is calculated in Equation (??). To limit the complexity of the model, we parameterise the gating mechanism by using two fully connected layers, a dimensionality-reduction layer with parameters W_1 with reduction ratio r (r is a hyperparameter) and a dimensionality-increasing layer with parameters W_2 .

$$z_a = G_{ap}(u_c) = \frac{1}{H * W} \sum_{i=1}^H \sum_{j=1}^W u_c(i, j), u_c \in F_m^{H * W * C} \quad (14)$$

$$w_a = F_{frfs}(z_a, W) = \sigma(g(z_a * W)) = \sigma(W_2 \delta(W_1 * z_a)), W_1 \in R^{\frac{C}{r} * C}, W_2 \in R^{C * \frac{C}{r}} \quad (15)$$

After the weight of each channel is calculated, w_a is weighted on the feature map u_c , which is expressed by $\zeta_*^+(\cdot)$ in Equation (16). Here the residual block is used in this model, and parameter R that the model needs to learn is derived, see Equation (17).

$$A_{GI} = \sum_{i=1}^N \zeta_*^+(w_a^i, u_c^i) \quad (16)$$

$$R = \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^N L \left\{ \operatorname{Res}^i(x, y), R_{\theta} \{ T_{GI}^i(x, y), A_{GI}^i(x, y) \} \right\} \quad (17)$$

Finally, the image is reconstructed

$$O(x, y) = T_{GI}(x, y) + R \{ T_{GI}(x, y) + A_{GI}(x, y) \} \quad (18)$$

This model involves attention mechanism module and residual module. It learns most of features of the image with the attention mechanism module. weight coefficient refers to the weight of each layer of feature map. By learning important information of the feature map automatically, weight coefficient of each feature will increase accordingly. After the weights of different feature layers are calculated, weight coefficient is applied to feature map. The weight ratio of each feature layer in the original image can be calculated. This model learns and optimizes its parameters according to the weight difference between different feature layers. The residual network used to retain feature information of the original image to the maximum extent together with some noises carried by the image, because it calculates every feature in the same way and doesn't focus on features that need to be retained. Here the attention mechanism makes up for this defect, and helps focus on features other than noises. The purpose of applying transposed convolution in the model is to reduce the number of parameters that the model needs to learn, realize parameter sharing, and realize image reconstruction efficiently.

Results

Under the same Nyquist sampling rate ($\beta = 100\%$), we use different decibels of Gaussian noise to reconstruct the image. The aim is to verify that the proposed model not only reconstructs the image with high quality after adding the attention module, but also pays more attention to the details of the image during the reconstruction process.

To make the model have a good generalization ability, the data set of the training model is 48000 images with Gaussian noise of 150dB, and the original images are used as labels through the matching algorithm. The proposed method is implemented in the PyTorch (1.6) framework and Python (3.7). With the support of batch processing and GPU, training process of the model is significantly accelerated and time consumption is effectively reduced. The gradient descent method is Adam, the initial learning rate is 0.0003, and the loss function of the model is calculated by using the `nn.MSELoss()`. Three representative index functions are used to evaluate the ability of model reconstruction.

Figure 4 shows the comparison of the test results of the two GIDL models for reconstructed images. Color images facilitate the observation of image destruction under the influence of noise, and to view the reconstruction effect comparison between DRU-Net and DAT-Net based on the attention mechanism. In Figure 4, the first column shows the original image, the second column is

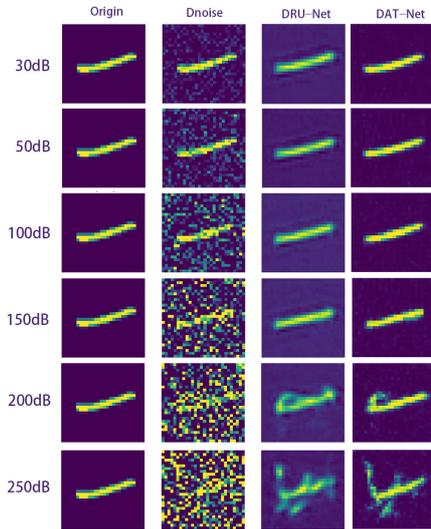


Fig. 4. Comparison of DAT-Net and DRU-Net reconstructed image quality different Gaussian noises.

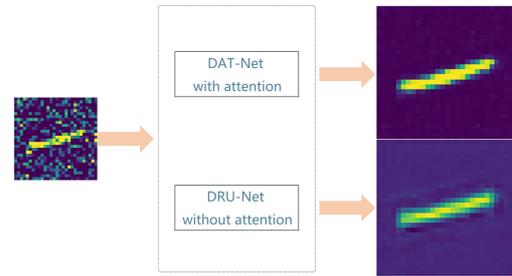


Fig. 5. $\beta = 1$, Enlarged details of the reconstructed images.

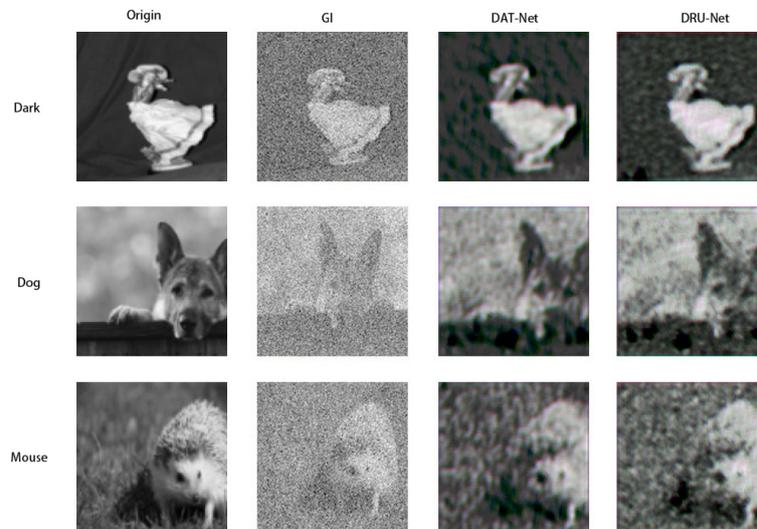


Fig. 6. A comparison between the results of GI, DAT-Net and DRU-Net.

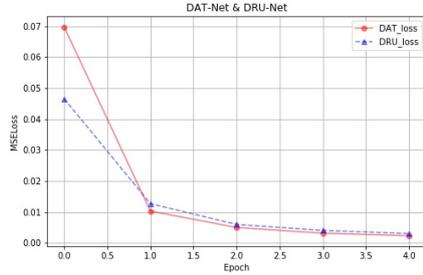


Fig. 7. The loss function values of DAT-Net and DRU-Net with the same training number.

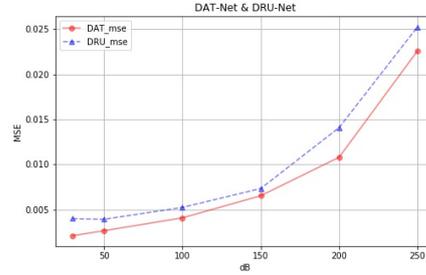


Fig. 8. Comparison of MSE between DAT-Net and DRU-Net.

the image obtained under the influence of noise, the third column is the image reconstructed by the DRU-Net network, and the fourth column is the image reconstructed by the DAT-Net. From the effect of the reconstructed image in the last two columns, it can be seen that the image obtained by using the attention mechanism model will be clearer in detail processing, indicating that the network model under the attention mechanism will pay more attention on the image itself. The weight of the noise is smaller, so when the image is reconstructed, the noise will be greatly reduced. With the continuous increase of noise, the degree of image destruction becomes more intense. It can be seen that the reconstruction effect of the image is also weakened accordingly. However, DAT-Net still performs quite well and focuses more on the image itself.

Figure 5 represents the enlarged details of the reconstructed images obtained by DAT-Net and DRU-Net. It can be observed in the magnified image that the model with attention mechanism can retain more image details. Compared with DRU-Net, the reconstructed image of our proposed model is significantly improved.

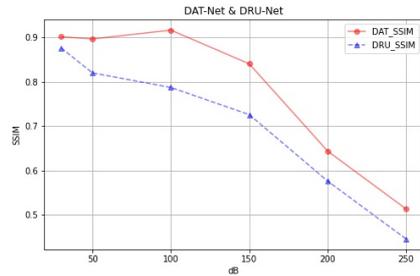


Fig. 9. Comparison of SSIM between DAT-Net and DRU-Net.

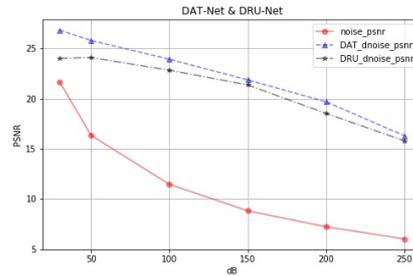


Fig. 10. Comparison of PSNR between DAT-Net and DRU-Net.

Figure 6 shows the experimental results with sampling rate $\beta = 100\%$. It can be concluded from the figure that the two models have certain differences in the reconstruction of quantum images. It is obvious that the reconstruction effect of our proposed model is better than that of the DRU-Net model.

The line graph in Figure 7 shows a comparison of the MSE Loss values of two models with the same training number (Epoch = 5). At the end of the first training, the MSE Loss value of DAT-Net is greater than that of DRU-Net, but from the second training, the MSE Loss value of DAT-Net is significantly lower than that of DRU-Net. According to the trend of the curve in the figure, in the training process of DAT-Net network, with the support of attention mechanism, the model focuses on the overall contour and details of the image when denoising the image. When the model is backpropagated, it will fully combine the weight value of the attention mechanism to optimize the parameters, pay more attentions to the characteristics of the image itself, and extract more image details.

In Figure 8, traditional MSE functions are used to evaluate the similarity between the reconstructed image and the original image. The figure shows that the MSE value of the DAT-Net is less than the DRU-Net, which has certain defects. In order to make the model more convincing, the SSIM indicator is used to evaluate the pros and cons between the two models. It can be concluded from Figure 9 that the SSIM score of DAT-Net is higher than that of DRU-Net network, which indicates that the network model using attention mechanism is much better than the model using only residual network.

In Figure 10, the PSNR of noises with different decibel values is shown together with the PSNR of the reconstructed image obtained by two different network models. It can be seen that the reconstruction effect of DAT-Net network based on the attention mechanism is better than that of DRU-Net.

Conclusion

In this work, we propose a method for denoising and ghost imaging based on depth residual network with attention mechanism. It has better generalization ability for images with random Gaussian noise. With attention mechanism, the proposed model focuses on the original image as much as possible in the process of reconstructing the image. Experiments proved that the comprehensive performance of DAT-Net is better than DRU-Net on MSE, SSIM and PSNR, and the quality of reconstructed images was significantly improved. It is foreseeable that with the support of the attention mechanism, new possibilities are opened up for the future application of biomedical images, such as membrane computing and bioinformatics image processing [34], and it may also be helpful for remote sensing and the detection of moving objects.

Acknowledgements

This work was supported by the National Natural Science Foundation of China [grant numbers 61872325]; the Fundamental Research Funds for the Central Universities [grant number 2652019028].

References

1. Pittman, T.B., Shih, Y., Strekalov, D., Sergienko, A.V.: Optical imaging by means of two-photon quantum entanglement. *Physical Review A* **52**(5), R3429 (1995)
2. Cheng, J., Han, S.: Incoherent coincidence imaging and its applicability in x-ray diffraction. *Physical review letters* **92**(9), 093903 (2004)
3. Gatti, A., Brambilla, E., Bache, M., Lugiato, L.A.: Correlated imaging, quantum and classical. *Physical Review A* **70**(1), 013802 (2004)
4. Valencia, A., Scarcelli, G., D'Angelo, M., Shih, Y.: Two-photon imaging with thermal light. *Physical review letters* **94**(6), 063601 (2005)
5. Shapiro, J.H.: Computational ghost imaging. *Physical Review A* **78**(6), 061802 (2008)
6. Yu, H., Li, E., Gong, W., Han, S.: Structured image reconstruction for three-dimensional ghost imaging lidar. *Optics express* **23**(11), 14541–14551 (2015)
7. Gong, W., Zhao, C., Yu, H., Chen, M., Xu, W., Han, S.: Three-dimensional ghost imaging lidar via sparsity constraint. *Scientific reports* **6**, 26133 (2016)
8. Liansheng, S., Yin, C., Ailing, T., Asundi, A.K.: An optical watermarking scheme with two-layer framework based on computational ghost imaging. *Optics and Lasers in Engineering* **107**, 38–45 (2018)
9. Wang, S., Meng, X., Yin, Y., Wang, Y., Yang, X., Zhang, X., Peng, X., He, W., Dong, G., Chen, H.: Optical image watermarking based on singular value decomposition ghost imaging and lifting wavelet transform. *Optics and Lasers in Engineering* **114**, 76–82 (2019)
10. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020)
11. Otter, D.W., Medina, J.R., Kalita, J.K.: A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems* (2020)
12. Guo, J., He, H., He, T., Lausen, L., Li, M., Lin, H., Shi, X., Wang, C., Xie, J., Zha, S., et al.: Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing. *Journal of Machine Learning Research* **21**(23), 1–7 (2020)
13. Huang, T., Zhang, Q., Liu, J., Hou, R., Wang, X., Li, Y.: Adversarial attacks on deep-learning-based sar image target recognition. *Journal of Network and Computer Applications* p. 102632 (2020)
14. Yoon, S., Kim, M., Jang, M., Choi, Y., Choi, W., Kang, S., Choi, W.: Deep optical imaging within complex scattering media. *Nature Reviews Physics* pp. 1–18 (2020)
15. Lyu, M., Wang, W., Wang, H., Wang, H., Li, G., Chen, N., Situ, G.: Deep-learning-based ghost imaging. *Scientific reports* **7**(1), 1–6 (2017)
16. Wang, F., Wang, H., Wang, H., Li, G., Situ, G.: Learning from simulation: An end-to-end deep-learning approach for computational ghost imaging. *Optics express* **27**(18), 25560–25572 (2019)

17. Antun, V., Renna, F., Poon, C., Adcock, B., Hansen, A.C.: On instabilities of deep learning in image reconstruction and the potential costs of ai. *Proceedings of the National Academy of Sciences* (2020)
18. Sheng-Mei, Z., Peng, Z.: Correspondence normalized ghost imaging on compressive sensing. *Chinese Physics B* **23**(5), 054203 (2014)
19. Wu, H., Wang, R., Zhao, G., Xiao, H., Liang, J., Wang, D., Tian, X., Cheng, L., Zhang, X.: Deep-learning denoising computational ghost imaging. *Optics and Lasers in Engineering* **134**, 106183 (2020)
20. Bian, T., Yi, Y., Hu, J., Zhang, Y., Wang, Y., Gao, L.: A residual-based deep learning approach for ghost imaging. *Scientific Reports* **10**(1), 1–8 (2020)
21. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 770–778 (2016)
22. Wang, F., Jiang, M., Qian, C., Yang, S., Li, C., Zhang, H., Wang, X., Tang, X.: Residual attention network for image classification. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 3156–3164 (2017)
23. Zhang, L., Li, X., Zhang, D.: Image denoising and zooming under the linear minimum mean square-error estimation framework. *IET Image Processing* **6**(3), 273–283 (2012)
24. Ndajah, P., Kikuchi, H., Yukawa, M., Watanabe, H., Muramatsu, S.: Ssim image quality metric for denoised images. In: *Proc. 3rd WSEAS Int. Conf. on Visualization, Imaging and Simulation*. pp. 53–58 (2010)
25. Hore, A., Ziou, D.: Image quality metrics: Psnr vs. ssim. In: *2010 20th international conference on pattern recognition*. pp. 2366–2369. IEEE (2010)
26. Liu, H.C., Yang, B., Guo, Q., Shi, J., Guan, C., Zheng, G., Mühlenbernd, H., Li, G., Zentgraf, T., Zhang, S.: Single-pixel computational ghost imaging with helicity-dependent metasurface hologram. *Science advances* **3**(9), e1701477 (2017)
27. Tian, C., Xu, Y., Fei, L., Yan, K.: Deep learning for image denoising: a survey. In: *International Conference on Genetic and Evolutionary Computing*. pp. 563–572. Springer (2018)
28. Jin, K.H., McCann, M.T., Froustey, E., Unser, M.: Deep convolutional neural network for inverse problems in imaging. *IEEE Transactions on Image Processing* **26**(9), 4509–4522 (2017)
29. Luong, M.T., Pham, H., Manning, C.D.: Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015)
30. Yamada, M., DIEKFUSS, J.A., RAISBECK, L.D.: Motor behavior literature fails to translate: A preliminary investigation into coaching and focus of attention in recreational distance runners. *International Journal of Exercise Science* **13**(5), 789 (2020)
31. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 7132–7141 (2018)
32. Han, Y., Wei, C., Zhou, R., Hong, Z., Zhang, Y., Yang, S.: Combining 3d-cnn and squeeze-and-excitation networks for remote sensing sea ice image classification. *Mathematical Problems in Engineering* **2020** (2020)
33. Menna, F., Nocerino, E., Ural, S., Gruen, A.: Mitigating image residuals systematic patterns in underwater photogrammetry. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* **43**, 977–984 (2020)
34. Yu, S., Su, T., Wu, H., Liu, S., Wang, D., Zhao, T., Jin, Z., Du, W., Zhu, M.J., Chua, S.L., et al.: Pslg, a self-produced glycosyl hydrolase, triggers biofilm disassembly by disrupting exopolysaccharide matrix. *Cell research* **25**(12), 1352–1367 (2015)

A new global harmony search algorithm for solving numerical optimization^{*}

Tianqi Liu^[0000-0001-5826-8702], Hua Yang^[0000-0001-6845-6125], Jing Yu^[0000-0002-9075-1734], and Kang
Zhou^[0000-0002-0205-768X] ^{**}

School of Mathematics and Computer. Wuhan Polytechnic University, Wuhan,430048,China
ltqsnail@126.com

Abstract. To overcome the shortcomings of the existing harmony search(HS)algorithm and improve its effectiveness and efficiency, this paper proposes an improved intersecting mutation global harmony search algorithm to solve the numerical function optimization problem. The improved intersecting mutation global harmony search algorithm proposed in this paper contains a new impromptu scheme, which uses the best and worst harmony cross-mutation to generate new harmony to enhance its local search and global search capabilities. It also contains a unique selection update mechanism to improve the new harmony selection update process to increase the diversity of the population and increase its convergence speed. Among them, through dynamically adjusted parameters, the breadth of the search is increased so that the algorithm can balance the development and exploration in the entire search process.To test the performance of the proposed algorithm, the classical CEC test function is used to carry out a simulation experiment. Finally, the experimental data show that:compared with the existing five harmony search algorithms, the improved intersecting mutation global harmony search algorithm has a faster convergence speed and excellent ability to deal with complex high-dimensional optimization problems.

Keywords: harmony search algorithm, intersecting mutation, numerical function optimization problem, best and worst harmony, selection update mechanism, CEC test function

1 Introduction

Due to the inherent optimization problems, which are complex, swarm intelligence optimization algorithms [1] can be an appropriate choice because they are initialized based on some random solutions and updated based on some simple rules at each iteration. These rules are used to develop an optimization algorithm inspired by the environment. For example, Yang et al.[2, 3] proposed the bat algorithm in 2010, which controls the dynamic behavior of the bat colony based on the echolocation behavior of microbes. Wu et al.[4] propose the Wolf pack algorithm (WPA) in 2014 inspired by wolf pack prey activity, the production rules of the Wolf King, and the survival mechanism of the fittest. Eberhart and Dr. Kennedy[5] proposed the Particle Group algorithm (PSO) in 1995 by simulating migration and clustering behavior during bird colony foraging. Yang et al.[6] proposed the Firefly Algorithm (FA) in 2013 by affecting the flickering and courtship behavior of fireflies; Xue et al.[7] proposed the Sparrow Search Algorithm (SSA) in 2020, which was inspired by the foraging behavior and anti-predation behavior of sparrows. Over the years, scholars have continuously improved and improved these Swarm intelligence optimization algorithms to play an essential role in most industries, which are highly efficient in solving real-life optimization problems[8–12].

Harmony Search Algorithm [13](HSA) is also a swarm intelligence optimization algorithm, a novel intelligent optimization algorithm [14] proposed by Geem et al. The algorithm simulates the principle of music performance. It evaluates each set of harmonies played by the band to obtain the optimal solution vector of the optimization problem. Experiment results show that the HS algorithms are robust, fast, and convenient in iterative computation. Many researchers have modified the classic version of HS to enhance its performance [15]. Using improved HS algorithms to solve different engineering problems [16–21]. All have achieved good results.

The HS algorithm has three essential parameters: harmony memory consideration rate (HMCR), Pitch adjustment rate (PAR), and bandwidth (BW). The original HS algorithm does not have outstanding results from search numerical

^{*} Supported by School of Mathematics and Computer. Wuhan Polytechnic University

^{**} Yang Hua:22669594@qq.com, Liu Tianqi:ltqsnail@126.com

optimization applications, so some scholars improve these three parameters. Therefore, the adaptive HS of the improved search strategy is an improvement direction used to improve the search efficiency. The improved HS algorithm is also applied to real-world optimization problems, and variants have been revised from multiple angles [22, 23]. One of them is an improved harmony search algorithm (IHA)[24], a representative first improvement.

On the other hand, the HS algorithm has a harmonic location selection update mechanism. The initial selection mechanism randomly selects within the harmonies library or generates the solution set outside the harmonies library to choose the resulting new harmonies. Since the initial HS algorithm search and development performance is not perfect, there are many improvements in this area, such as global optimal harmonies search (GHS) [25] algorithm, Novel Global Harmonic Search (NGHS), Intersect mutation global harmony search algorithm (IMGHSA) [26]. They are one of the HS variants, with the successful completion of many reliabilities analysis experiments. Based on the solution to the numerical optimization algorithm, the paper proposed an improved intersect mutation global harmony searches algorithm (IIMGHSA), Introducing an adaptive parameter, novel mutation method, and harmonic selection update mechanism in the HS process to prove that the improved steps can improve the performance of the original algorithm. Therefore, IIMGHSA is applied to solve 15 benchmark functions, and the results obtained are compared with the results of several improved HS algorithm processing benchmark functions. Select these five classic harmony search algorithms: Intersecting Mutation Global Harmony Search Algorithm (IMGHSA), Harmony Search Algorithm with Chaos (HSCH) [27], New Global Harmonic Search (NGHS) [28], Harmony Search with Worst and Best (HSWB) [29] and Efficient Global Harmony Search (EGHS) [30, 31]. Finally, the pros and cons of the algorithm are tested through simulation experiments.

When the existing harmony search algorithms deal with unconstrained optimization problems, the results obtained are not satisfactory, and some of the harmony search algorithms run for a long time. To solve such issues, this paper proposes three improvements programs. HSA is improved on an intersect mutation global harmony searches algorithm (IMGHSA) by combining these three improvements: adaptive parameters, new loop selection mechanism, and cross-mutation. And connecting the advantage points of other harmonic search algorithms, such as the introduction of the GNHS algorithm to avoid the advantages of premature convergence, the harmony vector in the initial harmonic memory (HM) is divided into a better part and a worse part, which will effectively improve search efficiency and provide space range of efficient search. In considering the relationship of equilibrium algorithm exploration and development, this paper introduces adaptive parameters. It adds cross mutation operators in the search process of the algorithm, which makes the algorithm perform a global search of the early search process and perform a local optimization in the later search period and increase the diversity of the population. An out-of-cycle selection update mechanism can optimize the population diversity and increase the convergence speed in the last stage of the algorithm.

The overall purpose is to test the proposed improved intersecting mutation global harmony search algorithm (IIMGHSA) and other improved harmony search algorithms by using 15 well-known benchmark optimization functions of the International Conference on Evolutionary Computing (CEC), which proves that IIMGHSA has better optimization capabilities.

2 The original Harmony Search Algorithm

2.1 Classical Harmonic Search algorithm(CHSA)

Scholars such as Z.W.Geem et al., when listening to music performance, according to the musicians repeatedly adjust the intonation of the instrument, and finally, form a beautiful harmony state inspiration, thus proposed the harmony search (HS) algorithm. The optimization operation roughly assumes the existing objective function $f(x)$ that needs to be optimized, where: $x = [x_1^i, x_2^i, \dots, x_n^i]$, Then x can be regarded as the Harmony combination of n musicians, In the process of adjustment, different harmony will be produced $x_n^i (i = 1, 2, 3, \dots, n)$, the objective function $f(x)$ is used as the evaluation standard for the pitch, and the musicians continuously modify the pitch n_n^i (Explore the iterative process) until a beautiful harmony is formed (Reach the evaluation criteria or the maximum number of iterations).

The algorithm steps of the HS algorithm:

step1. Specifies the problem's optimization direction and initialization parameters according to the actual situation, then encodes a set of variables in the established mathematical model as a set of tones that make up the harmony.

Step2. Initialize the harmony memory bank and calculate the initial fitness value:

Generating random initial solution:

$$X^i = LB + (UB - LB) * rand(0, 1), i \in (1, HMS) \quad (1)$$

Initialize Harmony memory:

$$HM = \begin{bmatrix} X^1 \\ X^2 \\ \vdots \\ X^{HMS} \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_N^1 \\ x_1^2 & x_2^2 & \dots & x_N^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{HMS} & x_2^{HMS} & \dots & x_N^{HMS} \end{bmatrix} \quad (2)$$

Target function optimization value under the initial solution set condition:

$$f(X^i) = \begin{bmatrix} f(X^1) \\ f(X^2) \\ \vdots \\ f(X^{HMS}) \end{bmatrix} \quad (3)$$

Step3.The improvisation iteratively selects a new harmony, and then a new harmony vector is generated: judge according to the HMCR selection probability of the harmony library and uses the *rand* function to create a number randomly r_1 between 0 and 1 to compare with HMCR. If r_1 is more significant than HMCR value randomly selects a harmony in the solution space, otherwise randomly selects a harmony from the harmony library, and then when r_1 is less than HMCR, the *rand* function is used to generate $r_2 \in (0, 1)$ randomly. If the pitch adjustment rate *PAR* is more significant than r_2 , the selected harmony will be disturbed. Otherwise, it will be directly used as a new harmony. The specific operations are as follows:

$$x(i, j)^{new} = \begin{cases} HM(i, j) & , \text{if } r_1 < HMCR \\ LB_j + (UB_j - LB_j) * rand(0, 1) & , \text{otherwise} \end{cases} \quad (4)$$

When PAR is greater than r_2 , the specific operations are as follows:

$$X_{i,j} = X_{i,j} \mp rand(1, j) * BW \quad (5)$$

step4.Update Harmony Library: Judge the new harmony selected in step3, compare the generated new harmony with the worst harmony in the harmony library, and let the harmony that can make the objective function take a better result instead of the position of the worst harmony in the harmony library.

The specific operations are as follows(X_{worst} is the worst-adapted harmony in the harmony library):

$$X_{worst} = \begin{cases} X^{new}, & \text{if } f(X^{new}) < f(X_{worst}) \\ X_{worst} & , \text{otherwise} \end{cases} \quad (6)$$

Step5.Determines whether the maximum value of the iteration is reached and stops. Otherwise, repeat steps step3 and step4, step5 until the iteration ends.

The main steps of the HS algorithm are shown in the following pseudo-code:

[1] **Algorithm 1** HS algorithm

- 1: Initializing HS algorithm's parameters.
- 2: Initialization of the harmony memory (HM) and compute the objective function.

```

3: while  $t < T_{max}$  do
4:   for  $i \in [HMS, 1]$  do
5:     for  $j \in [1, N]$  do
6:       if  $r_1 < HMCR$  then
7:          $x_{i,j}^{new} = HM_{i,j}$ 
8:         if  $r_2 < PAR$  then
9:            $x_{i,j}^{new} = x_{i,j}^{new} \hat{A} \hat{S} rand * BW$ 
10:        endif
11:       else
12:          $x_{i,j}^{new} = LB_j + (UB_j - LB_j) * rand(0, 1)$ 
13:       endif
14:     endfor
15:     Evaluate the new harmony vector.
16:     Update the harmony memory (HM).
17:   endfor
18: endwhile

```

Among them: UB and LB are the upper and lower limits of the search space. The algorithm will search for optimization within the range. When $x_{i,j}^{new}$ is not in the range, it needs to perform out-of-bounds processing. When $x_{i,j}^{new}$ is less than LB_j , make $x_{i,j}^{new}$ equal to LB_j , or when $x_{i,j}^{new}$ is greater than UB_j , make $x_{i,j}^{new}$ equal to UB_j .

3 Intersect mutation global harmony search algorithm (IMGHSA)

The intersecting mutation global harmony search algorithm is a variant of the global search harmony algorithm, which abandons the GHS algorithm for random harmonic selection from the harmonic library in the harmonic update iteration process but instead chooses to select the optimal worst harmonic for computational iteration, which effectively improves the search efficiency. Combining the intersection mutation operation (IMO) and the global harmonic algorithm effectively improves the position update strategy, increases the small probability mutation strategy, facilitates the global population search and local search, and more effectively weighs the balance between the algorithm development and exploration. The specific steps are as follows:

step1. Initialize the required parameters of the algorithm.

step2. Initialization establishes a harmonic memory library.

step3-step4. Obtain new harmony through location update and small probability changes, and update the harmony library. The specific pseudo code is as follows:

[2] **Algorithm 2** IMGHSA-step3-step4:pseudo code

```

1: while  $t < T_{max}$  do
2:   for  $i \in [HMS, 1]$  do
3:     for  $j \in [1, N]$  do
4:       if  $rand < HMCR$  then
5:          $xr = 2 \times x_{best_{i,j}} - x_{worst_{i,j}}$ 
6:         if  $xr > UB_j$  then
7:            $xr = UB_j$ 
8:         endif
9:         if  $xr < LB_j$  then
10:           $xr = LB_j$ 
11:        endif
12:         $x_{i,j}^{new} = x_{worst_{i,j}} + (xr - x_{worst_{i,j}}) \times rand(0, 1)$ 

```

```

13:     if  $rand < P_m$  then
14:          $x_{i,j}^{new} = LB_j + (UB_j - LB_j) \times rand$ 
15:     endif
16:     else
17:          $x_{i,j}^{new} = \mu_1 \times x_{worst_{i,j}} + \mu_2 \times x_{best_{i,j}}$ 
18:         if  $rand < PAR$  then
19:              $x_{i,j}^{new} = x_{i,j}^{new} \hat{A} \hat{s} rand * BW$ 
20:         endif
21:     endif
22: endfor
23: Evaluate the new harmony vector.
24: Update the harmony memory (HM).
25: endfor
26: endwhile

```

Among them, x_j^{new} is the variable of the j_{th} dimension of the new harmony, x_{worst_j} and x_{best_j} represent the worst and best harmony in the current harmony library, and the positions are the j row and the i row of the harmony library matrix, $rand(0, 1)$ is a random value from 0 to 1. LB is the lower limit of the problem, and UB is the upper limit of the problem.

step5: If the maximum number of iterations is reached, the algorithm terminates and outputs the optimal value. Otherwise, step3, step4, step5 are executed in a loop until the termination condition is reached.

4 Improved harmonic search algorithm

4.1 An improved intersecting mutation global harmony search algorithm(IIMGHSA)

While combining the advantages of IMGHSA, adaptive parameters are added to the IMGHS algorithm so that the algorithm can be more rationally developed and explored during operation. There can be pretty sizeable adaptive parameter values in the early stage, which is more conducive to the global search of the algorithm. The late adaptive parameter value is small, which is conducive to the local optimization of the algorithm. In addition to the cyclic selection update mechanism, a new harmonic selection update mechanism has also been added. It is conducive to rich population diversity and speeds up the convergence speed of the algorithm. In addition, based on IMO, different mutation operators adopt different degrees of "push and pull" for the new harmony. After many tests, good results have been achieved. Since the IIMGHS algorithm is improved based on the IMGHS algorithm, the steps of the two algorithms are roughly the same, so the improvement points are pointed out below, and the operation steps of step3-step4 are improved:

step3-step4: Operation steps Improve operations such as the following pseudo-code:

[3] **Algorithm 3** IIMGHSA-step3-step4:pseudo code

```

1: while  $t < T_{max}$  do
2:     for  $i \in [HMS, 1]$  do
3:         if  $rand < HMCR$  then
4:             for  $j \in [1, N]$  do
5:                  $xr = 2 \times x_{best_{i,j}} - x_{worst_{i,j}}$ 
6:                 if  $xr > UB_j$  then
7:                      $xr = UB_j$ 
8:                 endif
9:                 if  $xr < LB_j$  then
10:                     $xr = LB_j$ 

```

```

11:      endif
12:       $x_{i,j}^{new} = x_{worst_{i,j}} + (x_r - x_{worst_{i,j}}) \times rand(0, 1)$ 
13:      if  $rand < P_m$  then
14:           $x_{i,j}^{new} = LB_j + (UB_j - LB_j) \times rand$ 
15:      endif
16:      endfor
17:  else
18:       $x_{i,j}^{new} = \mu_1 \times x_{worst_{i,j}} + \mu_2 \times x_{best_{i,j}}$ 
19:      if  $rand < PAR$  then
20:           $x_{i,j}^{new} = x_{i,j}^{new} \hat{A} \times rand * BW$ 
21:      endif
22:      if  $x_j^{new} > UB_j$  then
23:           $x_j^{new} = UB_j$ 
24:      endif
25:      if  $x_j^{new} < LB_j$  then
26:           $x_j^{new} = LB_j$ 
27:      endif
28:  endif
29:  Evaluate the new harmony vector.
30:  Update the harmony memory (HM).
31: endfor
32: endwhile

```

Among them: According to the analysis of literature[26], μ_1 and μ_2 are two parameters that are adjusted artificially. These parameters must be selected to meet the constraint of $\mu_1 + \mu_2 = 1$. According to our test, the values $\mu_1 = \frac{1}{3}$ and $\mu_2 = \frac{2}{3}$ have a significant influence on the algorithm.

$$HMCR = HMCR_{max} - (HMCR_{max} - HMCR_{min}) * \frac{j}{T_{max}} \quad (7)$$

$$BW = BW_{max} - (BW_{max} - BW_{min}) * \frac{j}{T_{max}} \quad (8)$$

step5: Determine whether the termination conditions are met, otherwise cycle step3, step4, step5 until the termination condition is reached.

5 Simulation experiment

5.1 Operating environment

The experimental environment is shown in the following table:

Table 1. Operating environment parameters

Device Name	DESKTOP-856VDEJ
Processor	Intel(R) Core(TM) i5-5200U CPU @2.20GHz 2.19 GHz
System type	64-bit OS, x64-based processor
Version	Windows 10 Professional Edition
Running tool	MATLAB R2018b

5.2 Specific experimental operation and analysis

In this research work, simulation and experimental results were obtained using MATLAB software. In this paper, to verify the optimization performance of the IIMGHS algorithm, 15 CEC benchmark functions in the literature [26][32][33] are selected. To set the basic parameters of the algorithm, choose the maximum number of iterations and the considered harmony library size to be 1000 and 30, respectively, and set the dimensionality to 30-dimensional and 50-dimensional, and conduct experiments, respectively. In addition, each optimization algorithm runs independently 30 times in both dimensions. The parameter description in this article is shown in Table 2. Various improved harmony search algorithms and IIMGHS algorithm operating parameters are shown in Table 3, 15 benchmark functions are shown in Table 4. The results of the IIMGHS algorithm and different HS algorithms are recorded in Table 5, where $f_{best_{mean}}$ represents the average of the 30 optimal fitness values after the algorithm has been run 30 times, $Time$ represents the average time for different algorithms to run 30 times, and $f_{best_{min}}$ represents the optimal average fitness value after 30 runs of the algorithm, $f_{best_{max}}$ represents the worst average fitness value after 30 runs of the algorithm, $f_{best_{std}}$ represents the standard deviation of the 30 optimal fitness values after 30 runs. The iterative graphs and box plots of different benchmark functions are shown in Fig.1-Fig.8.

The settings of these algorithm parameters come from Gholami J et al. [26], YONG et al. [28], Zou et al. [30], and are uniformly modified to ensure the accuracy of the experiment. The specific content is shown in Table 2 and Table 3:

Table 2. Symbol Description

NGHS	Novel global harmony search algorithm
HSCH	Harmony Search Algorithm with Chaos
HSWB	Harmony Search with Worst and Best
EGHS	Effective global harmony search algorithm
IMGHSA	Intersect mutation global harmony search algorithm
IIMGHSA	Improved intersecting mutation global harmony search algorithm
HMCR	Harmony memory considering rate
HMCRMAX	Maximum Harmony Memory Consideration Rate
HMCRMIN	Minimum Harmony Memory Consideration Rate
P_m	Mutation probability
PAR	Pitch adjusting rate
BW	Bandwidth
BW_{max}	Maximum bandwidth
BW_{min}	Minimum bandwidth
LUP	location updating probability
u_1	Intersecting mutation factor 1
u_2	Intersecting mutation factor 2
Dim	Dimension

Table 3. Various improved harmony search algorithm parameters

Algorithm category	Parameters
NGHS	$P_m = 0.005$
HSCH	$HMCR = 0.9, PAR = 0.3, BW = 0.01$
HSWB	$HMCR = 0.9, PAR = 0.3, BW = 0.01$
EGHS	$LUP = 0.9$
IMGHSA	$HMCR = 0.9, PAR = 0.3, BW = 0.01, P_m = 0.005$
IIMGHSA	$HMCR_{MAX} = 0.95, HMCR_{MIN} = 0.8, P_m = 0.005, LUP = 0.9,$ $BW_{MAX} = \frac{UB+UL}{20}, BW_{MIN} = 0.001$

Table 4: 15 benchmark functions

Label	Test function	Iterations	range	Dimension
F1	$f_1(x) = x_1^2 + 10^6 \sum_{i=2}^D x_i^2$	1000	$[-10, 10]^d$	30(50)
F2	$f_2(x) = \sum_{i=1}^D x_i ^{i+1}$	1000	$[-5, 10]^d$	30(50)
F3	$f_3(x) = \sum_{i=1}^D x_i^2 + \left(\sum_{i=1}^D 0.5x_i\right)^2 + \left(\sum_{i=1}^D 0.5x_i\right)^4$	1000	$[-5, 10]^d$	30(50)
F4	$f_4(x) = \sum_{i=1}^{D-1} \left(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right)$	1000	$[-10, 10]^d$	30(50)
F5	$f_5(x) = \frac{\pi}{D} \left\{10 \sin(\pi y_1) + \sum_{i=1}^{D-1} (y_i - 1)^2 [1 + 10 \sin^2(\pi y_{i+1})]\right\}$ $+ \frac{\pi}{D} \left\{(y_D - 1)^2\right\} + \sum_{i=1}^D u(x_i, 10, 100, 4); \quad y_i = 1 + \frac{x_{i+1}}{4}$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{cases}$	1000	$[-50, 50]^d$	30(50)
F6	$f_6(x) = \frac{1}{4000} \sum_{i=1}^D x_i^2 - \prod_{i=1}^D \cos\left(\frac{x_i^2}{\sqrt{i}}\right) + 1$	1000	$[-600, 600]^d$	30(50)
F7	$f_7(x) = \sum_{i=1}^D ix_i^4 + \text{random}(0, 1)$	1000	$[-1.28, 1.28]^d$	30(50)
F8	$f_8(x) = -\exp\left(-0.5 \sum_{i=1}^D x_i^2\right)$	1000	$[-1, 1]^d$	30(50)
F9	$f_9(x) = \sum_{i=1}^D x_i^2 - 450$	1000	$[-100, 100]^d$	30(50)
F10	$f_{10}(x) = \sum_{i=1}^D (x_i - 10 \cos(2\pi x_i) + 10)$	1000	$[-5.12, 5.12]^d$	30(50)
F11	$f_{11}(x) = \frac{1}{n} \sum_{i=1}^D (x_i^4 - 16x_i^2 + 5x_i)$	1000	$[-5, 5]^d$	30(50)
F12	$f_{12}(x) = \sum_{i=2}^D i(2x_i^2 - x_{i-1}^2) - (x_1 - 1)^2$ $0.4 \cos(4\pi x_{i+1}) + 0.7]$	1000	$[-10, 10]^d$	30(50)
F13	$f_{13} = \left \left(\sum_{i=1}^D x_i^2\right)^2 - \left(\sum_{i=1}^D x_i\right)^2 \right ^{\frac{1}{2}} +$ $\frac{(0.5 \sum_{i=1}^D x_i^2 + \sum_{i=1}^D x_i)}{D} + 0.5$	1000	$[-100, 100]^d$	30(50)
F14	$f_{14}(x) = \left \sum_{i=1}^D x_i^2 - D \right ^{\frac{1}{4}} + \frac{(0.5 \sum_{i=1}^D x_i^2 + \sum_{i=1}^D x_i)}{D} + 0.5$	1000	$[-100, 100]^d$	30(50)
F15	$f_{15}(x) = 418.9829 \times D - \sum_{i=1}^D g(z_i)$	1000	$[-100, 100]^d$	30(50)

$$z_i = x_i + 4.209687462275036e + 002$$

$$\text{if } |z_i| \leq 500; g(z_i) = z_i \sin(|z_i|^{1/2})$$

$$\text{if } z_i > 500; g(z_i) = (500 - \text{mod}(z_i, 500)) * \sin(\sqrt{|500 - \text{mod}(z_i, 500)|}) - \frac{(z_i - 500)^2}{10000D}$$

$$\text{if } z_i < -500; g(z_i) = (\text{mod}(|z_i|, 500) - 500) * \sin(\sqrt{|\text{mod}(|z_i|, 500) - 500|}) - \frac{(z_i + 500)^2}{10000D}$$

Table 5: Data record of the results obtained from the IIMGHS algorithm and various HS algorithms

<i>FUNCTION</i>	<i>Algorithm</i>	<i>fbest_{min}</i>	<i>fbest_{mean}</i>	<i>fbest_{max}</i>	<i>fbest_{std}</i>	<i>Time</i>
F1(30dim)	HSCH	3.67E+07	7.18E+07	9.58E+07	1.72E+07	1.74E-02
	HSWB	3.00E+07	5.78E+07	9.45E+07	1.56E+07	1.37E-02
	EGHS	2.37E+07	5.01E+07	7.43E+07	1.33E+07	1.18E-02
	NGHS	8.61E+05	3.94E+06	9.04E+06	1.97E+06	1.01E-02
	IMGHSA	5.13E+05	3.13E+06	1.19E+07	2.57E+06	1.53E-02
	IIMGHSA	2.23E+05	9.60E+05	2.75E+06	5.50E+05	1.28E-02
F1(50dim)	HSCH	1.47E+08	2.05E+08	3.21E+08	4.39E+07	1.82E-02
	HSWB	1.15E+08	1.58E+08	2.33E+08	2.85E+07	1.94E-02
	EGHS	1.75E+08	2.73E+08	3.84E+08	4.58E+07	2.02E-02
	NGHS	2.23E+07	4.53E+07	8.58E+07	1.63E+07	1.71E-02
	IMGHSA	1.69E+07	3.22E+07	6.74E+07	1.03E+07	2.34E-02
	IIMGHSA	8.19E+06	1.68E+07	3.25E+07	5.51E+06	1.96E-02
F2(30Dim)	HSCH	4.63E+31	4.90E+41	5.66E+42	1.24E+42	1.69E-02
	HSWB	1.10E+31	2.10E+43	6.30E+44	1.15E+44	1.52E-02
	EGHS	1.30E+26	1.45E+29	1.09E+30	2.72E+29	1.61E-02
	NGHS	2.28E+12	6.48E+23	1.47E+25	2.79E+24	1.60E-02
	IMGHSA	1.29E+15	3.42E+22	9.13E+23	1.67E+23	1.79E-02
	IIMGHSA	2.07E+13	9.44E+22	8.96E+23	2.61E+23	1.76E-02
F2(50Dim)	HSCH	5.93E+61	5.94E+76	8.34E+77	1.80E+77	2.64E-02
	HSWB	3.98E+61	1.63E+73	3.75E+74	7.06E+73	2.73E-02
	EGHS	4.42E+54	1.14E+62	2.99E+63	5.45E+62	2.83E-02
	NGHS	1.47E+38	2.39E+53	7.10E+54	1.30E+54	2.57E-02
	IMGHSA	7.31E+42	5.18E+54	1.54E+56	2.81E+55	3.07E-02
	IIMGHSA	1.53E+40	4.71E+51	1.32E+53	2.40E+52	2.59E-02
F3(30Dim)	HSCH	9.79E+01	1.89E+02	3.12E+02	4.73E+01	1.12E-02
	HSWB	8.37E+01	1.33E+02	1.86E+02	2.51E+01	1.04E-02
	EGHS	4.10E+01	7.51E+01	1.08E+02	1.79E+01	1.24E-02
	NGHS	1.63E+00	8.40E+00	2.16E+01	5.84E+00	9.06E-03
	IMGHSA	1.17E+00	9.40E+00	5.03E+01	9.56E+00	1.21E-02
	IIMGHSA	5.33E-01	2.07E+00	5.28E+00	1.22E+00	1.19E-02
F3(50Dim)	HSCH	4.30E+02	5.79E+02	7.81E+02	9.45E+01	1.55E-02
	HSWB	1.96E+02	3.75E+02	5.10E+02	6.67E+01	1.49E-02
	EGHS	2.24E+02	3.05E+02	3.75E+02	4.21E+01	1.97E-02

A new global harmony search algorithm for solving numerical optimization

	NGHS	3.14E+01	7.00E+01	1.81E+02	3.17E+01	1.59E-02
	IMGHSA	2.74E+01	6.28E+01	1.16E+02	2.27E+01	2.02E-02
	IIMGHSA	1.36E+01	4.07E+01	8.07E+01	1.67E+01	1.78E-02
F4(30Dim)	HSCH	2.40E+04	9.37E+04	2.37E+05	5.34E+04	1.36E-02
	HSWB	2.03E+04	6.21E+04	1.12E+05	2.34E+04	1.18E-02
	EGHS	1.37E+04	4.69E+04	9.74E+04	2.16E+04	1.44E-02
	NGHS	2.83E+02	1.68E+03	5.19E+03	1.11E+03	1.22E-02
	IMGHSA	2.93E+02	1.99E+03	8.41E+03	2.10E+03	1.55E-02
F4(50Dim)	IIMGHSA	1.72E+02	7.53E+02	2.08E+03	4.60E+02	1.54E-02
	HSCH	2.57E+05	4.67E+05	8.78E+05	1.51E+05	1.66E-02
	HSWB	1.07E+05	2.54E+05	4.23E+05	7.60E+04	1.63E-02
	EGHS	2.70E+05	5.65E+05	9.18E+05	1.50E+05	1.92E-02
	NGHS	1.00E+04	3.74E+04	1.96E+05	3.54E+04	1.74E-02
	IMGHSA	9.74E+03	3.41E+04	1.23E+05	2.68E+04	2.10E-02
	IIMGHSA	3.18E+03	1.08E+04	2.98E+04	6.32E+03	1.89E-02

Table 5: IIMGHSA algorithm and data recording of the results obtained by various HS algorithms (continued)

F5(30Dim)	HSCH	5.42E+05	5.12E+06	1.71E+07	4.17E+06	2.46E-02
	HSWB	1.94E+05	1.93E+06	5.46E+06	1.53E+06	1.63E-02
	EGHS	2.61E+03	5.68E+05	2.11E+06	6.10E+05	1.92E-02
	NGHS	6.50E+01	4.12E+02	5.05E+03	8.96E+02	1.53E-02
	IMGHSA	3.87E+01	4.42E+02	6.48E+03	1.15E+03	1.89E-02
F5(50Dim)	IIMGHSA	7.20E+01	1.65E+02	3.61E+02	7.89E+01	2.23E-02
	HSCH	1.30E+07	4.30E+07	8.48E+07	2.08E+07	3.04E-02
	HSWB	4.34E+06	1.47E+07	3.56E+07	8.35E+06	2.41E-02
	EGHS	1.20E+07	3.95E+07	7.87E+07	1.95E+07	2.98E-02
	NGHS	6.83E+02	7.57E+05	2.49E+06	7.80E+05	2.53E-02
	IMGHSA	1.25E+03	3.62E+05	6.79E+06	1.23E+06	2.77E-02
	IIMGHSA	6.32E+02	1.75E+04	1.02E+05	2.43E+04	2.94E-02
F6(30Dim)	HSCH	3.66E+01	6.75E+01	1.30E+02	1.72E+01	1.65E-02
	HSWB	3.08E+01	5.50E+01	8.33E+01	1.35E+01	1.21E-02
	EGHS	3.39E+01	5.52E+01	7.82E+01	1.34E+01	1.54E-02
	NGHS	1.85E+00	6.40E+00	1.40E+01	3.09E+00	1.28E-02
	IMGHSA	2.27E+00	4.99E+00	1.34E+01	2.91E+00	1.64E-02
F6(50Dim)	IIMGHSA	1.37E+00	2.14E+00	3.55E+00	5.65E-01	1.77E-02
	HSCH	1.28E+02	1.97E+02	2.97E+02	3.99E+01	1.76E-02
	HSWB	9.15E+01	1.44E+02	2.23E+02	2.85E+01	1.69E-02
	EGHS	1.81E+02	2.61E+02	3.52E+02	3.93E+01	2.11E-02
	NGHS	1.77E+01	4.21E+01	1.04E+02	1.88E+01	1.91E-02
	IMGHSA	1.32E+01	3.52E+01	8.05E+01	1.69E+01	2.21E-02
	IIMGHSA	7.41E+00	1.74E+01	4.03E+01	6.26E+00	2.14E-02
F7(30Dim)	HSCH	1.43E+00	5.35E+00	1.03E+01	2.16E+00	1.78E-02
	HSWB	7.27E-01	2.23E+00	5.03E+00	9.82E-01	1.57E-02
	EGHS	1.35E+00	2.54E+00	5.14E+00	8.33E-01	1.68E-02
	NGHS	4.95E-01	1.34E+00	4.86E+00	8.31E-01	1.70E-02
	IMGHSA	5.95E-01	1.10E+00	2.21E+00	4.79E-01	1.93E-02

F7(50Dim)	IIMGHSA	2.50E-01	7.64E-01	1.41E+00	3.05E-01	2.03E-02
	HSCH	1.37E+01	2.56E+01	4.08E+01	6.39E+00	2.43E-02
	HSWB	9.76E+00	1.65E+01	2.33E+01	3.59E+00	2.32E-02
	EGHS	1.28E+01	2.93E+01	5.13E+01	1.04E+01	2.75E-02
	NGHS	1.68E+00	4.23E+00	1.09E+01	2.12E+00	2.41E-02
	IMGHSA	1.48E+00	3.76E+00	7.90E+00	1.54E+00	2.86E-02
	IIMGHSA	1.39E+00	2.50E+00	3.49E+00	6.13E-01	2.67E-02
F8(30Dim)	HSCH	-8.25E-01	-6.99E-01	-5.54E-01	6.26E-02	1.05E-02
	HSWB	-8.35E-01	-7.60E-01	-6.58E-01	4.22E-02	1.06E-02
	EGHS	-8.33E-01	-7.42E-01	-6.51E-01	5.48E-02	1.03E-02
	NGHS	-9.95E-01	-9.71E-01	-9.30E-01	1.66E-02	8.92E-03
	IMGHSA	-9.97E-01	-9.88E-01	-9.64E-01	9.29E-03	1.33E-02
	IIMGHSA	-9.97E-01	-9.95E-01	-9.88E-01	2.14E-03	1.29E-02
	F8(50Dim)	HSCH	-4.62E-01	-3.44E-01	-1.99E-01	6.27E-02
HSWB		-5.83E-01	-4.56E-01	-3.42E-01	6.64E-02	1.37E-02
EGHS		-3.68E-01	-2.41E-01	-1.42E-01	5.28E-02	1.72E-02
NGHS		-9.11E-01	-8.00E-01	-5.64E-01	7.76E-02	1.38E-02
IMGHSA		-9.27E-01	-8.32E-01	-6.58E-01	6.64E-02	1.88E-02
IIMGHSA		-9.64E-01	-9.09E-01	-8.17E-01	2.98E-02	1.55E-02

(Please see Appendix.1 for the experimental data of the benchmark function F9-F15)

In Table 5, you can visually see six kinds of harmony search algorithms running 30 cycles results in each standard function. The best value for each standard is marked in dark color. From the experimental data, it can be seen that when the dimension is 30, the search ability of IIMGHSA is better than other Harmony search algorithms in most cases. When the dimension is 50, it can be seen from the table. Under the benchmark function test, it is found that the searchability of the IIMGHSA algorithm is better than other algorithms, indicating that the IIMGHSA algorithm has better high-dimensional search capabilities. Moreover, it is not difficult to see from Table 5 that when the dimension is 30, the running time and results of the algorithm under certain benchmark functions are better than the IMGHSA algorithm. When the dimension is 50, only when the benchmark function is F5, the running time of the IIMGHSA algorithm is higher than that of the IMGHSA algorithm, which fully shows the convergence speed of the IIMGHSA algorithm is better than that of the IMGHSA algorithm. Although it is slightly inferior to other algorithms under the F10 and F11 benchmark functions, it is generally better than the IMGHSA algorithm. Comprehensive comparison, the IIMGHSA algorithm still shows better search capabilities than other algorithms.

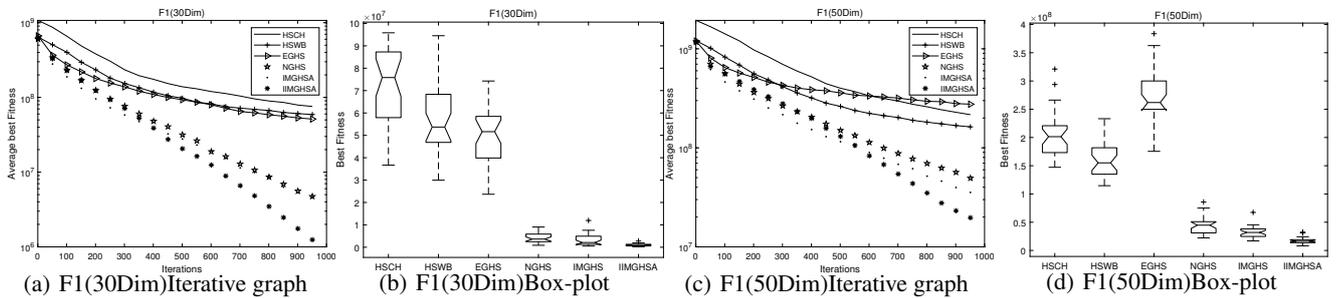


Fig. 1. Iterative graph and box-plot of the average optimal fitness value of F1 function after 30 runs under different algorithms

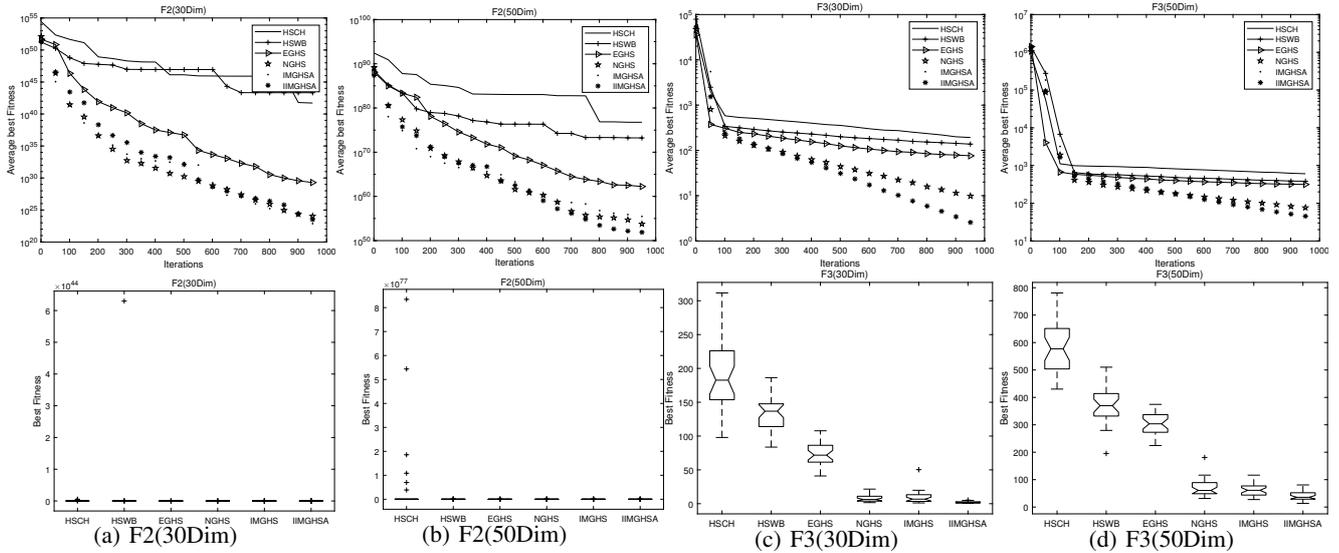


Fig. 2. Iterative graph and box-plot of the average optimal fitness value of F2-F3 function after 30 runs under different algorithms

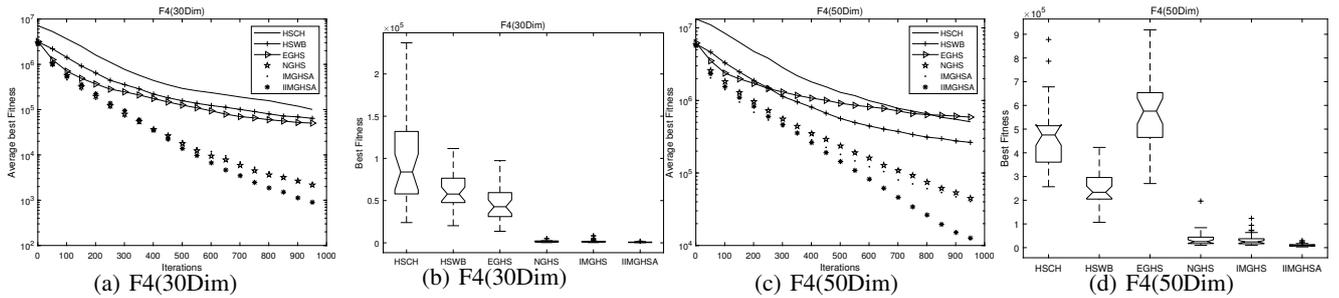
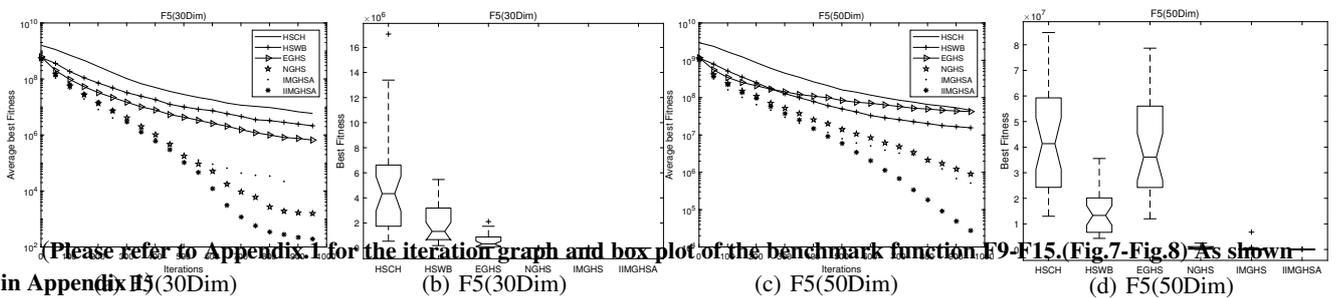


Fig. 3. Iterative graph and box-plot of the average optimal fitness value of F4 function after 30 runs under different algorithms



(Please refer to Appendix I for the iteration graph and box plot of the benchmark function F9-F15. (Fig.7-Fig.8) As shown in Appendix II)

Figures 1 to 8 in the paper show the iteration graphs under different benchmark functions. It can be seen from the content in the figure that although the convergence effect and convergence speed of the algorithm under the F10 and F11 benchmark functions are not the best, under the comprehensive test of 15 benchmark functions, the convergence ability

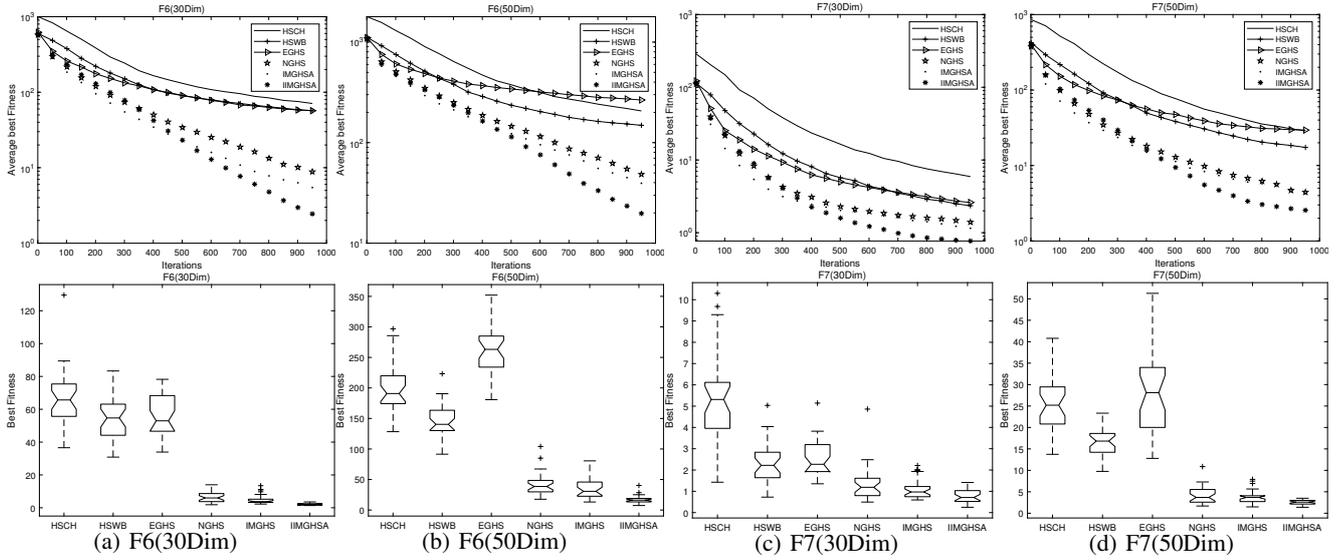


Fig. 5. Iterative graph and box-plot of the average optimal fitness value of F6-F7 function after 30 runs under different algorithms

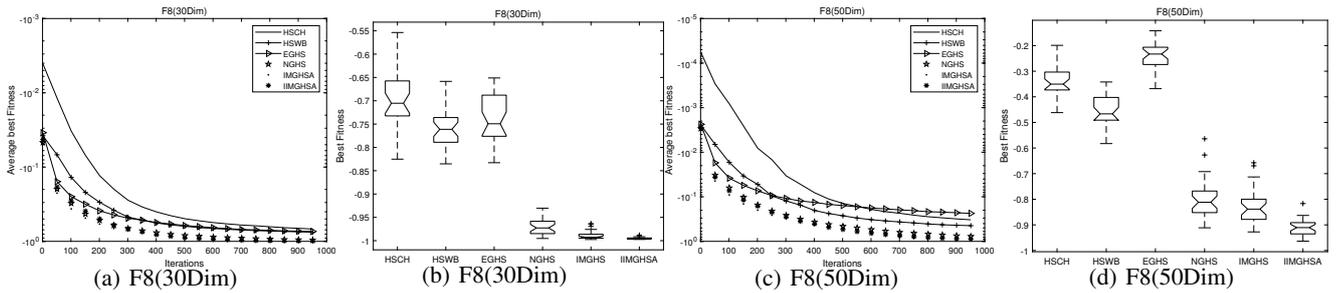


Fig. 6. Iterative graph and box-plot of the average optimal fitness value of F8 function after 30 runs under different algorithms

and convergence speed of the algorithm are better than other algorithms. Although it can be seen from the figure that under the test of several benchmark functions, although the search efficiency of this algorithm is better than that of the IMGHS algorithm, the performance of this algorithm is still not as good as other HS algorithms. It can also be seen from the figure that the IIMGHS algorithm performs better than other algorithms under most benchmark functional tests. Therefore, it can be seen that the IIMGHS algorithm proposed in this paper performs better under the comprehensive test of 15 benchmark functions. Moreover, these results' global convergence ability and search effect can also be seen directly from the box plots of different algorithms in Figure 1-8. The results of other algorithms vary greatly among multiple searches. However, the maximum quality factor searched by the IIMGHS algorithm still has a stable optimization ability under a random initial value, indicating that the IIMGHS algorithm has a good and stable search ability. Among the abnormal points, the optimal fitness of the IIMGHS algorithm is also better than that of its algorithm, which means that the stability and convergence of the IIMGHS algorithm are better than other algorithms.

19. Alharkan Ibrahim M.,Qamhan Ammar A.,Badwelan Ahmed, Alsamhan Ali,Hidri Lotfi. Modified Harmony Search Algorithm for Resource-Constrained Parallel Machine Scheduling Problem with Release Dates and Sequence-Dependent Setup Times[J]. Processes,2021,9(4).
20. Zou Rui. Research and Application of Distribution Network Based on Improved Harmony Search Algorithm[D]. Xi'an University of Architecture and Technology,2020.
21. Srikanth. R, Bikshalu K . Multilevel thresholding image segmentation based on energy curve with harmony Search Algorithm[J]. AIN SHAMS ENGINEERING JOURNAL.2021.12(1):1-20
22. Xinchao Zhao,Rui Li,Junling Hao,Zhaohua Liu,Jianmei Yuan. A New Differential Mutation Based Adaptive Harmony Search Algorithm for Global Optimization[J]. Applied Sciences,2020,10(8).
23. J.L. Ponz-Tienda, A. Salcedo-Bernal, E. Pellicer,J. Benlloch-Marco. Improved Adaptive Harmony Search algorithm for the Resource Leveling Problem with minimal lags[J]. Automation in Construction,2017,77.
24. Mahdavi,Fesanghary,Damangir. An improved harmony search algorithm for solving optimization problems[J], APPLIED MATHEMATICS AND COMPUTATION ,2007 ,188(2):1567-1579
25. Omran, Mahamed G. H. Mahdavi,Mehrdad.Global-best harmony search[J] APPLIED MATHEMATICS AND COMPUTATION,2008,198(2):643-656
26. Gholami J , Ghany K , Zawbaa H M . A Novel Global Harmony Search Algorithm for Solving Numerical Optimizations[J]. Soft Computing, 2021.
27. YONG L Q,LIU S Y,TUO S H,et al.Improved harmony search algorithm with chaos for absolute value equation[J].Telkomnika,2013,11(4): 835-844.
28. Yong Longquan.An improved harmony search algorithm for solving nonlinear equations[J].Journal of Chongqing University of Technology (Natural Science),2020,34(10):231-237.
29. Yong Longquan, Liu Sanyang, Tuo Shouheng, Xiong Wentao, Chen Tao. Improved harmony search algorithm for absolute value equation[J].Journal of Natural Science of Heilongjiang University,2013,30(03):321-327.
30. An effective global harmony search algorithm for reliability problems[J]. Expert Systems with Applications An International Journal, 2011, 38(4):4642-4648.
31. Ouyang Haibin. Improvements and Applications of Harmony Search Algorithm[D]. Northeastern University,2015.
32. N. H. Awad, M. Z. Ali, P. N. Suganthan, J. J. Liang and B. Y. Qu. Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. 2016.
33. Wang L, Hu H, Liu R, et al. An improved differential harmony search algorithm for function optimization problems[J]. Soft Computing, 2018.

(Appendix.1 on the next page)

Appendix.1

A Benchmark function F9-F15 experimental data

Table 6: Data record of the results obtained from the IIMGHS algorithm and various HS algorithms(**Table 5 continued**)

<i>FUNCTION</i>	<i>Algorithm</i>	$f_{best_{min}}$	$f_{best_{mean}}$	$f_{best_{max}}$	$f_{best_{std}}$	<i>Time</i>
F9(30Dim)	HSCH	3.50E+03	7.03E+03	1.34E+04	1.92E+03	1.00E-02
	HSWB	2.87E+03	5.56E+03	9.00E+03	1.35E+03	1.13E-02
	EGHS	3.21E+03	5.58E+03	8.13E+03	1.49E+03	9.76E-03
	NGHS	-3.54E+02	1.51E+02	9.95E+02	3.44E+02	9.19E-03
	IMGHSA	-3.73E+02	-3.36E+01	1.79E+03	4.00E+02	1.19E-02
	IIMGHSA	-4.12E+02	-3.36E+02	-1.41E+02	6.57E+01	1.27E-02
F9(50Dim)	HSCH	1.49E+04	2.11E+04	3.03E+04	3.73E+03	1.35E-02
	HSWB	9.71E+03	1.51E+04	2.39E+04	3.38E+03	1.24E-02
	EGHS	1.95E+04	2.85E+04	3.86E+04	4.37E+03	1.72E-02
	NGHS	1.40E+03	4.12E+03	1.10E+04	2.09E+03	1.40E-02
	IMGHSA	6.40E+02	3.64E+03	9.20E+03	2.12E+03	1.73E-02
	IIMGHSA	5.99E+02	1.50E+03	3.20E+03	6.71E+02	1.58E-02
F10(30Dim)	HSCH	9.46E+01	1.21E+02	1.69E+02	1.82E+01	9.96E-03
	HSWB	7.66E+01	1.11E+02	1.51E+02	1.92E+01	1.22E-02
	EGHS	1.47E+02	1.84E+02	2.09E+02	1.58E+01	1.11E-02
	NGHS	6.44E+01	1.10E+02	1.62E+02	2.52E+01	1.05E-02
	IMGHSA	6.88E+01	1.06E+02	1.79E+02	2.84E+01	1.53E-02
	IIMGHSA	6.67E+01	1.12E+02	1.65E+02	2.62E+01	1.33E-02
F10(50Dim)	HSCH	2.31E+02	2.71E+02	3.27E+02	2.38E+01	1.61E-02
	HSWB	2.14E+02	2.49E+02	3.02E+02	2.48E+01	1.55E-02
	EGHS	3.75E+02	4.38E+02	5.14E+02	3.30E+01	1.90E-02
	NGHS	1.94E+02	2.68E+02	3.13E+02	3.03E+01	1.62E-02
	IMGHSA	1.93E+02	2.66E+02	3.35E+02	3.47E+01	2.00E-02
	IIMGHSA	1.90E+02	2.85E+02	3.65E+02	4.22E+01	1.77E-02
F11(30Dim)	HSCH	-7.18E+01	-6.84E+01	-6.54E+01	1.94E+00	1.76E-02
	HSWB	-7.24E+01	-6.87E+01	-6.55E+01	1.87E+00	1.46E-02
	EGHS	-7.08E+01	-6.56E+01	-6.00E+01	2.26E+00	1.67E-02
	NGHS	-7.22E+01	-6.87E+01	-6.30E+01	2.13E+00	1.62E-02
	IMGHSA	-7.41E+01	-6.99E+01	-6.33E+01	2.25E+00	1.81E-02
	IIMGHSA	-7.44E+01	-6.91E+01	-6.25E+01	2.82E+00	1.85E-02
F11(50Dim)	HSCH	-6.66E+01	-6.30E+01	-5.92E+01	2.06E+00	2.51E-02
	HSWB	-6.80E+01	-6.42E+01	-6.06E+01	1.94E+00	2.40E-02
	EGHS	-5.99E+01	-5.59E+01	-5.08E+01	2.37E+00	2.57E-02
	NGHS	-6.77E+01	-6.39E+01	-6.04E+01	1.95E+00	2.27E-02
	IMGHSA	-6.84E+01	-6.48E+01	-6.13E+01	1.70E+00	2.90E-02
	IIMGHSA	-6.88E+01	-6.50E+01	-6.14E+01	1.94E+00	2.69E-02
F12(30Dim)	HSCH	9.28E+04	2.24E+05	5.69E+05	1.06E+05	1.38E-02
	HSWB	2.61E+04	7.29E+04	1.40E+05	2.90E+04	1.58E-02
	EGHS	1.17E+04	4.68E+04	1.20E+05	2.45E+04	1.40E-02
	NGHS	1.93E+03	8.40E+03	2.26E+04	5.87E+03	1.30E-02

F12(50Dim)	IMGHSA	9.12E+02	6.93E+03	1.93E+04	4.90E+03	1.73E-02
	IIMGHSA	1.71E+02	1.89E+03	8.72E+03	1.77E+03	1.67E-02
	HSCH	5.85E+05	1.02E+06	1.97E+06	3.15E+05	1.76E-02
	HSWB	1.99E+05	3.78E+05	5.64E+05	1.03E+05	1.78E-02
	EGHS	2.42E+05	5.91E+05	1.06E+06	2.18E+05	1.83E-02
	NGHS	2.91E+04	9.10E+04	2.08E+05	5.12E+04	1.40E-02
	IMGHSA	3.79E+04	9.13E+04	1.75E+05	4.08E+04	1.89E-02
	IIMGHSA	5.96E+03	4.36E+04	1.23E+05	3.13E+04	1.82E-02
F13(30Dim)	HSCH	4.56E+03	7.84E+03	1.32E+04	2.34E+03	1.01E-02
	HSWB	3.28E+03	6.05E+03	8.72E+03	1.33E+03	1.29E-02
	EGHS	4.14E+03	6.02E+03	8.47E+03	1.31E+03	1.10E-02
	NGHS	1.54E+02	5.11E+02	1.18E+03	2.84E+02	9.70E-03
	IMGHSA	7.04E+01	3.98E+02	1.43E+03	3.46E+02	1.48E-02
F13(50Dim)	IIMGHSA	4.74E+01	1.18E+02	2.64E+02	5.03E+01	1.25E-02
	HSCH	1.62E+04	2.16E+04	3.01E+04	3.11E+03	1.36E-02
	HSWB	1.08E+04	1.56E+04	2.17E+04	2.66E+03	1.35E-02
	EGHS	2.02E+04	3.00E+04	3.97E+04	4.73E+03	1.54E-02
	NGHS	1.86E+03	4.81E+03	8.83E+03	1.85E+03	1.34E-02
	IMGHSA	1.33E+03	3.95E+03	7.48E+03	1.63E+03	1.76E-02
	IIMGHSA	9.06E+02	2.19E+03	4.19E+03	8.64E+02	1.56E-02
	F14(30Dim)	HSCH	7.80E+01	1.31E+02	2.41E+02	3.00E+01
HSWB		6.33E+01	1.05E+02	1.66E+02	2.75E+01	1.04E-02
EGHS		7.29E+01	1.15E+02	1.73E+02	2.46E+01	1.13E-02
NGHS		4.56E+00	1.29E+01	3.10E+01	6.23E+00	1.02E-02
IMGHSA		2.79E+00	1.04E+01	2.85E+01	6.00E+00	1.54E-02
IIMGHSA		3.42E+00	5.13E+00	1.04E+01	1.33E+00	1.21E-02
F14(50Dim)	HSCH	1.52E+02	2.25E+02	2.92E+02	4.07E+01	1.38E-02
	HSWB	1.05E+02	1.73E+02	2.33E+02	2.99E+01	1.31E-02
	EGHS	2.49E+02	3.29E+02	4.37E+02	4.41E+01	1.82E-02
	NGHS	3.22E+01	5.43E+01	1.07E+02	1.60E+01	1.33E-02
	IMGHSA	2.45E+01	4.72E+01	8.02E+01	1.57E+01	1.78E-02
	IIMGHSA	1.20E+01	2.72E+01	4.91E+01	9.20E+00	1.66E-02
F15(30Dim)	HSCH	6.96E+03	6.99E+03	7.02E+03	1.62E+01	1.67E-02
	HSWB	6.97E+03	7.04E+03	7.07E+03	2.50E+01	1.38E-02
	EGHS	7.00E+03	7.07E+03	7.12E+03	2.71E+01	1.50E-02
	NGHS	6.89E+03	6.91E+03	6.99E+03	2.55E+01	1.57E-02
	IMGHSA	6.89E+03	6.92E+03	6.99E+03	2.83E+01	1.76E-02
	IIMGHSA	6.88E+03	6.90E+03	6.95E+03	1.43E+01	1.86E-02
F15(50Dim)	HSCH	1.17E+04	1.17E+04	1.18E+04	2.83E+01	2.52E-02
	HSWB	1.17E+04	1.18E+04	1.20E+04	4.98E+01	1.93E-02
	EGHS	1.19E+04	1.21E+04	1.22E+04	6.36E+01	2.37E-02
	NGHS	1.16E+04	1.16E+04	1.17E+04	4.14E+01	2.11E-02
	IMGHSA	1.16E+04	1.17E+04	1.18E+04	5.61E+01	2.56E-02
	IIMGHSA	1.16E+04	1.16E+04	1.18E+04	4.87E+01	2.23E-02

B Iterative graph and box-plot of F9-F15 benchmark function under different algorithms

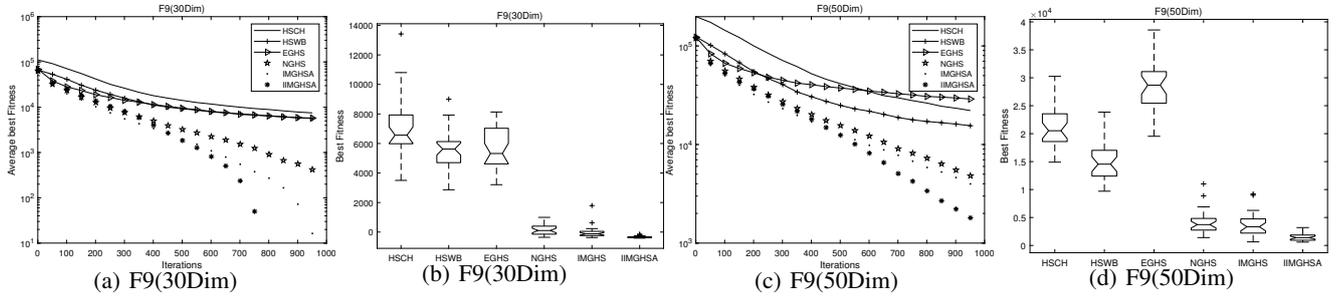


Fig. 7. Iterative graph and box-plot of the average optimal fitness value of F9 function after 30 runs under different algorithms

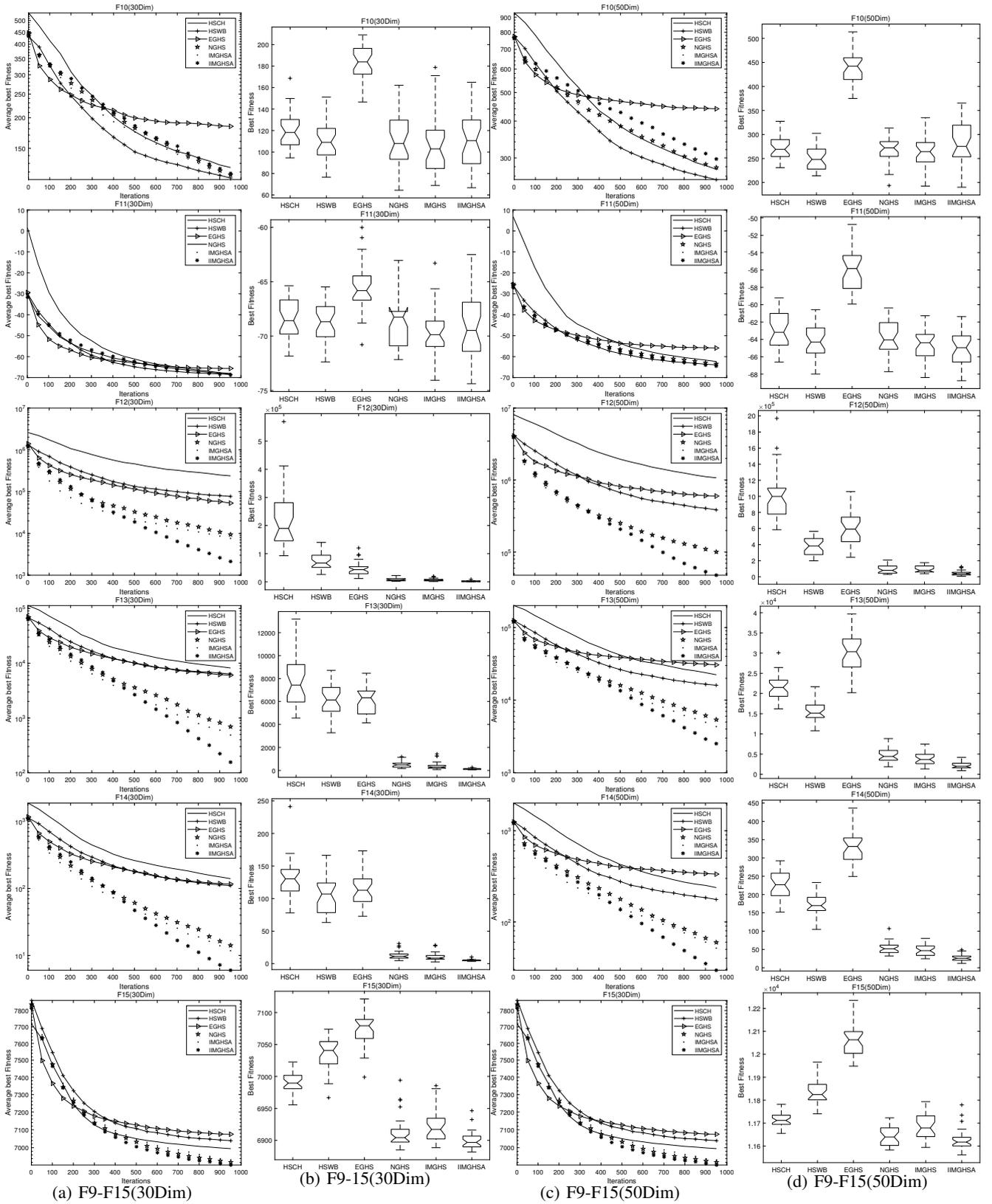


Fig. 8. Iterative graph and box-plot of the average optimal fitness value of F10-F15 function after 30 runs under different algorithms

Research on Deep Learning Model for Solving the Evaluation of Wine Quality

Jin Zhou¹[0000-0003-0987-3943], Kang Zhou¹[0000-0002-0205-768X] *, Gexiang Zhang²[0000-0001-8034-0977], and Jiuju Yin¹

¹ School of Mathematics and Computer Science, Wuhan Polytechnic University, Wuhan 430023, China
zhoukang65@whpu.edu.cn

² Research Center for Artificial Intelligence, Chengdu University of Technology, Chengdu 610059, China
zhgxdylan@126.com

Abstract. The traditional wine quality assessment only uses sensory test which is susceptible to individual subjective factors, so this paper takes the physicochemical properties of wine as the basis for quality assessment. When neural network solves this kind of problem, in order to overcome the problem that model training is easy to fall into local optimum and the accuracy is not high, intelligent optimization algorithm is introduced to improve these problems, which also greatly increases the computational workload. In order to improve the prediction accuracy of the model and reduce the computational complexity of simultaneous optimization of the structure and parameters of the neural network, this paper designs a deep collaborative optimization framework. The preparatory work for using this framework to optimize is to determine the type of neural network suitable for solving such problems as the candidate set according to the characteristics of the problems and the data. The framework consists of three core parts: feature extraction, neural network structure optimization and neural network parameter optimization. Feature extraction is to analyze the relationship between features and eliminate the data features with low importance ranking and high similarity. Neural network optimization is to use modified genetic algorithm and firefly algorithm to optimize the structure and parameters of neural network respectively. Finally, a neural network model with the best type, structure and parameters will be selected, and this model will be applied to the quality evaluation of wine. The optimal neural network model obtained through simulation experiments, for red wine data, the average accuracy of tolerance of 0.5 and 1.0 increased by 3% and 4.7%, respectively; for white wine data, a tolerance of 0.5 has a similar effect, and a tolerance of 1.0 increases the average accuracy value by 1.7%. This also proves the effectiveness and feasibility of the deep collaborative optimization framework.

Keywords: Neural network optimization · Intelligent optimization algorithm based on NNs · Feature selection.

* Corresponding author

1 Introduction

With the continuous improvement of people's living standards, wine, as a kind of fruit wine, is more and more favored by the majority of consumers. The International Organization of Vine and Wine (OIV) pointed out in the 2020 online video conference that the global wine production in 2020 is estimated to be between 253.9 and 262.2 billion liters, which is a very large number. With the increasing pursuit of consumers for wine, manufacturers have been committed to improving the quality of wine, so that their wines can be favored by the majority of consumers. Taste, appearance color and aroma are three important indexes to evaluate the quality of wine. Everyone has different preferences for wine, so how to objectively evaluate the quality of wine is very important.

The traditional evaluation of wine quality is mainly from two aspects: sensory testing and physicochemical properties analysis. Sensory testing uses professional wine appraisers to determine the quality of wine. However, in recent years, there have been wine informatics studies based on analyzing wine reviews and calculating wine tasting rounds [1-4]. This is done by analyzing a large number of consumers. For quality evaluation by comments, both professional appraisers and consumers will be affected by subjective factors such as personal favorites and experience, so their evaluations are more or less subjectively biased [5][6]. The physicochemical properties are the use of data mining, artificial intelligence and other technologies to explore the internal relationship between the physical and chemical properties and quality of wine, so as to evaluate the quality of wine [5] [7-9], and at the same time, the selection of wine raw materials and their Proportion for tuning [10]. Therefore, for a wine quality system formed by some main physicochemical indexes, how to find the best data mining method to establish a wine quality scoring system and make the wine quality scoring system more accurate is a very important problem.

With the continuous development of information technology, people gradually began to adopt neural network technology for this type of forecasting problem. In the field of deep learning, neural network technology has been applied in a wide range of fields, such as pattern recognition, function optimization, image processing, classification and regression problems [11][12][13]. Neural network is a way to simulate the biological nervous system to process information, which has significant advantages in nonlinear modeling and has good generalization ability [14]. The performance of neural network model depends on its network structure and network parameters. The traditional neural network structure consists of four parts: the number of hidden layers, the number of nodes in the hidden layers, the activation function and the optimizer [15]. Network parameters include the weight of the connection between nodes and the offset value, etc. The traditional training methods of neural network can be divided into two types: gradient descent algorithm and stochastic algorithm. The training algorithm based on gradient descent has the problem of easily falling into the local optimum and low accuracy [12][16][17]. Therefore, in order to overcome these problems, researchers continue to introduce meta-heuristic algorithms into neural networks [18].

Compared with the algorithm based on gradient descent, the meta-heuristic algorithm has a better effect in global optimization [12][19]. In recent years, the combination of meta-heuristic algorithm and neural network has been widely used in medical detection [20], climate prediction [21], digital image detection [22], software reliability prediction [23] and other fields [24][25], all of which have achieved good results. The optimization of meta-heuristic algorithms in neural networks mainly focuses on two directions: the structure of the neural network and the parameters in the neural network. When the meta-heuristic algorithm is used to optimize the neural network structure, the traditional neural network training algorithm will make the parameters fall into the local optimum [15] [26]. When only optimizing the network parameters, the network structure needs to be determined in advance, which is often set based on experience, and it is impossible to guarantee whether the network structure is optimal [12][27]. Maochuan Wu[23] and Yadav A[28] used the meta-heuristic algorithm to optimize the structure and parameters of the neural network at the same time and achieved a good result. But, this will produce a large number of coding parameters and is a large-scale optimization problem [12][14], which will consume a lot of time.

In order to overcome the problems that traditional neural network training algorithm is easy to fall into local optimum, low precision and high complexity of simultaneous optimization of neural network structure and parameters, a deep collaborative optimization framework is designed in this paper to optimize and screen the types, structures and parameters of neural networks. Finally, the framework is used to obtain an optimal neural network model, which can be used as a wine quality scoring system.

The main contributions of this paper are as follows:

(1) A deep collaborative optimization framework is designed. The collaborative optimization framework consists of three stages: feature extraction, neural network structure optimization, and neural network parameter optimization. First, according to the characteristics of the problem to be solved and the data, the type of neural network suitable for solving this type of problem is determined through investigation and simple experiment as the type candidate set. Secondly, different analysis methods are used to extract the features of the data set from different perspectives, so as to provide more comprehensive feature data for the subsequent search of the optimal neural network model. Then, some neural network models with ideal structures are calculated for different neural network types as candidate sets. Finally, the neural network parameters are globally optimized in the candidate set, so that the candidate neural network model achieves the best performance, and thus the optimal neural network model is obtained.

(2) Meta-heuristic algorithm reconstruction based on NNS. In view of the optimized structure of the neural network, the chromosome is divided into different segments, and each segment corresponds to a different function. For the intermediate layer number of neural network, genetic algorithm carries out variable length coding, in which for the optimal node number of each layer, this paper designs a coding method and search method based on the interval. In

addition, the genetic operation of group optimization is introduced for neural networks of different depths. The disturbance in the traditional position update formula of the firefly algorithm is completely random. For this, a position update formula with firefly position information disturbance is proposed, and an adaptive disturbance factor is introduced to adapt to the changes of the population environment.

The organization structure of this paper is as follows: Section 2 shows the basis of the data set and the method and technology used; Section 3 designs a deep collaborative optimization framework and the description of the optimization algorithm; Section 4 is the final experimental results and analysis; Section 5 is the summary and prospect of the model research.

2 Materials and Methods

2.1 Data set and scoring system

The wine quality data set Wine Quality [5] used in this experiment comes from the UCI machine learning library. The data set contains two types of red wine and white wine. These two data sets are about the red and white varieties of Portuguese "Vinho Verde" wine. Among them, there are 1599 sample data for red wine and 4898 sample data for white wine. In this data set, each sample contains 11 physicochemical indicators of wine and a quality score based on sensory data (the median of at least 3 evaluations by wine experts). The sensory data is that the taster rated it according to a ten-point scale, where 0 means very poor and 10 means best. The 11 physicochemical indexes of wine are shown in Table1, and the distribution of quality scores is shown in Table2. It can be seen from Table1 that the range of characteristic data is quite different and inconsistent, so it needs to be standardized before the data is put into use.

Regarding the quality system formed by these 11 physicochemical indicators of wine, this paper looks for an optimal data mining method to establish a wine quality scoring system, so that the wine quality scoring system can score more accurately. Grading wine quality is a typical regression problem, and neural network technology is one of the effective methods to solve this problem.

In the feature extraction stage, the methods used in this paper are random forest and pearson coefficient. The meta-heuristic algorithm used for neural network optimization includes genetic algorithm and firefly algorithm.

2.2 Feature extraction method

Random forest [29] is an extremely important branch ensemble learning in machine learning. It is a classifier that contains multiple decision trees. Through the self-service resampling technology, from the original data set (size N), there are replacement samples N times to form a training set, thereby constructing a decision tree, and repeating this many times to form a decision tree forest. Finally, the classification of each tree in the forest is integrated to determine the

Table 1. Physicochemical properties of wine

Attribute	Red			White		
	Mean	Interval	STD	Mean	Interval	STD
<i>Fixeda acidity(g(tartaric acid)/dm3)</i>	8.32	[4.60, 15.90]	1.74	6.86	[3.80, 14.20]	0.84
<i>Volatile acidity(g(acetic acid)/dm3)</i>	0.53	[0.12,1.58]	0.18	0.28	[0.08,1.10]	0.10
<i>Citric acid(g/dm3)</i>	0.27	[0.00,1.00]	0.20	0.33	[0.00,1.66]	0.12
<i>Residual sugar(g/dm3)</i>	2.54	[0.90,15.50]	1.41	6.39	[0.60,65.80]	5.07
<i>Chlorides(g(sodium chloride)/dm3)</i>	0.09	[0.12,0.61]	0.05	0.05	[0.01,0.35]	0.02
<i>Free sulfur dioxide(mg/dm3)</i>	15.87	[1.00,72.00]	10.46	35.31	[2.00,289.0]	17.01
<i>Total sulfur dioxide(mg/dm3)</i>	46.47	[6.00,289.0]	32.89	138.4	[9.00,440.0]	42.50
<i>Density(g/cm3)</i>	0.10	[0.99,1.00]	0.002	0.99	[0.987,1.039]	0.003
<i>pH</i>	3.31	[2.74,4.01]	0.15	3.19	[2.72,3.82]	0.15
<i>Sulphates(g(potassium sulphate)/dm3)</i>	0.66	[0.33,2.00]	0.17	0.49	[0.22,1.08]	0.11
<i>Alcohol(%vol)</i>	10.42	[8.40,14.90]	1.07	10.51	[8.00,14.20]	1.23

Table 2. Wine score distribution

Rate	3	4	5	6	7	8	9
<i>Red(1599)</i>	10	53	681	638	199	18	0
<i>White(4898)</i>	20	163	1457	2198	880	175	5

category of a sample. Random forest can also be used for regression, with good generalization and error balance ability. Random forest can give the importance of each feature when dealing with classification or regression problems, which provides an important reference for feature selection. The calculation formula of its feature importance is as follows:

$$importance = \frac{1}{N} * \sum_{i=1}^N |errOOB_2^i - errOOB_1^i| \quad (1)$$

Where N is the number of decision trees in the random forest, $errOOB_1^i$ is the out-of-bag data error of decision tree i , and $errOOB_2^i$ is the out-of-bag data error after adding interference to the OOB feature of the out-of-bag data.

The Pearson correlation coefficient[30] is a measure of the correlation between two characteristic data. The Pearson correlation coefficient varies from -1 to +1, where r represents the Pearson correlation coefficient. When $r > 0$, it means that the two variables are positively correlated. When $r < 0$, it means that the two variables are negatively correlated. When $r = 0$, it indicates that the two variables are linearly uncorrelated. The greater the absolute value of r , the stronger the correlation between the two variables. The Pearson correlation coefficient is defined as the quotient of the covariance and standard deviation between two variables. The calculation formula is as follows:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (2)$$

When analyzing the data, we must comprehensively consider the importance and relevance of the features. Extract features from these two angles, and delete some unimportant feature data. At the same time, the data volume is reduced and the noise data is removed without changing the meaning of the data set and without distortion, so that the calculation accuracy and efficiency can be improved.

2.3 Metaheuristic algorithm

Genetic algorithm is a classic evolutionary algorithm that simulates natural selection and biological evolution process [31]. This algorithm has been widely used in many optimization problems since its birth [32][33]. The core of genetic algorithm lies in three genetic operations: selection, crossover and mutation.

Selection operator is the beginning of genetic evolution. Excellent individuals are selected as parents to reproduce in the group. Common choices include tournament selection and roulette selection. The crossover operator acts on the parent and is the main way to generate new individuals. It plays a central role in the genetic operator. Common crossover methods include single-point, multi-point crossover, recombination, and rearrangement. The mutation operator is

the process of simulating gene mutation, perturbing the gene on the chromosome with a small probability. The parent population will form a new offspring population under the action of three genetic operators.

The firefly algorithm is a relatively novel swarm intelligence optimization algorithm [34][35]. Its core idea is that fireflies are always attracted to and approached by their brighter companions. Compared with other population evolution algorithms, the firefly algorithm can traverse every part, avoid falling into the local optimum, and has a good global search ability.

Firefly fluorescence brightness:

$$I = I_0 e^{-\gamma r^2} \tag{3}$$

Where, I_0 is the absolute brightness of fireflies, and I is the fluorescence brightness of fireflies at the distance r .

Cartesian distance between firefly x_i and firefly x_j :

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^d (x_{ik} - x_{jk})^2} \tag{4}$$

The firefly x_i is attracted by the firefly x_j and moves towards it. The formula for movement of the firefly x_i is

$$x_i^{t+1} = x_i^t + \beta_0 e^{(-\gamma r_{ij}^2)} (x_j^t - x_i^t) + \alpha \theta_i^t \tag{5}$$

Where, γ is the light absorption coefficient, and β_0 is the maximum attraction of the firefly, that is, the attraction at $r_{ij} = 0$. α is a random step disturbance factor between 0 and 1, and θ_i^t is a random number vector with Gaussian distribution or uniform distribution.

2.4 Neural Networks

Since the development of neural networks, many types have emerged. Each type of neural network has its own advantages and disadvantages. Therefore, through investigation and simple experiments, this paper selects three different types of neural networks for the final optimization. The three types of neural networks are back propagation neural network (BP) [36], recurrent neural network (RNN) [37] and convolutional neural network (CNN) [38].

BP neural network is a traditional forward network, which does not consider the correlation between data and has better nonlinear mapping capabilities. The output of the network is only related to the input of the current network. It is mainly composed of input layer, hidden layer and output layer, and each layer is composed of multiple neuron nodes, and each neuron between adjacent layers is fully connection. RNN is a kind of feedback network. Compared with the forward network, the network information must be transmitted backward through the memory unit while forwarding, thus further enhancing the accuracy and fault tolerance of network learning. Although convolutional neural networks

are widely used in computer vision and natural language processing, because of their outstanding feature extraction capabilities, this paper refers to one-dimensional convolutional neural networks for data processing, which is mainly composed of convolutional layers, pooling layers and the fully connected layer.

3 Deep learning framework

3.1 Deep collaborative optimization framework

The grading problem of wine quality is a regression problem, and neural network technology is one of the effective means to solve this problem. How to find the best neural network model to achieve the best regression effect is the key problem in this field.

This paper believes that the best neural network model is mainly reflected in the following three aspects: the most suitable neural network type; the optimal neural network structure; the optimal neural network parameters. Therefore, this paper proposes a framework for finding the optimal neural network model as shown in Figure 1, which is called a deep collaborative optimization framework.

The calculation process of the deep collaborative optimization framework is divided into three steps: feature extraction, neural network structure optimization, and neural network parameter optimization. The preliminary preparation for finding the best neural network model by using this optimization framework is as follows: according to the characteristics of the problems to be solved and the data, the neural network type suitable for solving this kind of problems is determined as candidate set through investigation and simple experiments. In the feature extraction stage, different analysis methods can be used in parallel to extract the features of data sets from different perspectives, so as to provide more comprehensive feature data for the subsequent search of the optimal neural network model. In the stage of neural network structure optimization, the purpose is to calculate some neural network models with ideal structures for different types of neural networks, which can be used as the candidate set of subsequent optimal neural network models. Because this part needs to search for several ideal neural network structures for each neural network type, the search range is wide and the calculation is large, so in order to improve the calculation efficiency, the traditional optimizer should be used to optimize the neural network parameters and the appropriate intelligent optimization algorithm should be selected to carry out the global search. In the stage of calculating the optimal neural network parameters, the purpose is to calculate the neural network model with the optimal neural network parameters from the candidate set of neural network models in the second stage. In this stage, the neural network parameters should be globally optimized so that the candidate neural network model can achieve the best performance.

As shown in algorithm 1, the specific working process of the deep collaborative optimization framework is as follows:

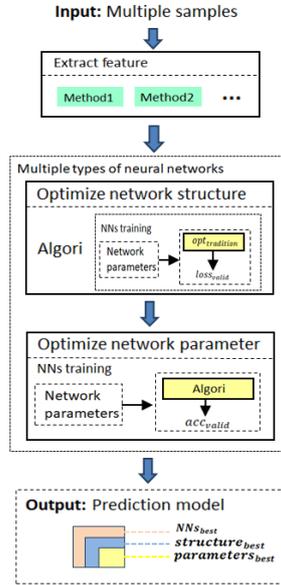


Fig. 1. Deep learning optimization framework

The first stage: the original input data I_{ij} is extracted by different methods, and the more concentrated data feature I_{ij}^* is finally used as the input of the neural network.

The second stage: the intelligent optimization algorithm is used to optimize the number of intermediate nodes (i.e., number of neurons or convolution kernel, etc.), activation function and optimizer of different deep neural networks. The traditional optimizer $opt_{tradition}$ is used to train the network parameters of the neural network. After parameter training, the loss function $loss_{valid}$ of the verification set is used as the objective function of the optimization algorithm. At the end of the second stage, a neural network model candidate set S^{best} with ideal neural network structure is obtained in each type of neural network.

The third stage: the intelligent optimization algorithm is used to conduct global optimization for all adjustable parameters of the network in the candidate set of neural network model S^{best} , and adjust them to the optimal P^{best} . In addition, since the objective function $loss_{valid}$ has achieved a relatively ideal state in the second stage, the accuracy value of the verification set acc_{valid} is taken as the objective function of the algorithm in this stage to calculate the neural network model with optimal neural network parameters from the candidate set of neural network models. Thus, the optimal neural network model $M_{N_{best}S_{best}P_{best}}$ is obtained, where N_{best} , S_{best} and P_{best} respectively represent the optimal network type, network structure and network parameters in the candidate set.

Faced with the wine quality prediction problem to be solved in this paper, the structure parameters of the neural network are shown in Table 3. After in-

Algorithm 1 Collaborative optimization framework

Variable declaration: n : sample size, m : feature sizes, d : maximum network depth

M : network model, N : network type, S : structure type, P : network parameters

$loss_{valid}$: Validation set loss function, acc_{valid} : Validation set accuracy

Input: data $I_{ij}(i = 1, 2, \dots, n, j = 1, 2, \dots, m)$

1: Feature extraction:

2: Extract the feature of the I_{ij} from different angles

3: Comprehensive feature extraction results

4: Remove features to make them more focused

5: Obtain final input data I_{ij}^* ($i = 1, 2, \dots, n, j = 1, 2, \dots, m^*, m^* < m$)

6: Network structure optimization:

7: Define the structure of different network types

8: Encode the network structure $S = \{s_1, s_2, \dots, s_d\}$

9: Train neural network parameters through $opt_{tradition}$

10: Calculate the objective function $loss_{valid}$

11: Perform grouping evolution according to network depth

12: Generate the best network structure $S^{best} = \{s_1^{best}, s_2^{best}, \dots, s_d^{best}\}$

13: Network parameter optimization:

14: Define the parameters of different network types

15: Encode the network parameters $P = \{p_1, p_2, \dots, p_d\}$

16: Train neural network parameters through algorithm

17: Calculate the objective function acc_{valid}

18: Generate the best network parameters $P^{best} = \{p_1^{best}, p_2^{best}, \dots, p_d^{best}\}$

19: Filter out the optimal network type N_{best} , structure S_{best} , and parameters P_{best}

Output: $M_{N_{best}S_{best}P_{best}}$

vestigation and simple experiments, three types of neural network are selected: feedforward neural network (BP), feedback recurrent neural network (RNN), and convolutional neural network (CNN). When faced with small-scale problems, the neural network needs only a few intermediate layers to obtain a better effect. Secondly, the more intermediate layers, the easier it is to cause overfitting. Therefore, the maximum intermediate layer in this paper is set as 4 layers. For the selection of the number of nodes in the middle layer, this paper adopts an interval search method, and the number of nodes is randomly selected within the range of the interval length of eight, as shown in Table 3, where k represents the decoded value. The activation function and optimizer of the network select several traditional effective methods, as shown in Table 3. The network parameters of the neural network include all its parameters that can be adjusted by the backpropagation algorithm. The parameters of the BP neural network include the weight and bias value of the neuron connection between layers. The network parameters of RNN not only include the same parameters as BP, but also include the previous value as the weight of this input. The network parameters of CNN include the convolution kernel and the connection weight and bias value between the convolution kernel.

According to the neural network structure and neural network parameter settings in this paper, genetic algorithm and firefly algorithm are selected as the algorithms for neural network structure optimization and neural network parameter optimization respectively.

Table 3. Optional parameters and range of neural network structure

<i>Network structure parameter</i>	<i>BP</i>	<i>RNN</i>	<i>CNN</i>
Number of nodes per layer	Neurons	Neurons	Convolution and pooled kernel [8k+1,8k+8]
<i>Activation function</i>	ReLU,LeakyReLU,ELU,TanH,Sigmoid, <i>Hard_Sigmoid</i> ,SoftPlus,Linear		
<i>Optimizer</i>	Adam,SGD,Adagrad,Adadelta,Adamax,Nadam,RMSprop,Ftrl		

3.2 Detailed description of neural network structure optimization algorithm

Genetic algorithm has better global search ability and can quickly search out all the solutions in the solution space, so it is introduced in the stage of neural network structure optimization. When encoding the neural network structure, the chromosome is divided into multiple functional segments, and each segment corresponds to a structural parameter of the neural network. Therefore, this paper designs an appropriate encoding and decoding method as well as genetic operation based on the idea of grouping optimization. The modified genetic algorithm optimizes the neural network structure as shown in Algorithm 2.

Algorithm 2 Network structure optimization

```
1: Define objective function  $f(x) = loss_{valid}$ 
2: Setting algorithm parameters
3: Initialize the GA population  $P = \{p^1, p^2, \dots, p^H\}$ 
4: Train the NNs model and decode the population
5:  $t = 0, Q_t = \{q_t^1, q_t^2, \dots, q_t^H\}$ 
6: while  $t \leq Iter_{max}$  do
7:   for each  $p_t^h (h = 1, 2, \dots, H)$  do
8:      $q_t^h = \phi$ 
9:     for each  $b \in p_t^h$  do
10:      if random  $\leq pc$  then
11:         $p = selection(p_t^h)$ 
12:      else if random  $\leq opc$  then
13:         $p = selection(p_t^{another})$ 
14:      end if
15:       $p^* = cross(b, p)$ 
16:      if random  $\leq pm$  then
17:         $p^* = mutation(p^*)$ 
18:      end if
19:       $q_t^h = q_t^h \cup p^*$ 
20:    end for
21:  end for
22:   $G_t = Mixed\ operation(p_t)$ 
23:   $P_{t+1} = TopN(elite(P_t \cup Q_t \cup G_t))$ 
24:   $t++$ 
25: end while
```

(1) Encoding and decoding

This paper encodes the number of middle layer nodes, activation function, and optimizer of the neural network in sequence. Neural networks of different depths use variable-length coding, in which to optimize the number of nodes in each intermediate layer, a method of interval coding and interval search is designed here. For each intermediate layer, this paper adopts the interval encoding with the node number upper limit of 128, that is, it is represented by the four-digit binary encoding, which stores the partition information of 16 node number intervals. The interval of the number of nodes means that the number of nodes in the middle layer can only be taken within the range of this interval. For example, the decoded value is k ($k \in Z$ and $k \in [0, 15]$), which means that an integer is randomly selected as the number of nodes in the layer in the node number interval $[8k+1, 8k+8]$. The upper limit of the middle layer of the neural network in this paper is four layers, so the coding length of the middle layer nodes of the neural network of one, two, three and four layers are 4, 8, 12 and 16 bits in sequence. In addition, there are eight types of activation functions and optimizers, so they are all represented by three-digit binary numbers. A certain code of different deep neural networks is shown in Figure 2. Each four-bit code in the front represents the interval of the number of nodes in the middle layer, and the last two three-bit codes respectively represent the activation function and the optimizer type, as shown in Table 4.

```

1010 101 010
 1010 0101 100 010
   1101 1010 0110 010 101
    0101 1100 1010 0110 110 011
    
```

Fig. 2. Neural network coding

Interval coding is used for the number of nodes in the middle layer, which can reduce the code length and is conducive to the operation of the algorithm. The number of reductions in the code length of a coding with H intermediate layers is shown in equation (6) m_c , where $l = \log_2 L$ is the original code length, and L is the upper limit of the number of nodes, which is generally taken as the power of 2. $l^* = \log_2 \frac{L}{M}$ is the code length after interval coding, and W represents the length of the interval. In this paper, the upper limit L is 128, the original code length $l = 7$, and the interval coding method with interval division length $W=8$ is adopted. According to the formula, the code length of only one layer of neural network coding is reduced by 3. For the four-layer neural network coding, the code length is reduced by 12, and the number of code reductions becomes more and more obvious as the middle layer increases.

$$m_c = H(l - l^*) = H \log_2 W \tag{6}$$

Table 4. Activation function and optimizer type

Code	Activation	Optimizer
000	<i>sigmoid</i>	<i>Ftrl</i>
001	<i>relu</i>	<i>Adam</i>
010	<i>leakyrelu</i>	<i>SGD</i>
011	<i>elu</i>	<i>Adagrad</i>
100	<i>tanh</i>	<i>Adadelta</i>
101	<i>hard_sigmoid</i>	<i>Adamax</i>
110	<i>softplus</i>	<i>Nadam</i>
111	<i>linear</i>	<i>RMSprop</i>

$$m_s = \left(\frac{2^l}{2^{l^*}}\right)^H = 2^{H \log_2 W} = 2^{m_c} \tag{7}$$

In addition, for the intermediate layer, when the feasible solution space is large, the dimension of the solution space can be reduced to improve the optimization efficiency. As shown in equation (7), m_s represents the multiple of the solution space reduction of a neural network with H intermediate layers. For a neural network coding with a depth of four, the solution space of the number of nodes in the middle layer is usually $(2^7)^4$. After using interval coding, the dimension of the solution space is reduced by 2^{12} times, which greatly improves the optimization efficiency.

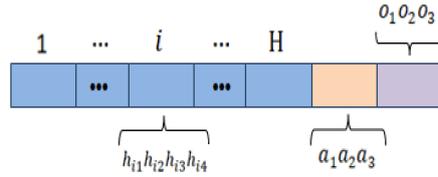


Fig. 3. Network code decoding

$$k_i = h_{i1} * 2^3 + h_{i2} * 2^2 + h_{i3} * 2 + h_{i4}, i = 1, 2, \dots, H \tag{8}$$

$$act = a_1 * 2^2 + a_2 * 2 + a_3 \tag{9}$$

$$opt = o_1 * 2^2 + o_2 * 2 + o_3 \tag{10}$$

The decoding process of neural network coding is shown in Figure 3 and equations (8) to (10), obtaining the interval of the number of nodes in each intermediate layer, as well as the activation function type and the optimizer type.

In equation (8), k_i represents the node number interval of the i -th layer, and h_{ij} represents the j -th bit of the i -th intermediate layer binary code. In equation (9), act represents the selected activation function type, and in equation (10), opt represents the selected optimizer type.

(2) Grouped optimization genetic operations

The traditional optimization of neural network structure is that the depth of the network is continuously optimized and eliminated along with the iteration of the algorithm, which easily causes the depth of the network to fall into the local optimum. A certain depth of neural network that performs well in the early stage of population evolution is easier to win in the subsequent competition. However, those individuals that are completely eliminated in the initial stage are eliminated without sufficient evolution. This makes the algorithm extremely easy to fall into the local chromosome segment of the depth of the neural network, which causes the whole to fall into the local optimum. Therefore, in order to overcome the above-mentioned problems, this paper follows the idea of multi-objective optimization and introduces the genetic operation of group optimization.

According to the network depth, the initial subpopulations of the same size H (representing the upper limit of the number of network intermediate layers) are generated to form the initial population of the genetic algorithm. In the subsequent process of population evolution, selection, crossover and variation always revolve around the inter-group and intra-group relations. At the same time, in order to improve the breadth search ability of the algorithm, a hybrid group is introduced in the middle, which discards the intra-group and inter-group relationship and carries out the traditional genetic operation. The concrete steps of the modified genetic algorithm to optimize the neural network structure are as follows:

Step 1: Generate an initial population P of size N , and calculate its fitness. There are at most H intermediate layers in the network structure, so in the initial population, H subpopulations of the same size are generated according to the number of layers, that is, $P = p^1, p^2, \dots, p^H$.

Step 2: Select operator, using the tournament selection method. For each group, an individual $r1$ is selected from that group, and another individual $r2$ is selected from other groups or from the same group according to intergroup crossover or intra-group crossover.

Step 3: Crossover operator. In the case of crossover probability pc , single-point crossover within the group was carried out according to the probability (gpc), and two-point crossover between the groups was carried out according to the probability (pc-gpc). In order to improve the ability of searching each functional region of chromosomes in depth, only segments with the same meaning are crossed.

Step 4: Mutation operator. For the two new individuals generated by the crossover, keep the best one, and mutate according to the probability pm , and the final individual is put into the intermediate population Q_t . Selection, crossover, and mutation are repeated until an intermediate population of the same size as the parent population is produced.

Step 5: Introduce a mixed group, which has the same size as the other groups. Step 3 is to improve the algorithm's in-depth search ability, and only crosses between segments with the same meaning. Therefore, here to improve the algorithm's breadth search ability, the Step 3 cross rule is abandoned, and the parent population is mixed cross to obtain a mixed group G_t .

Step 6: Elite strategy, population renewal. For the parent population P_t and the intermediate population $Q_t = q_t^1, q_t^2, \dots, q_t^H$ and the mixed group G_t , the elite mechanism is used to form a synthetic population $P_t \cup Q_t \cup G_t$ and group by group sort them and select the first $N/4$ excellent individuals to form the next-generation population $P_{t+1} = p_{t+1}^1, p_{t+1}^2, \dots, p_{t+1}^h$.

Step 7: Iteration termination judgment. When the number of iterations reaches the maximum number of iterations or finally converges, the iteration is terminated, otherwise, go to **Step2**.

3.3 Detailed description of neural network parameter optimization algorithm

The firefly algorithm is a new type of swarm intelligence optimization algorithm, which has the characteristics of simple operation, not easy to fall into the local optimum and good stability, so it is introduced to optimize the neural network parameters globally. In this stage, real number encoding is adopted to encode all adjustable parameters of the neural network, and the encoding length depends on the number of parameters of the neural network. The optimization process of network parameters of the modified firefly algorithm is shown in algorithm 3.

The traditional firefly algorithm can know from its position update formula that the disturbance to the firefly is completely random and has no correlation with any information contained in the firefly itself, which has great blindness. In addition, there is no reasonable standard for the selection of disturbance factor, and when the disturbance factor is fixed, it is not conducive to adapt to the changes of firefly population environment. Considering the above two issues comprehensively, this paper proposes a random disturbance with firefly position information and an adaptive disturbance factor.

(1) Position update formula based on firefly information

Faced with different problems, the meaning of the location information of the fireflies is also different. When faced with all problems, we adopt completely random disturbances, and it is difficult to obtain a better position than before. Therefore, in view of the randomness and guidance, the location update formula adds the location information of the firefly itself. While the fireflies are attracted by other fireflies and move toward them, the fireflies perform random disturbances with partial position information. The amount of information contained depends on the information scale factor (that is, the disturbance step length), and the position movement formula is as follows:

$$x_i^{t+1} = (1 + \varepsilon^t(\alpha)) \left(x_i^t + \beta_0 e^{-\gamma r_{ij}^2} (x_j^t - x_i^t) \right) \quad (11)$$

Algorithm 3 Network parameter optimization

```
1: Define objective function  $f(x)=loss_{valid}$ 
2: Setting algorithm parameters
3: Initialize the firefly population  $P = \{x_i|i = 1, 2, \dots, n\}$ 
4: Light intensity  $I_i$  at  $x_i$  is determined by  $f(x_i)$ 
5:  $t = 0$ 
6: while  $t < Iter_{max}$  do
7:    $Q_t = \phi$ 
8:   for each  $x_i$  ( $i=1,2,,n$ ) do
9:     for each  $x_j$  ( $j = i + 1, , n$ ) do
10:      if  $I_i < I_j$  then
11:         $x^* =$  move firefly  $x_i$  towards  $x_j$ 
12:      else if  $I_i > I_j$  then
13:         $x^* =$  move firefly  $x_j$  towards  $x_i$ 
14:      else  $x^* =$  random move  $x_i$ 
15:      end if
16:       $Q_t = x^* \cup Q_t$ 
17:    end for
18:  end for
19:   $P_{t+1} = elite(P_t \cup Q_t)$ 
20:   $t = t + 1$ 
21: end while
```

Where, the information scaling factor $\alpha \in [0, 1]$, $\xi^t(\alpha)$ is a random number vector evenly distributed between $[-\alpha, \alpha]$. β_0 is the maximum attraction of fireflies (i.e., $r_{ij}=0$), which is generally set to 1. $\gamma = \frac{0.2}{(U_w - L_w)^2}$, where U_w and L_w represent upper and lower limits of location information, respectively.

When the brightness of the two fireflies are equal, the fireflies will perform random disturbances. the disturbance formula is as follows:

$$x_i^{t+1} = 1 + \varepsilon^t(\alpha) x_i^t, \quad \alpha = \begin{cases} \alpha_u, & \text{if } p \geq p_r \\ \alpha_l, & \text{if } p < p_r \end{cases} \quad (12)$$

Where, $\alpha_l < \alpha_u$, with probability p_r for small-range disturbance, with probability $1 - p_r$ for large-range disturbance, jumping out of the local area. In the above position movement formula, when α takes different values, a large range of disturbance and a small range of local disturbance can be realized.

(2) Adaptive disturbance factor

Randomization plays an important role in the exploration and development of algorithms. The essence of this kind of randomization is random walk. Random walk is the number of steps to make the firefly move around at its current position. The amount of movement is affected by the step disturbance factor. When the step disturbance factor is appropriate, the algorithm's global search ability can be improved. In the evolutionary process of the population, the fixation of the disturbance factor does not adapt to the change and development of the population environment. When the disturbance factor is too small, the convergence rate is slowed down in the early stage of population evolution, and it is easy to fall into the local optimum; when the disturbance factor is too large, because the population has tended to converge in the later stage, too large step size is not conducive to the later local optimization. Therefore, the disturbance factor should be constantly changing with the environment during the evolution of the entire population. The adaptive formula is as follows:

$$\alpha = \frac{\alpha_0}{1 + e^{6(\frac{it}{0.4 * it_{max}} - 1)}} + 0.05 \quad (13)$$

Where, it and it_{max} represent the current number of iterations and the maximum number of iterations, respectively, and 0.05 represents the lower limit of the disturbance factor. No matter how many iterations are, the disturbance will not be invalidated. At the same time, 0.05 and α_0 together constitute the initial value of the disturbance. Figure 4 shows the adaptive disturbance factor with $\alpha_0 = 0.45$ and the maximum number of iterations $it_{max} = 100$.

As can be seen from Fig. 4, in the early stage of population evolution, the decreasing speed of disturbance factor curve is relatively gentle, making it maintain a relatively large value in the early stage, which is conducive to the global search of the algorithm. In the middle stage, the population evolved to a relatively ideal state, and the disturbance factor decreased rapidly. In the later stage, the decline rate of the disturbance factor was relatively gentle, maintaining a relatively small value for a long period of time, which was conducive to local search and accelerated convergence.

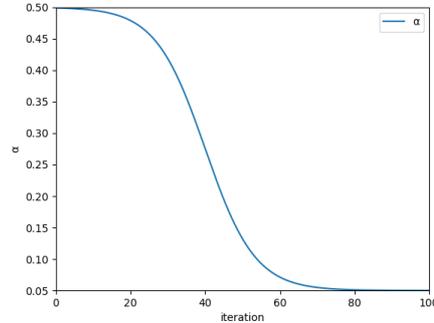


Fig. 4. Adaptive perturbation factor

Firefly algorithm optimization neural network weight steps are as follows:

Step 1: Generate an initial population P of size N , and calculate its fluorescence. For the neural network model obtained in the structure optimization stage, the traditional optimizer is used to train the network $2N$ times to obtain $2N$ sets of parameters, which are sorted by fluorescence, and the first N are taken as the initial population of the second stage.

Step 2: Firefly location updated. The fireflies were compared in pairs. The less bright fireflies flew to the brighter fireflies according to the motion formula. One of the fireflies with the same brightness is perturbed in a small range with probability p_r , and in a large range with probability $1 - p_r$. As fireflies fly, they form a new intermediate population, Q_t .

Step 3: The elite strategy is adopted to update the population. Sort the synthetic population ($P_t \cup Q_t$), and select the first N individuals as the next generation population P_{t+1} .

Step 4: Iteration termination judgment. When the number of iterations reaches the maximum value or finally converges, the iteration is terminated, otherwise, go to **Step 2**.

4 Experiment and result analysis

This section verifies the effectiveness and feasibility of the deep collaborative optimization framework through simulation experiments and data analysis.

The experiment first extracts the features to make the features more concentrated, which is convenient for the training of the neural network; secondly, through the neural network structure and parameter optimization, the effectiveness of the parameter optimization is verified; finally, the best neural network wine prediction model is selected and compared with the experimental data in the paper[5]. The experimental environment for all experiments is: Experimental software: PyCharm and SPSS; Programming language: Python; Processor: Intel(R) Core(TM) i5-9300h CPU 2.40GHz; Operating system: Windows10, 64-bit.

In order to ensure the influence of hyperparameters on experimental results, the program parameters of the selected algorithm were tuned in advance, as shown in Table 5. For the three important hyperparameters of random forest, this paper adopts grid search with cross-validation (GridSearchCV) to determine, as shown in RF in Table 5, the first number represents the Red wine data set parameters, and the back number represents the White wine data set parameter. For NNS, GA and FA parameters, after repeated tests and adjustments, the final setting results are shown in Table 5.

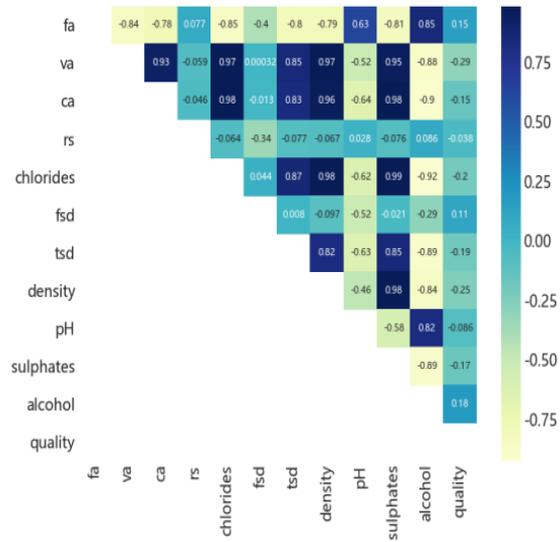
Table 5. Experimental parameter Settings

Classified	Parameter names	Value
NNs	train_epoch	200
	max_features	3—2
RF	n_estimators	1400—4800
	min_sample_leaf	3—1
	N	80
GA	iter	50
	pc	0.80
	gpc	0.65
	pm	0.05
FA	N	50
	iter	50
	p_r	0.70
	α_0	0.35

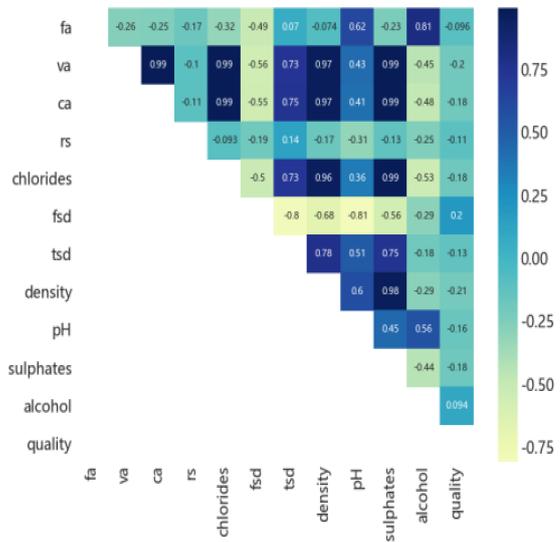
4.1 Data feature extraction

The Pearson correlation coefficients between wine data features are shown in Figure 5. fa, va, ca, rs, fsd, and tsd respectively represent the data features fixed acidity, volatile acidity, citric acid, residual sugar, free sulfur dioxide and total sulfur dioxide. For the two data sets, Sulphates, volatile acidity, Chlorides, Density and Citric acid characteristics showed high correlation with each other, and Pearson correlation coefficients were all above 0.9.

The importance of features obtained by random forest is shown in Table 6 and Figure 6. For the two data sets, four of the top five feature importance are the same: alcohol, volatile acidity, total sulfur and density. It can be seen that these four physicochemical properties of wine play an important role in the quality of both red and white wines. For red wine, the physicochemical property of alcohol is more important to the quality of wine, as shown in Figure 6(a). For



(a)



(b)

Fig. 5. Pearson coefficient heat map:(a) stands for red wine,(b) stands for white wine

white wines, the importance measures of various physicochemical properties did not differ significantly, as shown in Figure 6(b). Sulphates was the second most important for red wine and the tenth most important for white wine, and there were differences in the kind of physicochemical properties that affected the quality of red and white wine.

Table 6. Importance of random forest features

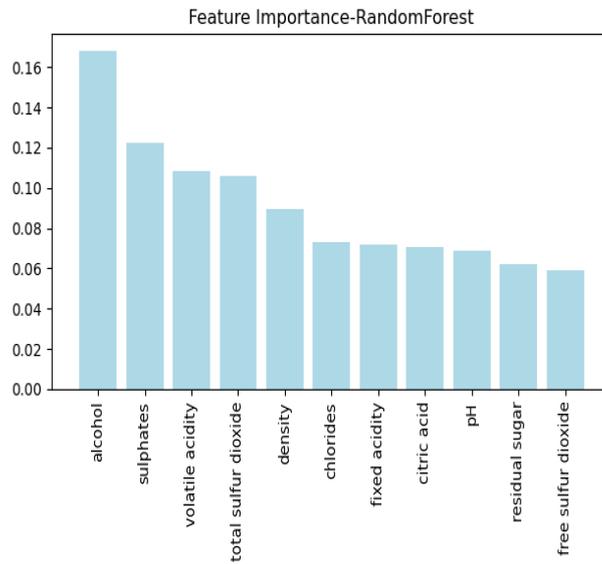
Red wine		White wine	
Features	Importance	Features	Importance
alcohol	0.167971	alcohol	0.112694
sulphates	0.122381	density	0.104024
volatile acidity	0.108299	volatile acidity	0.097358
total sulfur dioxide	0.106158	free sulfur dioxide	0.093790
density	0.089543	total sulfur dioxide	0.092351
chlorides*	0.073102	residual sugar	0.088275
fixed acidity	0.072021	chlorides	0.086534
citric acid*	0.070762	pH	0.085546
pH	0.068723	citric acid*	0.082638
residual sugar	0.062264	sulphates*	0.081043
free sulfur dioxide	0.058776	fixed acidity	0.075746

* Synthetically consider the data features to be removed

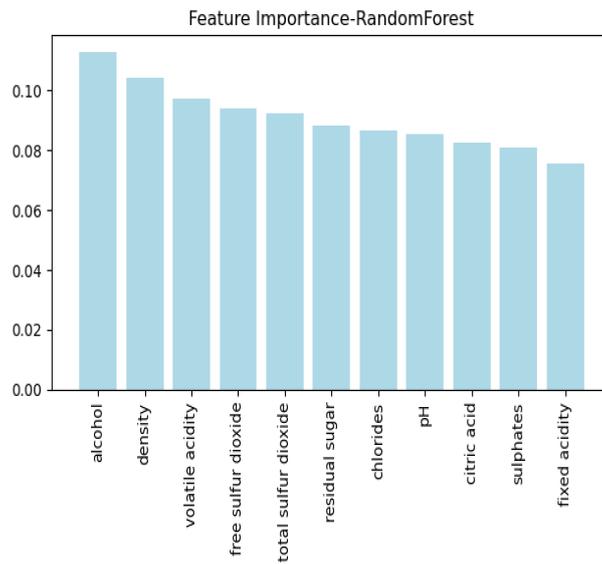
In summary, in view of the importance and relevance of features, in red wine, Chlorides and Citric acid, which are strongly correlated and ranked low in importance, are removed; In white wine, citric acid and sulphates, which are strongly correlated and ranked ninth and tenth in importance, are removed, as shown in Table 6.

4.2 Neural network structure and parameter optimization

For each type of neural network and each depth of the network, 30 experiments were performed, and the summary of the experiments of each depth of the network is the result with a confidence of 95%. Mean square error(MSE) and accuracy are used to measure the quality of the neural network model, and three different error tolerances (i.e., T=0.25, 0.5 and 1.0) are set for the accuracy of the regression problem. The error tolerance is an absolute deviation of the actual value, and the predicted value within the error tolerance is considered to be accurate. At the same time, in order to improve the generalization ability of



(a)



(b)

Fig. 6. Feature importance:(a) stands for red wine,(b) stands for red wine

the network model, the data set is divided into training set, verification set and test set. Among them, the test set accounts for 20%, and in the remaining 80% of the samples, another 20% is divided as the verification set, and the rest is the training set. The training set is used to train the neural network, the loss function and accuracy of the verification set are used as the objective function of the optimization algorithm, and the test set is used to evaluate the generalization ability and accuracy of the final network model.

The results of the neural network structure and parameter optimization are shown in Table 7 to 12. Delta represents the difference between the average accuracy of parameters and structural optimization. Among them, the MSE in the structural optimization and parameter optimization results is not much different. This is because in the structural optimization part, the MSE whose objective function is the verification set has achieved a relatively ideal state. Therefore, the parameter optimization takes the accuracy value of the verification set as the objective function, and the result can be seen from the table that the average accuracy value of each neural network corresponding to the test set has been further improved. For white wine, the parameter optimization effects of BP and CNN are relatively more obvious, especially the BP network has been improved on the training set, validation set, test set, and upper and lower limits, as shown in Table 10. Figures 7 and 8 respectively show the red wine and white wine data sets for 10 different divisions, and the best neural network model optimization results for each neural network type. It can also be seen from the figure that parameter optimization can bring about an improvement in prediction accuracy.

Table 7. BP results of Red wine

Red wine(T=0.5)	Optimize structure				Optimize parameters			
		\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	$delta\%$
1,[114], ELU, Adam	train	0.661	[58.06, 61.10]	59.49	0.688	[57.37, 60.80]	59.59	0.1
	valid	0.693	[51.17, 64.45]	58.03	0.677	[56.25 , 63.67]	60.14	2.11
	test	0.686	[55.00, 64.69]	59.02	0.669	[57.50, 65.94]	60.45	1.43
2,[56, 15], ELU, Nadam	train	0.661	[57.37, 61.29]	59.97	0.665	[58.85, 61.49]	60.13	0.16
	valid	0.657	[55.47, 64.84]	59.56	0.675	[54.69, 63.28]	59.04	-0.52
	test	0.656	[55.31, 62.81]	59.68	0.649	[57.19, 64.06]	60.85	1.17
3,[23, 118, 11], ELU, Nadam	train	0.657	[57.68, 61.89]	59.45	0.674	[58.54, 61.59]	60.29	0.84
	valid	0.673	[53.52, 63.28]	58.49	0.657	[55.86, 63.67]	59.88	1.39
	test	0.661	[54.06, 62.81]	59.42	0.637	[58.13, 66.88]	62.16	2.74
4,[53, 27, 53, 16], LeakyReLU, Adamax	train	0.634	[58.85, 62.17]	60.20	0.658	[57.87, 60.91]	59.31	-0.89
	valid	0.644	[55.08, 63.28]	59.57	0.671	[54.69, 62.50]	58.66	-0.91
	test	0.648	[54.06, 62.81]	59.27	0.653	[55.63, 64.06]	60.34	1.07

Table 8. CNN results of Red wine

Red wine(T=0.5)		Optimize structure			Optimize parameters			
		\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	$\delta\%$
1,[86],tanh,Adam	train	0.679	[55.52,58.15]	56.77	0.692	[55.42, 58.94]	57.17	0.4
	valid	0.680	[51.56,62.11]	56.69	0.682	[53.13,64.45]	58.22	1.53
	test	0.675	[53.75,62.50]	56.92	0.682	[54.06,62.81]	57.85	0.93
2,[29, 29], tanh, Adam	train	0.628	[57.37,60.79]	58.70	0.648	[56.01,59.71]	58.21	-0.49
	valid	0.628	[51.56,63.67]	59.54	0.625	[53.52,66.80]	58.76	-0.78
	test	0.634	[53.44,62.19]	57.83	0.635	[54.38,63.44]	58.78	0.95
3,[102, 102, 34], ELU, Adam	train	0.645	[57.78,61.58]	59.75	0.639	[57.09,60.81]	58.85	-0.9
	valid	0.634	[53.91,64.06]	58.49	0.643	[53.52, 64.45]	58.93	0.44
	test	0.645	[55.94,62.81]	59.08	0.635	[55.31, 64.06]	59.73	0.73
4,[104, 51, 51, 24],ELU, Adam	train	0.601	[59.34,62.09]	61.30	0.610	[59.53,62.74]	60.75	-0.55
	valid	0.614	[54.30,64.84]	59.84	0.594	[57.03,67.58]	61.35	1.51
	test	0.607	[55.94,64.06]	60.51	0.594	[58.75,65.94]	62.03	1.52

4.3 Neural network model comparison

For red wine data, the three types of neural networks have the best effect when the depth is 3 (BP_3) and 4 (CNN_4 and RNN_4), respectively, as shown in Table 13, 14 and 15. For CNN_4 and RNN_4 , the average accuracy at T=0.25 is at least 4.67% and 5.75% higher than that of other deep networks, respectively. The accuracy of RNN_4 at T=0.5 is above 64.50%, and the accuracy of other different depth networks is below 64.00%. At T=1.0, the accuracy is above 93.30%, and the accuracy of other deep networks is below 91.1%. For the white wine data, the three types of neural networks performed best at depth 2 (BP_2) and 4 (CNN_4 and RNN_4), respectively, as shown in Table 16, 17 and 18. For CNN, CNN_4 is dominant when T=0.25 and T=1.0, and CNN_3 is slightly dominant when T=0.5. When the depth of RNN is 4, the average accuracy within the three different error tolerances T is at least 7.9%, 7.12% and 1.55% higher than that of other similar deep networks, respectively, as shown in Table 18.

Figure 7 and 8 show the optimal results of network model structure optimization and parameter optimization among the three types of neural networks. Phase1 represents the network structure optimization of the neural network, and Phase2 represents the neural network parameter optimization. For the two kinds of data, neural network parameter optimization has a greater improvement on BP network in the three kinds of networks, as shown in Fig. 7 (a) and 8 (a). Meanwhile, it can be seen from the figure that the accuracy value of RNN_4 is significantly higher than that of other optimal neural network models.

Table 9. RNN results of Red wine

Red wine(T=0.5)		Optimize structure			Optimize parameters			
		\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	$delta\%$
1,[60],leaky relu,Adam	train	0.641	[58.75,62.47]	60.27	0.659	[58.64,62.16]	60.39	0.12
	valid	0.654	[53.13,64.06]	59.87	0.657	[55.08,65.63]	60.55	0.68
	test	0.628	[54.38,64.69]	60.05	0.646	[57.19,65.63]	61.31	1.26
2,[110, 110],leaky relu, RMSprop	train	0.657	[57.68,60.90]	59.30	0.664	[57.57, 61.29]	59.51	0.21
	valid	0.659	[53.91,62.11]	59.10	0.658	[54.69,66.41]	60.33	1.23
	test	0.656	[55.31,63.13]	59.39	0.648	[55.00, 65.63]	59.92	0.53
3,[78, 78, 45],leaky relu, RMSprop	train	0.629	[60.02,63.94]	61.94	0.625	[60.87,64.52]	62.18	0.24
	valid	0.630	[54.69,66.41]	61.81	0.634	[55.47,66.80]	61.82	0.01
	test	0.620	[57.19,66.88]	62.87	0.605	[58.44,65.63]	63.00	0.13
4,[37, 37, 12, 118],tanh, Nadam	train	0.500	[62.96,67.46]	65.17	0.507	[63.44,65.69]	64.48	-0.69
	valid	0.508	[58.98,71.09]	65.16	0.501	[58.98,70.70]	64.44	-0.72
	test	0.507	[60.00,70.31]	64.85	0.494	[61.25,70.63]	65.46	0.61

Table 10. BP results of White wine

white wine(T=0.5)		Optimize structure			Optimize parameters			
		\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	$delta\%$
1,[85],leaky relu,Nadam	train	0.775	[49.88,52.74]	51.59	0.807	[52.63,56.85]	55.96	4.37
	valid	0.786	[48.25,54.75]	51.02	0.804	[55.24,59.00]	56.19	5.17
	test	0.767	[49.56,54.38]	52.15	0.794	[54.29,59.09]	56.91	4.76
2,[52, 14], leaky relu, Nadam	train	0.754	[53.12,55.71]	54.65	0.768	[55.74,57.94]	57.12	2.47
	valid	0.751	[50.38,58.38]	54.51	0.772	[52.63,60.13]	56.91	2.4
	test	0.760	[52.22,56.73]	54.67	0.763	[54.90,60.57]	57.18	2.51
3,[123, 27, 27],leaky relu,Adam	train	0.750	[53.58,56.09]	55.03	0.767	[56.07,57.53]	56.82	1.79
	valid	0.739	[50.75,58.75]	55.16	0.758	[54.63,59.75]	56.90	1.74
	test	0.737	[52.96,57.32]	55.24	0.756	[54.64,59.62]	57.33	2.09
4,[36, 95, 36, 36], leaky relu, Nadam	train	0.735	[53.66,56.16]	54.87	0.742	[56.05,58.10]	56.94	2.07
	valid	0.722	[52.63,58.13]	55.48	0.747	[53.50,59.75]	56.79	1.31
	test	0.727	[52.10,57.76]	55.41	0.758	[54.36,59.03]	56.75	1.34

Table 11. CNN results of White Wine

white wine(T=0.5)		Optimize structure			Optimize parameters			
		\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	$delta\%$
1,[65],tanh,RMSprop	train	0.783	[52.56,54.66]	53.39	0.779	[52.19, 54.79]	53.81	0.42
	valid	0.781	[50.00,56.75]	53.02	0.778	[50.63,56.75]	53.78	0.76
	test	0.783	[50.54,56.53]	53.37	0.776	[50.71,57.60]	54.05	0.68
2,[44,20],tanh,Adam	train	0.712	[55.62,57.50]	56.54	0.714	[55.49, 57.88]	56.69	0.15
	valid	0.708	[53.88,60.50]	57.15	0.704	[54.38,60.50]	57.21	0.06
	test	0.705	[53.79,59.68]	56.39	0.709	[53.51, 59.86]	56.82	0.43
3,[9, 57, 123],tanh,Adam	train	0.712	[57.66,59.45]	58.54	0.712	[57.37, 59.76]	58.84	0.30
	valid	0.719	[55.13,62.13]	58.57	0.719	[55.13,61.63]	58.70	0.13
	test	0.719	[56.05,60.34]	58.61	0.693	[56.29,62.56]	59.16	0.55
4,[91, 56, 56, 16],elu, Adam	train	0.652	[55.90,58.68]	57.68	0.662	[57.19,59.20]	58.28	0.60
	valid	0.668	[53.88,59.88]	57.63	0.659	[55.13,61.75]	58.30	0.67
	test	0.658	[53.61,61.15]	57.49	0.658	[55.71,60.69]	58.38	0.89

Table 12. RNN results of White Wine

white wine(T=0.5)		Optimize structure			Optimize parameters			
		\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	\overline{MSE}	$[min_{acc}\%, max_{acc}\%]$	$\overline{acc}\%$	$delta\%$
1, [18], tanh, Nadam	train	0.772	[53.03,55.13]	54.35	0.780	[52.96,54.94]	54.01	-0.34
	valid	0.765	[50.88,57.13]	54.74	0.764	[52.38,58.25]	55.00	0.26
	test	0.766	[51.77,57.80]	54.69	0.752	[52.16,57.56]	55.07	0.38
2,[13, 70], tanh, RMSprop	train	0.737	[55.04,58.04]	56.65	0.737	[54.87,57.05]	56.26	-0.39
	valid	0.727	[53.63,60.50]	56.83	0.731	[54.00,59.38]	56.01	-0.82
	test	0.732	[53.75,60.08]	56.42	0.732	[54.13,60.12]	56.73	0.31
3,[48, 17, 48], elu, RMSprop	train	0.771	[53.66,56.61]	54.76	0.765	[54.19,55.97]	54.99	0.23
	valid	0.758	[50.38,58.38]	55.20	0.783	[51.50,57.00]	54.50	-0.7
	test	0.764	[53.02,59.32]	55.44	0.759	[52.46,58.81]	55.69	0.25
4,[23, 96, 96, 41], tanh, Nadam	train	0.618	[61.68,64.32]	63.56	0.620	[61.67, 64.74]	63.22	-0.34
	valid	0.616	[59.88,67.25]	63.48	0.618	[58.50, 69.00]	63.11	-0.37
	test	0.618	[60.54,67.22]	63.25	0.610	[61.05,67.46]	63.85	0.6

Table 13. Summary of BP results for Red wine

Test sets	BP_1		BP_2		BP_3		BP_4	
	Mean	Interval	Mean	Interval	Mean	Interval	Mean	Interval
$Accuracy\%_{T=0.25}$	21.12	[20.51,21.71]	20.19	[19.56,21.15]	36.33	[35.29,37.33]	32.82	[32.01,33.70]
$Accuracy\%_{T=0.50}$	60.31	[59.66,60.75]	60.83	[60.18,61.53]	61.60	[60.62,62.63]	60.37	[59.47,61.22]
$Accuracy\%_{T=1.00}$	88.22	[87.66,88.97]	88.22	[87.57,88.87]	89.00	[88.16,89.70]	88.24	[87.67,88.92]

Table 14. Summary of CNN results for Red Wine

Test sets	CNN_1		CNN_2		CNN_3		CNN_4	
	Mean	Interval	Mean	Interval	Mean	Interval	Mean	Interval
$Accuracy\%_{T=0.25}$	28.60	[27.89,29.34]	31.28	[30.52,31.96]	31.76	[30.76,32.82]	36.43	[35.59,37.37]
$Accuracy\%_{T=0.50}$	57.82	[57.17,58.54]	58.78	[57.84,59.72]	59.72	[58.83,60.63]	62.00	[61.35,62.71]
$Accuracy\%_{T=1.00}$	87.84	[87.22,88.46]	89.39	[88.77,89.91]	89.64	[89.12,90.21]	89.92	[89.27,90.46]

Table 15. Summary of RNN results for Red wine

Test sets	RNN_1		RNN_2		RNN_3		RNN_4	
	Mean	Interval	Mean	Interval	Mean	Interval	Mean	Interval
$Accuracy\%_{T=0.25}$	37.22	[36.22,38.20]	28.43	[27.65,29.28]	31.00	[30.26,31.78]	42.97	[41.93,44.04]
$Accuracy\%_{T=0.50}$	61.27	[60.51,62.11]	59.86	[58.91,60.92]	63.07	[62.41,63.59]	65.44	[64.54,66.38]
$Accuracy\%_{T=1.00}$	89.55	[89.01,90.11]	88.45	[87.94,88.95]	90.57	[89.93,91.02]	93.73	[93.33,94.08]

Table 16. Summary of BP results for White wine

Test sets	BP_1		BP_2		BP_3		BP_4	
	Mean	Interval	Mean	Interval	Mean	Interval	Mean	Interval
$Accuracy\%_{T=0.25}$	21.84	[21.42,22.22]	22.42	[22.13,22.78]	20.53	[20.15,20.91]	21.78	[21.34,22.42]
$Accuracy\%_{T=0.50}$	56.92	[56.41,57.40]	57.38	[56.91,57.92]	57.35	[56.89,57.78]	56.75	[56.33,57.16]
$Accuracy\%_{T=1.00}$	83.95	[83.60,84.28]	85.95	[85.45,86.30]	85.63	[85.38,85.86]	85.72	[85.40,86.03]

Table 17. Summary of CNN results for White Wine

Test sets	CNN_1		CNN_2		CNN_3		CNN_4	
	Mean	Interval	Mean	Interval	Mean	Interval	Mean	Interval
$Accuracy\%_{T=0.25}$	21.51	[21.02,21.91]	22.48	[22.07,22.85]	20.15	[19.70,20.57]	28.83	[28.36,29.36]
$Accuracy\%_{T=0.50}$	54.04	[53.48,54.63]	56.84	[56.27,57.38]	59.13	[58.57,59.74]	58.41	[57.91,58.87]
$Accuracy\%_{T=1.00}$	86.03	[85.75,86.34]	87.15	[86.80,87.50]	87.27	[86.98,87.60]	89.77	[89.34,90.17]

Table 18. Summary of RNN results for White wine

Test sets	RNN_1		RNN_2		RNN_3		RNN_4	
	Mean	Interval	Mean	Interval	Mean	Interval	Mean	Interval
$Accuracy\%_{T=0.25}$	21.33	[20.90,21.76]	22.24	[21.69,22.75]	24.58	[24.21,24.97]	32.48	[32.02,33.01]
$Accuracy\%_{T=0.50}$	55.04	[54.45,55.58]	56.69	[56.27,57.19]	55.70	[55.12,56.25]	63.81	[63.38,64.32]
$Accuracy\%_{T=1.00}$	86.69	[86.33,87.06]	86.98	[86.61,87.31]	86.61	[86.31,86.88]	88.53	[88.21,88.89]

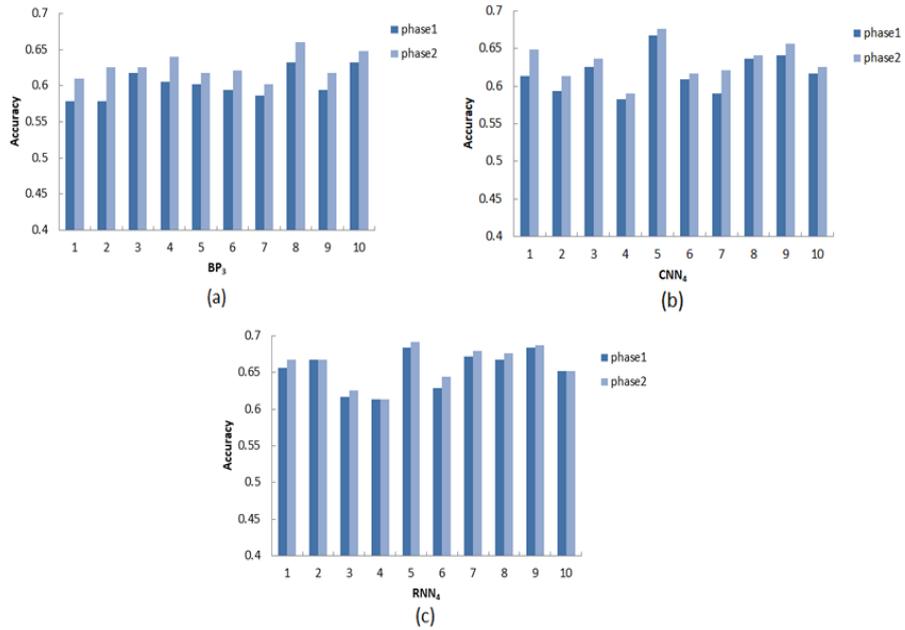


Fig. 7. Red Wine structure and parameter optimization

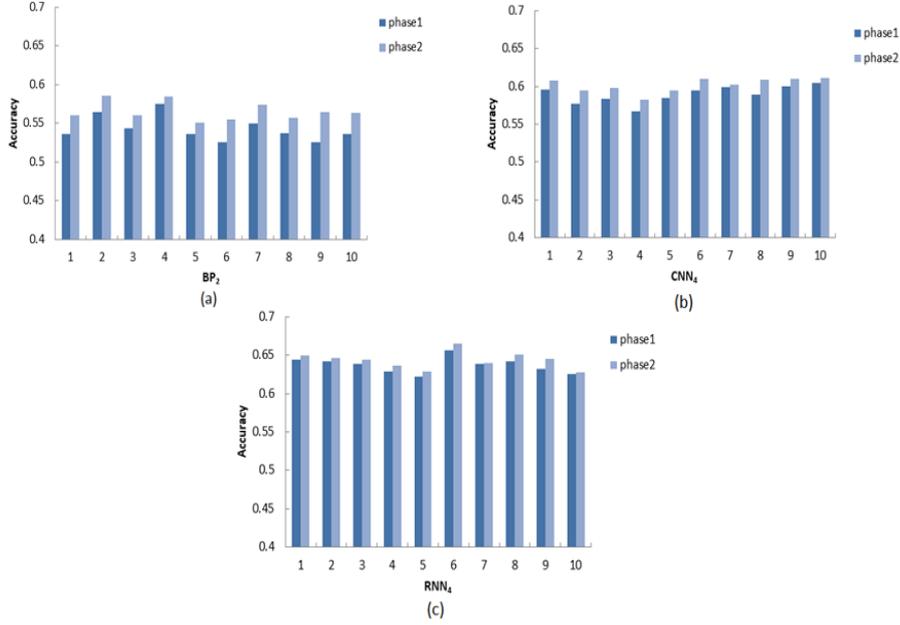


Fig. 8. White Wine structure and parameter optimization

The box diagrams of the three types of optimal neural network models are shown in Fig. 9, where (a), (b) and (c) respectively represent the results corresponding to the three error limits on the Red wine data set. It is obvious that the neural network model RNN₄ shows the best results. Figure 9(d), (e) and (f) respectively represent the results corresponding to the three different tolerances on the White Wine dataset. The neural network model RNN₄ shows the best performance when T=0.25 and 0.5, but there are four outliers when T=0.5, and the neural network model CNN₄ has the best effect when T=1.0. In conclusion, the neural network model CNN₄ has good performance in most cases.

The experimental data of the three optimal neural network models are shown in Table 19 and 20. For Red wine data, it is obvious that RNN₄ shows better results. When T=0.25, the accuracy of RNN₄ is above 41.9%, BP₃ and CNN₄ are below 37.4%, and the average accuracy is higher by 6.64% and 6.54%, respectively. Compared with BP₃ and CNN₄, the average accuracy of RNN₄ is 3.84% and 3.44% higher when T=0.5, and 4.73% and 3.81% higher when T=1.0, respectively. For White wine data, RNN₄ showed better performance at T=0.25 and 0.5, and the average precision value was at least 3.65% and 5.40% higher than that of the contrast model, respectively. When T=1.0, CNN₄ performs better, and the accuracy value is above 89.30%, while RNN₄ is below 89.00.

The detailed prediction results of the optimal prediction network model RNN₄ of neural network at T=0.5 are shown in the confusion matrix in Ta-

Table 19. Optimal network models of three types for Red Wine

Test sets	BP_3		CNN_4		RNN_4	
	Mean	Interval	Mean	Interval	Mean	Interval
$Accuracy\%_{T=0.25}$	36.33	[35.29,37.33]	36.43	[35.59,37.37]	42.97	[41.93,44.04]
$Accuracy\%_{T=0.50}$	61.60	[60.62,62.63]	62.00	[61.35,62.71]	65.44	[64.54,66.38]
$Accuracy\%_{T=1.00}$	89.00	[88.16,89.70]	89.92	[89.27,90.46]	93.73	[93.33,94.08]

Table 20. Optimal network models of three types for White Wine

Test sets	BP_2		CNN_4		RNN_4	
	Mean	Interval	Mean	Interval	Mean	Interval
$Accuracy\%_{T=0.25}$	22.42	[22.13,22.78]	28.83	[28.36,29.36]	32.48	[32.02,33.01]
$Accuracy\%_{T=0.50}$	57.38	[56.91,57.92]	58.41	[57.91,58.87]	63.81	[63.38,64.32]
$Accuracy\%_{T=1.00}$	85.95	[85.45,86.30]	89.77	[89.34,90.17]	88.53	[88.21,88.89]

ble 21. The model’s prediction scores for wine were mainly 5, 6 and 7. For the Red wine data set, the model has a good prediction effect on the samples labeled 5 and 6, with the recall rate reaching 82.1% and 74.3%, respectively. Meanwhile, the precision value of the samples predicted 5 and 6 reaches 71.4% and 58.1%, respectively. For the White data set, the model has a better prediction effect in the samples labeled 5, 6 and 7, with the recall rates of 61.6%, 69.9% and 41.9%, respectively. In the data predicted by the model for these three labels, the accuracy rates are 58.3%, 58.3% and 51.3%, respectively.

At the same time, it can be seen from Table 2 that samples with true labels 5 and 6 account for 82.5% in the Red wine data set, and samples with true labels 5, 6 and 7 account for 90.9% in the White wine data set. This prediction model is more sensitive to such large number of samples. This is because in the training process, the model can capture more information for a certain label with a large number of samples in the data set. Therefore, it can grasp the characteristics of such label data more accurately and show better results in the final model evaluation.

4.4 Comparison of optimal model experiments

The experimental results show that the neural network models RNN_4 (4, [37, 37, 12, 118], tanh, Nadam) and RNN_4 (4, [91, 56, 56, 16], elu, Adam) have the best effect on Red wine and White wine data sets respectively. This paper compares this model with the experimental results of MR, NN and SVM in paper [5], as shown in Table 22 and 23.

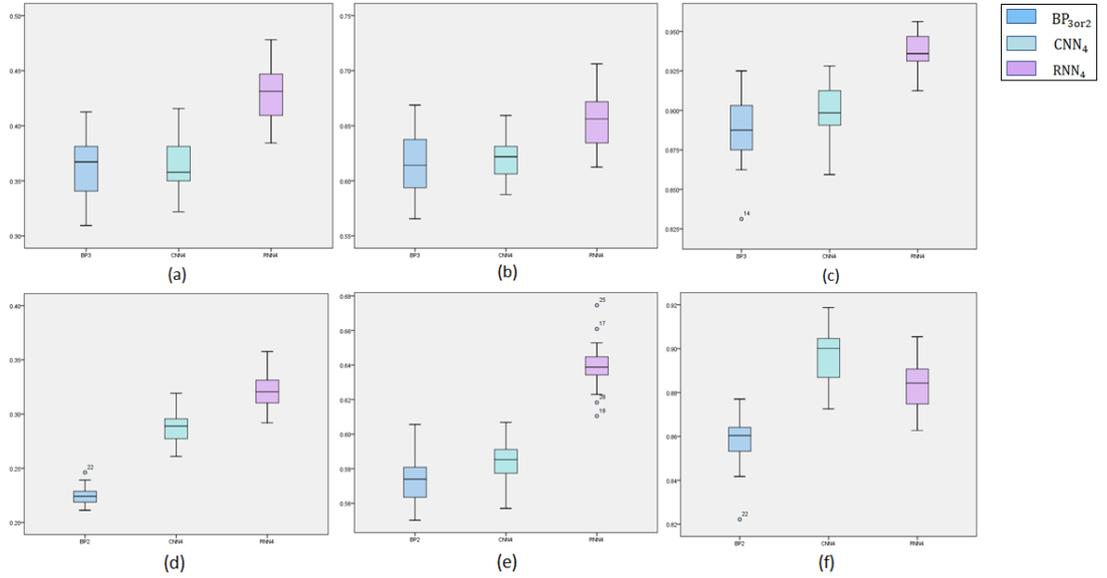


Fig. 9. Neural network box diagram

Table 21. Confusion matrix for wine data

Actual score	Prediction score(red)			$Recall_{T=0.5}\%$	Prediction score(white)			$Recall_{T=0.5}\%$
	5	6	7		5	6	7	
3	10	0	0	0.0	16	4	0	0.0
4	43	10	0	0.0	128	32	3	0.0
5	559	122	0	82.1	898	530	29	61.6
6	164	474	0	74.3	453	1537	208	69.9
7	7	192	0	0.0	44	467	369	41.9
8	0	18	0	0.0	0	66	109	0.0
9					1	2	2	0.0
$Precision_{T=0.5}\%$	71.4	58.1	0.0		58.3	58.3	51.3	

Table 22. Red Wine’s prediction model results

Test sets	MR		NN		SVM		RNN_4	
	Mean	Interval	Mean	Interval	Mean	Interval	Mean	Interval
$Accuracy\%_{T=0.25}$	31.2	[31.0,31.4]	31.1	[24.1,31.8]	43.2	[42.6 ,43.8]	43.0	[41.9, 44.0]
$Accuracy\%_{T=0.50}$	59.1	[59.0,59.2]	59.1	[58.8,59.4]	62.4	[62.0,62.8]	65.4	[64.5 , 66.4]
$Accuracy\%_{T=1.00}$	88.6	[88.5,88.7]	88.8	[88.6,89.0]	89.0	[88.8,89.2]	93.7	[93.3 , 94.1]

Table 23. White Wine’s prediction model results

Test sets	MR		NN		SVM		RNN_4	
	Mean	Interval	Mean	Interval	Mean	Interval	Mean	Interval
$Accuracy\%_{T=0.25}$	25.6	[31.0,31.4]	26.5	[26.2,26.8]	50.3	[49.2 , 51.4]	32.5	[32.0,33.0]
$Accuracy\%_{T=0.50}$	51.7	[51.6,51.8]	52.6	[52.3,52.9]	64.6	[64.2 , 65.0]	63.8	[63.4,64.3]
$Accuracy\%_{T=1.00}$	84.3	[84.2,84.4]	84.7	[84.6,84.8]	86.8	[86.4,87.0]	88.5	[88.2 , 88.9]

For Red wine data with relatively small sample size, the neural network model RNN_4 performed best when $T=0.5$ and 1.0 , and the average accuracy and lower limit of interval were slightly lower than SVM when $T=0.25$, as shown in Table 22. When $T=0.5$, compared with SVM, the average accuracy of the model RNN_4 is improved by 3%, and the accuracy range is between 64.5% and 66.4%, while the upper limit of SVM accuracy is less than 62.9%. When $T=1.0$, the average accuracy of the model RNN_4 is improved by 4.7% compared with SVM, and the lower limit of its interval is greater than 93%, while the upper limit of the interval of other comparison methods is less than 89.5%. Compared with MR, NN and SVM, the overall prediction accuracy of the model RNN_4 has been improved to a certain extent, and it is more suitable to solve the problem of Red wine quality evaluation.

For the relatively large amount of White wine data, the experimental results of the neural network model RNN_4 at $T=0.25$ are significantly better than MR and NN. Although it was inferior to SVM, the performance of the neural network model RNN_4 at $T=0.5$ was very close to that of SVM, and it was better at $T=1.0$, as shown in Table 23. When $T=0.5$, compared with SVM, the average precision value is slightly 0.8% smaller. At this time, the difference between the upper and lower limits of the interval is 0.8% and 0.7% respectively, almost having the same performance. When $T=1.0$, the average accuracy is 4.23% and 3.83 higher than MR and NN, which is 2% higher than SVM. Although the model RNN_4 does not completely show the optimal performance within all tolerances, it has approximate and slightly superior performance with SVM under tolerances of 0.5 and 1.0, respectively. Therefore, it is competitive to a certain extent when solving the problem of evaluating the quality of White wine.

5 Summary and Prospect

This paper designs a deep collaborative optimization framework, and uses this optimization framework to obtain a neural network prediction model that solves the problem of wine quality evaluation. The collaborative optimization framework consists of three stages: feature extraction, neural network structure optimization, and neural network parameter optimization.

The feature extraction stage refines the data features to make the data convenient for neural network training. The neural network structure optimization stage follows the idea of multi-objective optimization, and finds the optimal structure for each deep neural network. In this part of the optimization of the number of nodes in the middle layer of the neural network, an interval coding and interval search method is designed to improve the efficiency of optimization. The neural network parameter optimization stage is to further fine-tune the network parameters to achieve the best network performance. The collaborative optimization framework is suitable for various types of neural networks, but it is necessary to determine in advance the type of neural network suitable for solving such problems as a candidate set according to the characteristics of the problems to be solved and the data, through investigation and simple experiments. In this paper, BP, CNN and RNN three types of neural network were selected through simple experiments, and the final experiment proved that the neural network model under the RNN type showed better performance. By comparing this model with the method in the paper [5], it was found that this network model had certain advantages in Red wine data. It is competitive on White Wine data set to a certain extent, which also proves the effectiveness and feasibility of the deep collaborative optimization framework.

According to the analysis of this experiment, for the wine data set, when the proportion of a certain label sample is very small, it is difficult for the neural network to obtain more characteristic information of the label sample during the training. Therefore, the prediction effect of the trained neural network model on this part of the label sample needs to be improved. In this regard, the data expansion function can be added in the feature extraction stage, and such data with a small proportion of tags can be expanded in advance, and then put into the formal training of the network. In addition, the collaborative optimization framework can increase the information exchange between different types of neural networks to speed up the convergence of the network.

References

1. Dong Z, Atkison T, Chen B: Wineinformatics: Using the Full Power of the Computational Wine Wheel to Understand 21st Century Bordeaux Wines from the Reviews. 2021.
2. Zeqing Dong, Xiaowan Guo, Syamala Rajana, Bernard Chen: Understanding 21st Century Bordeaux Wines from Wine Reviews Using Nave Bayes Classifier[J]. Beverages, 2020, 6(1).

3. James Palmer,Victor S. Sheng,Travis Atkison,Bernard Chen: Classification on grade, price, and region with multi-label and multi-target methods in wineinformatics[J]. *Big Data Mining and Analytics*,2020,3(1).
4. Bernard Chen,Valentin Velchev,James Palmer,Travis Atkison: Wineinformatics: A Quantitative Analysis of Wine Reviewers[J]. *Fermentation*,2018,4(4).
5. Paulo Cortez,Antnio Cerdeira,Fernando Almeida,Telmo Matos,Jos Reis: Modeling wine preferences by data mining from physicochemical properties[J]. *Decision Support Systems*,2009,47(4).
6. Ebeler S E : Linking Flavor Chemistry to Sensory Analysis of Wine[M]. Springer US, 1999.
7. Yeim Er, Atasoy A: The Classification of White Wine and Red Wine According to Their Physicochemical Qualities[J]. 2016.
8. Ndirangu D, Mwangi W, Nderu L: An ensemble outlier detection method for multiclass classification problem in data mining[C]//Proceedings of the 2018 International Conference on Data Science and Information Technology. 2018: 38-42.
9. Soo Chung,Tu Park,Soo Park,Joon Kim,Seongmin Park,Daesik Son,Young Bae,Seong Cho: Colorimetric Sensor Array for White Wine Tasting[J]. *Sensors*,2015,15(8).
10. Wei Fan,Zhi Pan: Application of Neural Network in Wine Grape Quality Evaluation[P]. *Proceedings of the 2013 International Conference on Advanced Computer Science and Electronics Information (ICACSEI 2013)*,2013.
11. Ding S , Su C , Yu J: An optimizing BP neural network algorithm based on genetic algorithm[J]. *Artificial Intelligence Review*, 2011, 36(2):153-162.
12. Aljarah I , Faris H , Mirjalili S: Optimizing connection weights in neural networks using the whale optimization algorithm[J]. *Soft Computing*, 2018, 22(1):1-15.
13. Schmidhuber, Jrgen: Deep Learning in Neural Networks: An Overview[J]. *Neural Netw*, 2015, 61:85-117.
14. Karaboga D , Akay B , Ozturk C: Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks[C]// International Conference on Modeling Decisions for Artificial Intelligence. Springer Berlin Heidelberg, 2007.
15. Hou P, Zhao B, Jolliet O, et al: Rapid Prediction of Chemical Ecotoxicity Through Genetic Algorithm Optimized Neural Network Models[J]. *ACS Sustainable Chemistry & Engineering*, 2020, 8(32): 12168-12176.
16. Afrakhteh S , Mosavi M R , Khishe M , et al: Accurate Classification of EEG Signals Using Neural Networks Trained by Hybrid Population-Physic-Based Algorithm[J]. *International Journal of Automation and Computing*, 2020.
17. Chatterjee S , Sarkar S , Hore S , et al: Particle swarm optimization trained neural network for structural failure prediction of multistoried RC buildings[J]. *Neural Computing & Applications*, 2017, 28(8):2005-2016.
18. Hamdi R E , Njah M , Chtourou M: Multilayer perceptron training using an evolutionary algorithm[J]. *International Journal of Modelling Identification & Control*, 2008, 5(4):305-312.
19. Nanda S J, Panda G: A survey on nature inspired metaheuristic algorithms for partitional clustering[J]. *Swarm and Evolutionary Computation*, 2014, 16:1-18.
20. Tao Y, Cloutie R S: Voxelwise detection of cerebral microbleed in CADASIL patients by genetic algorithm and back propagation neural network[C]//2018 3rd International Conference on Communications, Information Management and Network Security (CIMNS 2018). Atlantis Press, 2018: 101-105.
21. Rostam M G , Sadatinejad S J , Malekian A: Precipitation forecasting by large-scale climate indices and machine learning techniques[J]. *Journal of Arid Land*, 2020, 12(5):854-864.

22. Cristin R , Raj V C , St.Peter's Institute of Higher Education and Research, et al: Consistency features and fuzzy-based segmentation for shadow and reflection detection in digital image forgery[J]. *Science China(Information Sciences)*, 2017.
23. [23] Wu M, Lin J, Shi S, et al. Hybrid Optimization-Based GRU Neural Network for Software Reliability Prediction[C]//International Conference of Pioneering Computer Scientists, Engineers and Educators. Springer, Singapore, 2020: 369-383.
24. Admasie S , Bukhari S , Gush T , et al: Intelligent Islanding Detection of Multi-distributed Generation Using Artificial Neural Network Based on Intrinsic Mode Function Feature[J]. *Journal of Modern Power Systems and Clean Energy*, 2020, 8(3):511-520.
25. Feng Y , Xu X: A short-term load forecasting model of natural gas based on optimized genetic algorithm and improved BP neural network[J]. *Applied Energy*, 2014, 134(dec.1):102-113.
26. Subirats J L , Franco L , Jerez J M: C-Mantec: A novel constructive neural network algorithm incorporating competition between neurons[J]. Elsevier Science Ltd. 2012.
27. Seiffert U: Multiple layer perceptron training using genetic algorithms[C]//ESANN. 2001: 159-164.
28. Yadav A , Chatterjee S , Equeenuddin S M: Suspended sediment yield modeling in Mahanadi River, India by multi-objective optimization hybridizing artificial intelligence algorithms[J]. *International Journal of Sediment Research*, 2020.
29. Belgiu M, Drgu L: Random forest in remote sensing: A review of applications and future directions[J]. *ISPRS journal of photogrammetry and remote sensing*, 2016, 114: 24-31.
30. Benesty J, Chen J, Huang Y, et al: Pearson correlation coefficient[M]//Noise reduction in speech processing. Springer, Berlin, Heidelberg, 2009: 1-4.
31. Andrew A M: Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, by John H. Holland[J]. *Robotica*, 1993, 11(5):489-489.
32. Lee C Y , Yao X: Evolutionary programming using mutations based on the Levy probability distribution[J]. *IEEE Transactions on Evolutionary Computation*, 2004, 8(1):1-13.
33. Rigelsford J: Intelligent optimisation techniques : genetic algorithms, tabu search, simulated annealing and neural networks[J]. *Industrial Robot*, 2000, 27(5).
34. Yang X S: Nature-Inspired Metaheuristic Algorithms. Luniver Press, 2010.
35. Yang X S: Multiobjective firefly algorithm for continuous optimization[J]. *Engineering with Computers*, 2013, 29(2):175-184.
36. Whittington J C R, Bogacz R: Theories of error back-propagation in the brain[J]. *Trends in cognitive sciences*, 2019, 23(3): 235-250.
37. Lukoevius M, Jaeger H: Reservoir computing approaches to recurrent neural network training[J]. *Computer Science Review*, 2009, 3(3): 127-149.
38. Albawi S, Mohammed T A, Al-Zawi S: Understanding of a convolutional neural network[C]//2017 International Conference on Engineering and Technology (ICET). Ieee, 2017: 1-6.

Short Papers

Application Of The Population Dynamics Membrane System In COVID-19 Propagation Model

Yourui Huang¹, Qi Song² *, Hongping Song², Shanyong Xu², and Tao Han²

¹ Anhui Science and Technology University, Chuzhou 233100, China

² School of Electrical and Information Engineering, Anhui University of Science and Technology, Huainan 232000 China

Abstract. In this paper, we collected information on more than 900 samples of new corona pneumonia from January 23 to February 16 in Guangdong, China, based on the membrane system of the population dynamics of the original infectious disease. The age of the diagnosed population was classified, and the transmission characteristics of different age groups were coupled, which was related to the family structure and multi generation living space in Guangdong Province, and combine the characteristics of population movement after sealing city to establish a computational model of the population dynamics membrane systems (PDP system) to meet the transmission characteristics of new crown pneumonia in Guangdong Province. Finally, the effectiveness of the model is verified by comparing with real data through the MeCoSim simulation.

Keywords: COVID-19; PDP system; MeCoSim

1 Introduction

COVID-19 is rampaging around the world. Up to now, nearly 200 million people have been diagnosed, and as many as 4 million people have died [1]. Because COVID-19 is an RNA virus [2], its mutability is much higher than that of common viruses, resulting in a variety of variants in the process of virus transmission, which is difficult to cure effectively. Therefore, it has important scientific value for COVID-19 to establish an early propagation model [3][4].

Guangdong Province is the largest province in China. It is located in the coastal area and has frequent import and export [5]. 90% of the entry population in Guangdong Province is easily affected by overseas imported cases and has a high risk of infection. As of July 2021, 2751 cases of COVID-19 were diagnosed in Guangdong Province, and 124 cases were confirmed. On this basis, studying and understanding the epidemiological characteristics of COVID-19 and establishing the relevant virus propagation model are the key to effectively prevent COVID-19 from breaking out again.

The population dynamics membrane systems are a new formal bionic modeling framework, which has been successfully applied to the real ecosystem population dynamics model. Compared with using abstract function to model the biological population, the membrane system of population dynamics is based on the probabilistic membrane system, and the rewriting rules on multiple sets are used to represent the number

* This work was supported by the National Natural Science Foundation of China under Grant 61772033

of discrete components of the system. The classical modeling framework mainly depends on relevant data, if the data is not accurate or missing, it can not be accurately predicted [6]. The population dynamics membrane system mainly depends on sociological investigation, and the requirements for relevant data are relatively low. Even if the current data can not be obtained, it can still rely on speculation and population sociological relationship to obtain relatively accurate prediction. In literature [7], the system is mainly used in the framework of large number and periodic repetitive dynamics model of population. In literature [8], the calculation model of captive giant panda ecosystem was successfully established by using population dynamic probabilistic membrane system. Literature used PDP system to establish SIR model and successfully established Japanese H1N1 influenza a transmission model.[9]

2 Sociological Survey of New Crown Epidemic in Guangdong Province

The detailed information of 994 patients in Guangdong Province from Jan . 9 to Feb. 16, 2020 is analyzed. [10] Fever is the main clinical symptom accounting for 73%. A few of the clinical symptoms are cough, dizziness, muscle ache, weakness, etc. Through age segmentation of new cases in Guangdong Province every day, we found that 80% of new cases are 20-64 years old, 15% are 65 years old and above, and only 5% are 0-19 years old.

The outbreak of COVID-19 in Guangdong Province shows obvious age differences, which is an important factor in building SIR model of PDP system.

Age	Infection Rate	Data sources
0-5years old	0.0164	MCMC
6-19years old	0.0164	MCMC
20-64 years old	0.0869	MCMC
Over 65 years old	0.0358	MCMC

Table1 Infection rate of different age groups in Guangdong Province

The population distribution of Guangdong Province in different age groups from 2019 to 2020 can be estimated by referring to the data of China's 2010 population census and the relevant population data provided in literature

Age	X1(0-5)	X2(6-19)	X3(20-64)	X4(Over 65)
Population	7153800	16207482	78760477	10664042

Table2 Population distribution of different age groups in Guangdong Province from 2019 to 2020 (unit: person)

Due to the lack of understanding of the characteristics of the virus at the beginning of the outbreak, the epidemic situation was basically in a state of free transmission from January 9 to January 22, and due to the lack of effective detection means at the initial stage, the data of confirmed cases were not accurate. In this study, the data from January 21 to 22 were used as the initial reference value. An average of 23 new patients per day.

3 The infectious disease PDP system

The infectious disease PDP system is a general population dynamic Sir calculation model based on the framework of probabilistic membrane P system [9]. S stands for susceptible population; I represent the infected population; R stands for people who have recovered or died. The model is inspired by the virus transmission channels generated by the interaction between populations, and simulates the transmission dynamics of the epidemic situation through the probability membrane P-system to simulate the interaction behavior of different populations and the different infection rates of other individuals.

Each PDP system is composed of seven rule modules, which will be executed only when the first initial infected person appears. Random infected persons will be selected in the crowd during initialization. The other six rule modules are scenario evolution in the same day. The details are shown in the figure below:

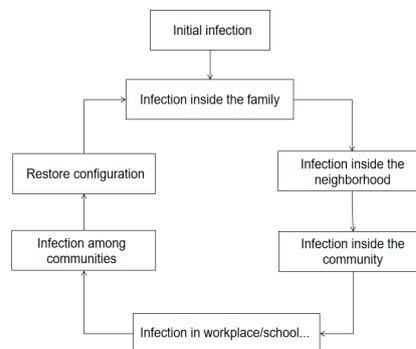


Fig. 1. SIR Epidemic model

The epidemic model consists of three physically separated communities. Each community consists of four blocks, where there are the basic facilities of daily life: schools, workplaces, shops, etc. Individuals in a community are organized into families (families may have different structures or number of members). In addition, seven groups were considered according to their age: kindergarten, primary school, secondary school, high school and two groups of adults (19-64 years old, over 64 years old). A susceptible person may be infected at home, at work, or in his spare time (nearby or traveling).

The Plingua detailed description of the infectious disease PDP system is shown in reference [11][12], which will not be repeated here.

4 Model assumptions and construction

Before January 23, 2020, due to the Spring Festival, population mobility is higher than usual, and disorder is high. Medical institutions do not have a deep understanding of the

disease and do not take effective monitoring measures. We did not model the previous data. Based on sociological survey, we make the following assumptions for PDP model

1. We assume that the population of the model is a fixed constant and the population of each age group remains unchanged.
2. The COVID-19 can be transmitted through air, and families will also spread around their neighbors at the same time.
3. the sick people will always stay at home, and other family members will choose to go out. We assume that the proportion of asymptomatic patients is 10% [13], and they have the same infectious ability as the patients [14][15], but they can always recover after one week.
4. Suppose that each patient has a latent period of 7 days and a diagnosis period of 3 days after the onset of the disease, that is, the effective transmission time is 10 days.

The number of communities and families is the parameters that have been put into operation, and the parameters are determined according to the COVID-19 sociological survey in Guangdong province. According to the population proportion of different age groups in Table 2, the proportion of different age groups is about 7:16:78:10.

Each community is composed of 20 identical modules, and each module is composed of N families. According to literature, the average number of each family is 1.73. The empirical value of n is 5.

In the final stage, 14 new cases were added on January 21, and 31 new cases were added on January 22. Due to the lag of disease detection, we took the average of 22 people in these two days as the initial value of simulation from January 21, and distributed them to three communities as the initial infected people in the community.

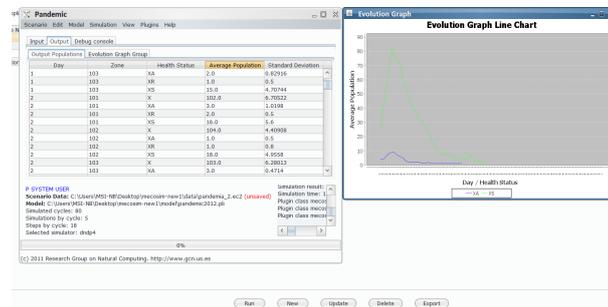


Fig. 2. The simulation results through the mecosim

5 Results analysis and comparison

When testing the daily new value predicted by the model, we started to predict from the closure day on January 23. From the figure below, we can find that the trend of the predicted value of daily new patients in Fig.3 is basically the same as that of the measured value, both of which reach the peak within 5-9 days after the closure, and

then gradually decline, which shows that the epidemic prevention strategy of restricting population flow is effective, It can also be inferred that the common latent period of epidemic is 5-9 days. A novel coronavirus pneumonia test is not detected in real time. The peak value of the peak is reached on the sixth day. The peak value of the measured value reaches eighth days. Considering the detection of the new crown pneumonia is not real-time, even if the disease is diagnosed, there is a delay in the diagnosis. Based on this, we think that the error of the number of days of peak arrival is acceptable.

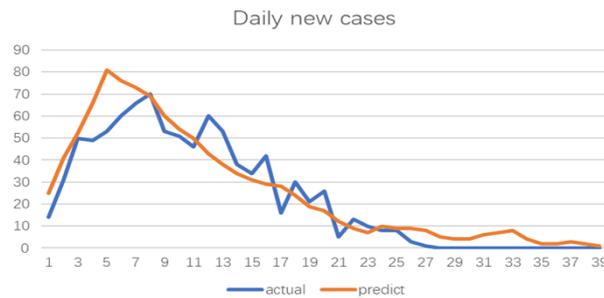


Fig. 3. Daily new cases

In Fig.4, we remove the asymptomatic infection, and only compare the number and trend of the predicted symptomatic infection with the measured symptomatic infection.

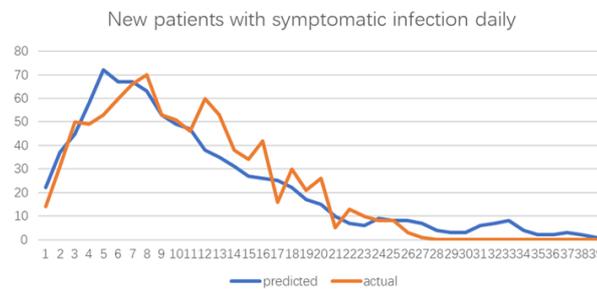


Fig. 4. Daily new cases without asymptomatic patient

We can find that the trend of predicted daily new value-added is not only the same as that of measured daily new value-added, but also the cumulative total value is basically the same. The predicted value of the table only includes symptomatic infection, so it is lower than the measured value. After the analysis, it is believed that although the actual survey data shows that the condition has been cleared on February 17, due to the existence of asymptomatic infection and the inability to accurately detect at that time, the condition has not really cleared. This hypothetical prediction is also in line with the

actual situation of sporadic confirmed cases in some cities after the disease is cleared in the epidemic prevention and control stage.

6 Summary and Prospect

In this study, we establish a new transmission model of new crown in Guangdong Province based on the membrane system of infectious disease population dynamics. With the good biomimetic performance of membrane computing, accurate epidemic prediction is achieved, and the specific application scope of membrane computing is expanded.

References

1. Klavin skis LS; Liu MA, et al. A timely up dateofglobal COVID-19 vaccine development. *Emerg Microbes Infect*, 2020, 9.
2. SHI Qing-feng; GAO Xiao-dong; HU Bi-jie. Research progress on characteristics, epidemiology and control measure of SARS-CoV-2DeltaVOC.
3. Arpit Jain; Abhinav Sharma, et al. "Use of AI, Robotics, and Modern Tools to Fight Covid-19," in *Use of AI, Robotics, and Modern Tools to Fight Covid-19*, River Publishers, 2021, pp. ii-xxx.
4. R. Y. Wang; T. Q. Guo; L. G. Li, et al. "Predictions of COVID-19 Infection Severity Based on Co-associations between the SNPs of Co-morbid Diseases and COVID-19 through Machine Learning of Genetic Data," 2020 IEEE 8th International Conference on Computer Science and Network Technology (ICCSNT), 2020, pp. 92-96.
5. XIE Yi-he. An Analysis of the Impact of Guangdong's Foreign Trade Import & Export on Fiscal Revenue.
6. Gh. P aun. *Membrane Computing: An Introduction*(Springer-Verlag, Berlin, 2002)
7. TIAN Hao; ZHANG Gexiang; QI Dunwu. Population model of giant panda ecosystem based on population dynamics P system.
8. M.A. Colomer; A. Margalida, et al. A bio-inspired computing model as a new tool for modeling ecosystems: the avian scavengers as a case study. *Ecol. Model.* 222(1), 3347 (2011)
9. Cugat M ;Fernandez G Q ; Luis Felipe Macas Ramos, et al. *Membrane System-Based Models for Specifying Dynamical Population Systems*[M]. Springer International Publishing, 2014.
10. Wang Guo-Qiang; Zhang Shuo. Study of coupling the age-structured contact patterns to the COVID-19 pandemic transmission.
11. The P-Lingua website. <http://www.p-lingua.org>
12. MeCoSim website. <http://www.p-lingua.org/mecosim>
13. <https://baijiahao.baidu.com/s?id=1663574581113505722&wfr=spider&for=pc>
14. Special Expert Group for Control of The Epidemic of Novel Coronavirus Pneumonia of The Chinese Preventive Medicine Association. An update on the epidemiological characteristics of novel coronavirus pneumonia (COVID-19)[J]. *Chinese Journal of Viral Diseases*, 2020, 10(2): 86-92. (in Chinese)
15. WU Z Y. Contribution of asymptomatic and pre-symptomatic cases of COVID-19 in spreading virus and targeted control strategies[J]. *Chinese Journal of Epidemiology*, 2020, 41(6): 801-805(in Chinese)

Some Results on Parallel Contextual Hexagonal Array Insertion Deletion P System

S. James Immanuel¹[0000-0003-0653-4882], S. Jayasankar²[0000-0001-9896-0779],
D.G.Thomas³[0000-0001-6327-8446], M.Gayathri Lakshmi, and Meenakshi
Paramasivan⁴[0000-0002-1509-6557]

¹ Department of Mathematics, Sri Sairam Institute of Technology, Chennai - 600044,
India

`james_imch@yahoo.co.in`

² Department of Mathematics, Ramakrishna Mission Vivekananda College, Chennai
600004, India

`ksjayjay@gmail.com`

³ Department of Science and Humanities (Mathematics Division), Saveetha School of
Engineering, Chennai - 602 105, India

`dgthomasmcc@yahoo.com`

⁴ Research Scholar, Saveetha School of Engineering [SIMATS]; Department of
Mathematics, Saveetha Engineering College, Chennai 602105, India

`kirthana306@gmail.com`

⁵ FB IV - Informatikwissenschaften, Universität Trier, 54286 Trier, Germany

`meena_maths@yahoo.com`

Abstract. Hexagonal picture languages are seen as an extension of two-dimensional rectangular picture languages. James et al (2016) have introduced a P system called parallel contextual hexagonal array P systems (PCHAPS). Jayasankar et al (2017) have studied a P system called parallel contextual hexagonal array insertion-deletion P system (PCHADPS) based on insertion-deletion operations. In this paper, we prove that the family of hexagonal picture languages generated by the P system PCHAPS is a sub-family of the family of hexagonal picture languages generated by PCHADPS.

Keywords: P system, Hexagonal array, Contextual grammars, Insertion and deletion operations

1 Introduction

Hexagonal pictures occur in several application areas especially in picture processing and image analysis [6]. Hexagonal kolam array grammars for generating hexagonal arrays and hexagonal patterns on triangular grids which can be treated as two-dimensional representation of three-dimensional blocks was constructed by Siromoney et al. [10].

Contextual grammars were introduced by S.Marcus[5] in 1969 as another model to describe natural languages. A contextual grammar produces a language by starting from a given finite set of strings and adding, iteratively, pairs

of strings(called as contexts), associated to sets of words(called selectors) to the string already obtained. Many variants of contextual grammars have been considered in the literature and investigated from a mathematical point of view. Two special cases of contextual grammars, called internal and external are very natural and have been extensively investigated. An external contextual grammar generates a language starting from a finite set of strings and iteratively adjoining to its contexts. In internal contextual grammars, the context are adjoined inside the current string. For further details, we refer to [8]

In [11], D.G.Thomas et al developed a new method of generating hexagonal arrays based on an extension of contextual grammars called parallel contextual hexagonal array grammars. Their systems yield languages of hexagons using parallel rewriting relations. They make use of 'window movement' on arrow heads to decide whether the languages are generated by array contexts of choice mappings by the applications of array contextual operations parallelly.

The area of membrane computing was initiated by Paun [7] introducing a new computability model, now called as P system. For more details of membrane computing we refer to [9]

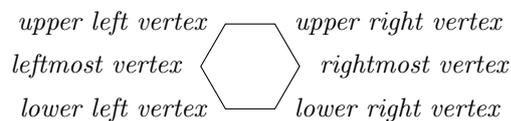
James et al [3] have introduced a P systems called external and internal parallel contextual hexagonal P systems to generate pictures using X, Y, and Z directional parallel contextual hexagonal array rules and studies some basic properties of these P systems and proved comparison results in terms of their generative powers. Jayasankar et al [4] developed a new P system model called parallel contextual hexagonal array insertion-deletion P system (PCHADPS) based on X, Y and Z directional contextual rules of parallel contextual hexagonal array grammar and makes use of parallel contextual double window movement along the arrow heads for generating pictures. They studied closure properties of PCHADPS and proved PCHADPS has more generative power compared to some well-known families of hexagonal languages like HLOC and HREC [1].

In this paper, we compare the generative powers of P system models PCHAPS and PCHADPS and prove that the family of hexagonal picture languages generated by PCHADPS properly contains the family of hexagonal picture languages generated by the P system PCHAPS.

2 Preliminaries

In this section we recall some notions related to parallel contextual hexagonal array P systems. We can refer to [11,1] for further details.

Definition 1. *We consider hexagons of the following type:*



Let Σ be a finite alphabet of symbols. A hexagonal picture p over Σ is a hexagonal array of symbols over Σ . For example, a hexagonal picture over the

alphabet $\{a, b, c\}$ is: $p = \begin{matrix} & a & b & \\ & \diagdown & \diagup & \\ a & c & b & \\ & \diagup & \diagdown & \\ & a & b & \end{matrix}$. The set of all hexagonal arrays over Σ is denoted by Σ^{**H} . A hexagonal picture language L over Σ is a subset of Σ^{**H} . With respect to a triad $x \swarrow \searrow y \nearrow z$ of triangular axes x, y, z , the coordinates of each element of a hexagonal picture can be fixed. For $l, m, n \geq 1$, $\Sigma^{l,m,n}$ denotes the set of all hexagonal pictures of size (l, m, n) with l, m, n stand for the length of the sides of the hexagon in the x, y, z directions respectively.

For the definitions of trapezium, parallelogram array types and different types of arrow-heads and for operations like concatenation and contextual operations, we refer to [3], For definitions regarding Parallel internal contextual hexagonal array P systems and Parallel external contextual array P systems, we refer to [3]. For the notions of Parallel Contextual hexagonal array insertion-deletion P system, we refer to [4].

3 Comparison Results

In this section, we compare the generative powers of two P systems $PCHAPS$ and $PCHAIDPS$ and prove that the family of hexagonal picture languages generated by $PCHAIDPS$ properly includes the family of hexagonal picture languages generated by the P system $PCHAPS$.

Theorem 1. $\mathcal{L}(PICHAP) \subseteq \mathcal{L}(PCHAIDPS)$.

Proof. For every internal parallel contextual hexagonal array P system with h membranes generating a language L , we construct a corresponding parallel contextual hexagonal array insertion P system generating the same language L .

Consider $\prod_{int} \in PICHAP_h$ generating a language L as

$$\prod_{int} = \left(V, T, \mu, XY, YZ, ZX, M_1, M_2, \dots, M_h, (R_1, \phi_1), (R_2, \phi_2), \dots, (R_h, \phi_h), i_0 \right).$$

We construct a $PCHAIP_h$,

$$\prod = \left(V', T, \mu, (ZX)', (ZY)', (XY)', (XZ)', (YX)', (YZ)', (M_1, I_1, D_1), \dots, (M_{h+1}, I_{h+1}, D_{h+1}), \Phi_X, \Phi_Y, \Phi_Z, i_0 \right), \text{ where } V' = V \cup \{Y\} \text{ and}$$

$$\begin{aligned} \forall \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$_{xy} \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in XY', \exists \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \in (ZX)', \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in (ZY)'. \\ \forall \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \$_{yx} \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in YX', \exists \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \in (ZY)', \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in (ZX)'. \\ \forall \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \$_{xx} \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in YX', \exists \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \in (ZX)', \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix} \in (ZX)'. \\ \forall \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \$_{yy} \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \in YY', \exists \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \in (ZY)', \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \in (ZY)'. \\ \forall \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$_{yz} \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in YZ', \exists \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \in (XY)', \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in (XZ)'. \end{aligned}$$

$$\begin{aligned}
 &\forall [u'_i \ u'_{i+1}] \$_{zy} \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in ZY', \exists [u'_i \ u'_{i+1}] \in (XZ)', \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in (XY)'. \\
 &\forall \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \$_{yy} \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix} \in YX', \exists \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \in (XY)', \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix} \in (XY)'. \\
 &\forall [u_k \ u_{k+1}] \$_{zz} \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \in YY', \exists [u_k \ u_{k+1}] \in (XZ)', [v_k \ v_{k+1}] \in (XZ)'. \\
 &\forall [u_i \ u_{i+1}] \$_{zx} \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in ZX', \exists [u_i \ u_{i+1}] \in (YZ)', \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in (YX)'. \\
 &\forall \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \$_{xz} [v'_i \ v'_{i+1}] \in XZ', \exists \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \in (YX)', [v'_i \ v'_{i+1}] \in (YZ)'. \\
 &\forall \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \$_{zz} [v_j \ v_{j+1}] \in ZZ', \exists \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \in (YZ)', [v_j \ v_{j+1}] \in (YZ)'. \\
 &\forall \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \$_{xx} \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \in XX', \exists \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \in (YX)', \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \in (YX)'.
 \end{aligned}$$

If $R_r = \left\{ \left(\left(\varphi_{xy}(xyTr_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$_{xy} \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix}, \right. \right. \right.$
 $\varphi_{yx}(yxTr_i) = \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \$_{yx} \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix}, \varphi'_{xx}(xxP_zA_j) = \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \$_{xx} \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix},$
 $\left. \left. \left. \varphi'_{yy}(yyP_zA_k) = \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \$_{yy} \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \right\}, \alpha \right\}$ $1 \leq i \leq m-1$ if $l > m$ or
 $1 \leq i \leq l-1$ if $m \geq l$, $j = 1, 2, \dots, l-m$ if $l > m$ and $k = 1, 2, \dots, m-l$ if
 $m > l$, $\alpha \in \{here, out, in_t\}$ then,

- $$\begin{aligned}
 I_r = &\left\{ \left(\left(\phi_{zx}^I(E_i, A_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \otimes \begin{bmatrix} Y \\ Y \end{bmatrix}, \phi_{zy}^I(F_i, B_i) = \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \otimes \begin{bmatrix} Y \\ Y \end{bmatrix}, \right. \right. \\
 &\phi_{zx}^I(G_i, C_i) = \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \otimes \begin{bmatrix} Y \\ Y \end{bmatrix}, \\
 &\left. \left. \left. \phi_{zy}^I(H_i, D_i) = \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \otimes \begin{bmatrix} Y \\ Y \end{bmatrix} \right\}, here \right) \right\} \\
 &\cup \left\{ \left(\left(\phi_{zy}^I(A'_i, E'_i) = \begin{bmatrix} Y \\ Y \end{bmatrix} \otimes \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix}, \phi_{zx}^I(B'_i, F'_i) = \begin{bmatrix} Y \\ Y \end{bmatrix} \otimes \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix}, \right. \right. \\
 &\left. \left. \left. \phi_{zx}^I(C_i, G'_i) = \begin{bmatrix} Y \\ Y \end{bmatrix} \otimes \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix}, \phi_{zy}^I(D_i, H'_i) = \begin{bmatrix} Y \\ Y \end{bmatrix} \otimes \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \right\}, here \right) \right\}
 \end{aligned}$$

where $1 \leq i \leq m-1$ if $l > m$ or $1 \leq i \leq l-1$ if $m \geq l$, $j = 1, 2, \dots, l-m$ if
 $l > m$ and $k = 1, 2, \dots, m-l$ if $m > l$,

$$\begin{aligned}
 E_i &= \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \text{ with } e_1, e_2 \in T \cup \{\varepsilon\}, F_i = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \text{ with } f_1, f_2 \in T \cup \{\varepsilon\}, \\
 G_i &= \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \text{ with } g_1, g_2 \in T \cup \{\varepsilon\}, H_i = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \text{ with } h_1, h_2 \in T \cup \{\varepsilon\}, \\
 E'_i &= \begin{bmatrix} e'_1 \\ e'_2 \end{bmatrix} \text{ with } e'_1, e'_2 \in T \cup \{\varepsilon\}, F'_i = \begin{bmatrix} f'_1 \\ f'_2 \end{bmatrix} \text{ with } f'_1, f'_2 \in T \cup \{\varepsilon\}, \\
 G'_i &= \begin{bmatrix} g'_1 \\ g'_2 \end{bmatrix} \text{ with } g'_1, g'_2 \in T \cup \{\varepsilon\}, H'_i = \begin{bmatrix} h'_1 \\ h'_2 \end{bmatrix} \text{ with } h'_1, h'_2 \in T \cup \{\varepsilon\} \text{ and}
 \end{aligned}$$

A_i is the maximum possible zx -array context of $xyTr_i$, B_i is the maximum possible zy -array context of $yxTr_i$, A'_i is the maximum possible zy -array context of $xyTr_i$, B'_i is the maximum possible zx -array context of $yxTr_i$, $C_i = xxP_zA_j$ and $D_i = yyP_zA_k$.

$$\begin{aligned}
 2. D_r &= \left\{ \left(\left(\phi_{zx}^D \left(\begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}, A_i \right) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix}, \phi_{zy}^D \left(\begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix}, B_i \right) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix}, \right. \right. \\
 &\quad \left. \left. \phi_{zx}^D \left(\begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix}, C_i \right) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix}, \phi_{zy}^D \left(\begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix}, D_i \right) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix} \right\}, in_{h+1} \right\} \\
 3. D_{h+1} &= \left\{ \left(\left(\phi_{zy}^D \left(A'_i, \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \right) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix}, \phi_{zx}^D \left(B'_i, \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \right) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix}, \right. \right. \\
 &\quad \left. \left. \phi_{zx}^D \left(C_i, \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix} \right) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix}, \phi_{zy}^D \left(D_i, \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \right) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix} \right\}, \alpha \right\}
 \end{aligned}$$

If $R_r = \left\{ \left(\left(\varphi_{yz}(yzTr_i) = \begin{bmatrix} u_i & \\ & u_{i+1} \end{bmatrix} \$_{yz} [v_i \ v_{i+1}], \right. \right.$
 $\varphi_{zy}(zyTr_i) = \begin{bmatrix} u'_i & u'_{i+1} \end{bmatrix} \$_{zy} \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix}, \varphi''_{yy}(yyP_xA_j) = \begin{bmatrix} u_j & \\ & u_{j+1} \end{bmatrix} \$_{yy} \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix},$
 $\left. \left. \varphi'_{zz}(zzP_xA_k) = \begin{bmatrix} u_k & u_{k+1} \end{bmatrix} \$_{zz} [v_k \ v_{k+1}] \right\}, \alpha \right\}$ $1 \leq i \leq n-1$ if $m > n$ or
 $1 \leq i \leq m-1$ if $n \geq m$, $j = 1, 2, \dots, m-n$ if $m > n$ and $k = 1, 2, \dots, n-m$ if
 $n > m$. $\alpha \in \{here, out, in_t\}$ then,

$$\begin{aligned}
 1. I_r &= \left\{ \left(\left(\phi_{xy}^I(E_i, A_i) = \begin{bmatrix} u_i & \\ & u_{i+1} \end{bmatrix} \odot \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix}, \phi_{xz}^I(F_i, B_i) = [u'_i \ u'_{i+1}] \ominus \right. \right. \\
 &\quad \left. \left. \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix}, \phi_{xy}^I(G_i, C_i) = \begin{bmatrix} u_j & \\ & u_{j+1} \end{bmatrix} \odot \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix}, \right. \right. \\
 &\quad \left. \left. \phi_{xz}^I(H_i, D_i) = [u_k \ u_{k+1}] \ominus \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix} \right\}, here \right\} \cup \left\{ \left(\left(\phi_{xz}^I(A'_i, E'_i) = [Y \ Y] \ominus \right. \right. \right. \\
 &\quad \left. \left. \begin{bmatrix} v_i \ v_{i+1} \end{bmatrix}, \phi_{xy}^I(B'_i, F'_i) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix} \odot \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix}, \right. \right. \\
 &\quad \left. \left. \phi_{xz}^I(C_i, G'_i) = [Y \ Y] \ominus [v_j \ v_{j+1}], \phi_{xy}^I(D_i, H'_i) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix} \odot \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \right\}, here \right\}
 \end{aligned}$$

where $1 \leq i \leq n-1$ if $m > n$ or $1 \leq i \leq m-1$ if $n \geq m$, $j = 1, 2, \dots, m-n$
 if $m > n$ and $k = 1, 2, \dots, n-m$ if $n > m$,

$$E_i = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \text{ with } e_1, e_2 \in T \cup \{\varepsilon\}, F_i = [f_1 \ f_2] \text{ with } f_1, f_2 \in T \cup \{\varepsilon\},$$

$$G_i = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \text{ with } g_1, g_2 \in T \cup \{\varepsilon\}, H_i = [h_1 \ h_2] \text{ with } h_1, h_2 \in T \cup \{\varepsilon\}$$

$$E'_i = \begin{bmatrix} e'_1 \\ e'_2 \end{bmatrix} \text{ with } e'_1, e'_2 \in T \cup \{\varepsilon\}, F'_i = [f'_1 \ f'_2] \text{ with } f'_1, f'_2 \in T \cup \{\varepsilon\},$$

$$G'_i = \begin{bmatrix} g'_1 \\ g'_2 \end{bmatrix} \text{ with } g'_1, g'_2 \in T \cup \{\varepsilon\}, H'_i = [h'_1 \ h'_2] \text{ with } h'_1, h'_2 \in T \cup \{\varepsilon\} \text{ and}$$

A_i is the maximum possible xy -array context of $yzTr_i$, B_i is the maximum possible xz -array context of $zyTr_i$, $C_i = yyP_xA_j$ and $D_i = zzP_xA_k$, A'_i is the maximum possible xz -array context of $yzTr_i$, B'_i is the maximum possible xy -array context of $zyTr_i$.

$$\begin{aligned}
 2. D_r &= \left\{ \left(\left(\phi_{xy}^D \left(\begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}, A_i \right) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix}, \phi_{xz}^D \left(\begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix}, B_i \right) = [Y \ Y], \right. \right. \\
 &\quad \left. \left. \phi_{xy}^D \left(\begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix}, C_i \right) = \begin{bmatrix} Y & Y \\ Y & Y \end{bmatrix}, \phi_{xz}^D \left(\begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix}, D_i \right) = [Y \ Y] \right\}, in_{h+1} \right\}
 \end{aligned}$$

$$3. D_{h+1} = \left\{ \left(\left\{ \phi_{xz}^D \left(A'_i, [v_i \ v_{i+1}] \text{ bigg} \right) = [Y \ Y], \phi_{xy}^D \left(B'_i, \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \right) = \begin{bmatrix} Y \\ Y \end{bmatrix}, \right. \right. \\ \left. \left. \phi_{xz}^D \left(C_i, [v_j \ v_{j+1}] \right) = [Y \ Y], \phi_{xy}^D \left(D_i, \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \right) = \begin{bmatrix} Y \\ Y \end{bmatrix} \right\}, \alpha \right\}$$

If $R_r = \left\{ \left(\left\{ \varphi_{zx}(zxTr_i) = [u_i \ u_{i+1}] \$_{zx} \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix}, \right. \right. \\ \varphi_{xz}(xzTr_i) = \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \$_{xz} [v'_i \ v'_{i+1}], \varphi''_{zz}(zzPyA_j) = [u_j \ u_{j+1}] \$_{zz} [v_j \ v_{j+1}], \\ \left. \left. \varphi''_{xx}(xxPyA_k) = \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \$_{xx} \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \right\}, \alpha \right\} \ 1 \leq i \leq l-1 \text{ if } n > l \text{ or} \\ 1 \leq i \leq n-1 \text{ if } l \geq n, j = 1, 2, \dots, n-l \text{ if } n > l \text{ and } k = 1, 2, \dots, l-n \text{ if} \\ l > n. \alpha \in \{here, out, in_t\} \text{ then,}$

$$1. I_r = \left\{ \left(\left\{ \phi_{yz}^I(E_i, A_i) = [u_i \ u_{i+1}] \ominus [Y \ Y], \phi_{yx}^I(F_i, B_i) = \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \oslash \begin{bmatrix} Y \\ Y \end{bmatrix}, \right. \right. \\ \left. \left. \phi_{yz}^I(G_i, C_i) = [u_j \ u_{j+1}] \ominus [Y \ Y], \phi_{yx}^I(H_i, D_i) = \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \oslash \begin{bmatrix} Y \\ Y \end{bmatrix} \right\}, here \right) \\ \cup \left\{ \left(\left\{ \phi_{yx}^I(A'_i, E'_i) = \begin{bmatrix} Y \\ Y \end{bmatrix} \oslash \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix}, \phi_{yz}^I(B'_i, F'_i) = [Y \ Y] \ominus [v'_i \ v'_{i+1}], \right. \right. \\ \left. \left. \phi_{yx}^I(C_i, G'_i) = [Y \ Y] \ominus [v_j \ v_{j+1}], \phi_{yz}^I(D_i, H'_i) = \begin{bmatrix} Y \\ Y \end{bmatrix} \oslash \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \right\}, here \right) \right\}$$

where $1 \leq i \leq l-1$ if $n > l$ or $1 \leq i \leq n-1$ if $l \geq n, j = 1, 2, \dots, n-l$ if $n > l$ and $k = 1, 2, \dots, l-n$ if $l > n,$

$$E_i = [e_1 \ e_2] \text{ with } e_1, e_2 \in T \cup \{\varepsilon\}, F_i = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \text{ with } f_1, f_2 \in T \cup \{\varepsilon\},$$

$$G_i = [g_1 \ g_2] \text{ with } g_1, g_2 \in T \cup \{\varepsilon\}, H_i = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \text{ with } h_1, h_2 \in T \cup \{\varepsilon\},$$

$$E'_i = \begin{bmatrix} e'_1 \\ e'_2 \end{bmatrix} \text{ with } e'_1, e'_2 \in T \cup \{\varepsilon\}, F'_i = [f'_1 \ f'_2] \text{ with } f'_1, f'_2 \in T \cup \{\varepsilon\},$$

$$G'_i = [g'_1 \ g'_2] \text{ with } g'_1, g'_2 \in T \cup \{\varepsilon\}, H'_i = \begin{bmatrix} h'_1 \\ h'_2 \end{bmatrix} \text{ with } h'_1, h'_2 \in T \cup \{\varepsilon\} \text{ and}$$

A_i is the maximum possible yz -array context of $zxTr_i, B_i$ is the maximum possible yx -array context of $xzTr_i, C_i = zzPyA_j$ and $D_i = xxPyA_k, A'_i$ is the maximum possible yx -array context of $zxTr_i, B'_i$ is the maximum possible yz -array context of $xzTr_i.$

$$2. D_r = \left\{ \left(\left\{ \phi_{yz}^D \left([u_i \ u_{i+1}], A_i \right) = [Y \ Y], \phi_{yx}^D \left(\begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix}, B_i \right) = \begin{bmatrix} Y \\ Y \end{bmatrix}, \right. \right. \\ \left. \left. \phi_{yz}^D \left([u_j \ u_{j+1}], C_i \right) = [Y \ Y], \phi_{yx}^D \left(\begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix}, D_i \right) = \begin{bmatrix} Y \\ Y \end{bmatrix} \right\}, in_{h+1} \right) \right\}$$

$$3. D_{h+1} = \left\{ \left(\left\{ \phi_{yx}^D \left(A'_i, \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \right) = \begin{bmatrix} Y \\ Y \end{bmatrix}, \phi_{yz}^D \left(B'_i, [v'_i \ v'_{i+1}] \right) = [Y \ Y], \right. \right. \\ \left. \left. \phi_{yx}^D \left(C_i, [v_j \ v_{j+1}] \right) = [Y \ Y], \phi_{yz}^D \left(D_i, \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \right) = \begin{bmatrix} Y \\ Y \end{bmatrix} \right\}, \alpha \right) \right\}.$$

Theorem 2. $\mathcal{L}(PICHAP) \subseteq \mathcal{L}(PCHAIPS).$

Proof. For every internal parallel contextual hexagonal array P system with h membranes generating a language L , we construct a corresponding parallel contextual hexagonal array insertion P system generating the same language L .

Consider $\prod_{int} \in \text{PICHAIP}_h$ generating a language L as

$$\prod_{int} = \left(V, T, \mu, XY, YZ, ZX, M_1, M_2, \dots, M_h, (R_1, \phi_1), \right. \\ \left. (R_2, \phi_2), \dots, (R_h, \phi_h), i_0 \right).$$

We construct a PCHAIP_h ,

$$\prod = \left(V, T, \mu, (ZX)', (ZY)', (XY)', (XZ)', (YX)', (YZ)', (M_1, I_1), \dots, (M_g, I_g), \right. \\ \left. \Phi_X, \Phi_Y, \Phi_Z, i_0 \right), \text{ where } g = 2h \text{ if } R_1 \neq \emptyset, g = 2h - 1 \text{ if } R_1 = \emptyset. \text{ In this proof we consider } R_1 \neq \emptyset.$$

$$\begin{aligned} \forall \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$_{xy} \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in XY', \exists \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \in (ZX)', \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in (ZY)'. \\ \forall \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \$_{yx} \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in YX', \exists \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \in (ZY)', \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in (ZX)'. \\ \forall \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \$_{xx} \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in YX', \exists \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \in (ZX)', \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix} \in (ZX)'. \\ \forall \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \$_{yy} \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \in YY', \exists \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \in (ZY)', \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \in (ZY)'. \\ \forall \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$_{yz} \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in YZ', \exists \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \in (XY)', \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in (XZ)'. \\ \forall \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \$_{zy} \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in ZY', \exists \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \in (XZ)', \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in (XY)'. \\ \forall \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \$_{yy} \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix} \in YX', \exists \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \in (XY)', \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix} \in (XY)'. \\ \forall \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \$_{zz} \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \in YY', \exists \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \in (XZ)', \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \in (XZ)'. \\ \forall \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$_{zx} \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in ZX', \exists \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \in (YZ)', \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \in (YX)'. \\ \forall \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \$_{xz} \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in XZ', \exists \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \in (YX)', \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix} \in (YZ)'. \\ \forall \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \$_{zz} \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix} \in ZZ', \exists \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \in (YZ)', \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix} \in (YZ)'. \\ \forall \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \$_{xx} \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \in XX', \exists \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \in (YX)', \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \in (YX)'. \end{aligned}$$

Case(i): If $R_r = \left\{ \left(\left\{ \varphi_{xy}(xyTr_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$_{xy} \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix}, \right. \right. \right.$
 $\varphi_{yx}(yxTr_i) = \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \$_{yx} \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix}, \varphi'_{xx}(xxP_zA_j) = \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \$_{xx} \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix},$
 $\left. \left. \left. \varphi'_{yy}(yyP_zA_k) = \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \$_{yy} \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \right\}, \alpha \right\} 1 \leq i \leq m - 1$ if $l > m$ or
 $1 \leq i \leq l - 1$ if $m \geq l$, $j = 1, 2, \dots, l - m$ if $l > m$ and $k = 1, 2, \dots, m - l$ if
 $m > l$, $\alpha \in \{here, out, in_t\}$ then,

1. $I_r = \left\{ \left(\left\{ \phi_{zx}^I(E_i, A_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}, \phi_{zy}^I(F_i, B_i) = \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix}, \phi_{zx}^I(G_i, C_i) = \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix}, \phi_{zy}^I(H_i, D_i) = \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \right\}, in_{(r+h)} \right\}$ where $1 \leq i \leq m - 1$ if $l > m$ or $1 \leq i \leq l - 1$ if $m \geq l$, $j = 1, 2, \dots, l - m$ if $l > m$ and $k = 1, 2, \dots, m - l$ if $m > l$,
 $E_i = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$ with $e_1, e_2 \in T \cup \{\varepsilon\}$, $F_i = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$ with $f_1, f_2 \in T \cup \{\varepsilon\}$,
 $G_i = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}$ with $g_1, g_2 \in T \cup \{\varepsilon\}$, $H_i = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$ with $h_1, h_2 \in T \cup \{\varepsilon\}$ and A_i is the maximum possible zx -array context of $xyTr_i$, B_i is the maximum possible zy -array context of $yxTr_i$, $C_i = xxP_zA_j$ and $D_i = yyP_zA_k$.
2. $I_{r+h} = \left\{ \left(\left\{ \phi_{zy}^I(A'_i, E'_i) = \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix}, \phi_{zx}^I(B'_i, F'_i) = \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix}, \phi_{zx}^I(C_i, G'_i) = \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix}, \phi_{zy}^I(D_i, H'_i) = \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \right\}, \alpha \right\}$ where $1 \leq i \leq m - 1$ if $l > m$ or $1 \leq i \leq l - 1$ if $m \geq l$, $j = 1, 2, \dots, l - m$ if $l > m$ and $k = 1, 2, \dots, m - l$ if $m > l$,
 $E'_i = \begin{bmatrix} e'_1 \\ e'_2 \end{bmatrix}$ with $e'_1, e'_2 \in T \cup \{\varepsilon\}$, $F'_i = \begin{bmatrix} f'_1 \\ f'_2 \end{bmatrix}$ with $f'_1, f'_2 \in T \cup \{\varepsilon\}$,
 $G'_i = \begin{bmatrix} g'_1 \\ g'_2 \end{bmatrix}$ with $g'_1, g'_2 \in T \cup \{\varepsilon\}$, $H'_i = \begin{bmatrix} h'_1 \\ h'_2 \end{bmatrix}$ with $h'_1, h'_2 \in T \cup \{\varepsilon\}$ and A'_i is the maximum possible zy -array context of $xyTr_i$, B'_i is the maximum possible zx -array context of $yxTr_i$, $C_i = xxP_zA_j$ and $D_i = yyP_zA_k$.

Case (ii): If $R_r = \left\{ \left(\left\{ \varphi_{yz}(yzTr_i) = \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \$_{yz} [v_i \ v_{i+1}], \varphi_{zy}(zyTr_i) = \begin{bmatrix} u'_i & u'_{i+1} \end{bmatrix} \$_{zy} \begin{bmatrix} v'_i \\ v'_{i+1} \end{bmatrix}, \varphi''_{yy}(yyP_xA_j) = \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} \$_{yy} \begin{bmatrix} v_j \\ v_{j+1} \end{bmatrix}, \varphi'_{zz}(zzP_xA_k) = \begin{bmatrix} u_k & u_{k+1} \end{bmatrix} \$_{zz} [v_k \ v_{k+1}] \right\}, \alpha \right\}$ $1 \leq i \leq n - 1$ if $m > n$ or $1 \leq i \leq m - 1$ if $n \geq m$, $j = 1, 2, \dots, m - n$ if $m > n$ and $k = 1, 2, \dots, n - m$ if $n > m$. $\alpha \in \{here, out, in_t\}$ then we define I_r and I_{r+h} similar to case (i).

Case (iii): If $R_r = \left\{ \left(\left\{ \varphi_{zx}(zxTr_i) = [u_i \ u_{i+1}] \$_{zx} \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix}, \varphi_{xz}(xzTr_i) = \begin{bmatrix} u'_i \\ u'_{i+1} \end{bmatrix} \$_{xz} [v'_i \ v'_{i+1}], \varphi''_{zz}(zzP_yA_j) = [u_j \ u_{j+1}] \$_{zz} [v_j \ v_{j+1}], \varphi''_{xx}(xxP_yA_k) = \begin{bmatrix} u_k \\ u_{k+1} \end{bmatrix} \$_{xx} \begin{bmatrix} v_k \\ v_{k+1} \end{bmatrix} \right\}, \alpha \right\}$ $1 \leq i \leq l - 1$ if $n > l$ or $1 \leq i \leq n - 1$ if $l \geq n$, $j = 1, 2, \dots, n - l$ if $n > l$ and $k = 1, 2, \dots, l - n$ if $l > n$. $\alpha \in \{here, out, in_t\}$ then, we define I_r and I_{r+h} similar to case(i).

4 Conclusion

In this paper, we have compared two hexagonal array generating P systems and proved that the family of hexagonal picture languages generated by the

P systems *PCHADPS* contains the family of hexagonal picture languages generated by the P systems *PCHAPS*. Our future work is to compare the generative powers of (i) PCHADPS and Hexagonal Contextual Array P Systems [1] and (ii) PCHADPS and grammatical models generating hexagonal array languages of Siromoney et al [10].

References

1. Dersanambika, K.S., Krithivasan, K., Agarwal, H.K. and Gupta, J.: Hexagonal Contextual Array P systems, Formal Models, Languages and Applications, Series in Machine Perception and Artificial Intelligence, Vol.66, 2006, 79–96
2. Dersanambika, K.S., Krithivasan, K., Carlos Martin-Vide and Subramanian, K.G.: Local and Recognizable Hexagonal Picture Languages, Int. J. Patt. Recogn. Artif. Intell. 19, 2005, 853–871
3. James Immanuel, S., Thomas, D.G., Robinson, T., Atulya K Nagar and Jayasankar, S.: Parallel Contextual Hexagonal Array P Systems, M. Gong et al (eds); BIC-TA 2016, Part I, CCIS 681, Springer Nature Singapore, 2016, 278–298
4. Jayasankar, S., James Immanuel, S., Thomas, D.G., Robinson, T., Atulya K Nagar : Parallel Contextual Hexagonal Array Insertion Deletion P System, In the Pre-Proceedings of the 6th Asian Conference on Membrane Computing (ACMC 2017), 336-364
5. Marcus, S.: Contextual grammars, Rev. Roum. Math. Pures et Appl. 14(10), 1969, 1525–1534
6. Middleton, L., Sivaswamy, J.: Hexagonal Image Processing: A Practical Approach, Springer- Verlag, 2005
7. Păun, Gh., Computing with membranes, J. Comput. Syst. Sci., 61, 2000, 108–143
8. Păun, Gh.: Marcus Contextual Grammars, kluwer, Dordrecht, 1997
9. Păun, Gh., Rozenberg, G., Salomaa, A. (Eds.): The Oxford Handbook of Membrane Computing, Oxford Univ. Press, 2010
10. Siromoney, G., Siromoney, R.: Hexagonal arrays and rectangular blocks, Computer Graphics and Image Processing 5, 1976, 353-381
11. Thomas, D.G., Begam, M.H., David, N.G.: Parallel Contextual Hexagonal Array Grammars and Languages, LNCS 5852, 2009, 344-357

nc – *eNCE* Graph Grammars and Graph Rewriting P Systems

Jayakrishna V.^{1,2}[0000–0003–4572–4654] and Lisa Mathew¹[0000–0002–3722–3326]

¹ Amal Jyothi College of Engineering, Kanjirappally, Kerala 686518, India
vjayakrishna@amaljyothi.ac.in, lisamathew@amaljyothi.ac.in

² Research scholar, APJ Abdul Kalam Technological University,
Thiruvananthapuram, Kerala 695 016, India

Abstract. Graph grammars are modelling systems capable of generating numerous families of graphs. Graph rewriting systems employ different techniques such as node replacement and edge replacement rewriting. P Systems are capable of modelling different graph based computations in a restricted membrane environment. Here we introduce a new graph grammar model called *nc*–*eNCE* graph grammars. An equivalent graph rewriting P system is also created from the *nc*–*eNCE* graph grammars.

Keywords: Membrane Computing · Graph Grammars · Confluence · Connection Instructions.

1 Introduction

Graph grammars first introduced in the sixties have been an area of interest in theoretical computer science [4, 5, 8, 11–13, 16] due to the variety of possible applications ranging from pattern matching to compiler construction. Graph grammars use graph rewriting to model the evolution of different types of graphs. Graph rewriting is of two types - node replacement and edge replacement [3, 13]. To increase the generative capacity of these grammars, we introduce a variant of the node replacement graph grammar called the non-confluent edge and node label controlled embedding graph grammars. Rewriting P systems [17] are variants of P systems having strings inside the membranes. The strings move into and out of the membranes based on the rules associated with the membranes in which they are present. In [15] the notion of context free graph rewriting P system has been studied. We introduce a modification of this concept called *nc* – *eNCE* graph rewriting P System.

Section 2 discusses the basic concepts of graph grammars. In section 3, we introduce a variant of *eNCE* Graph Grammars. Section 4 deals with *nc*–*eNCE* graph rewriting P Systems which form an alternative mechanism to generate the same classes of graphs.

2 Preliminaries

2.1 Graph Grammars

A typical graph grammar [8, 9] consists of a host graph H , and a set of transformation rules (productions) P of the form $P(M, D, E)$. Here M is the mother graph present in H (to be replaced), D is the daughter graph (to be embedded with the remaining portion ($H' = H - M$) of the host graph) and E is the embedding mechanism.

The string graph corresponding to $w = x_1x_2x_3 \cdots x_n$, is given by $G = (V, E)$ where $V = \{v_1, v_2, v_3, \dots, v_n\}$ where v_i is labelled x_i and $E = \{(v_i, v_{i+1}) | i = 1, 2, 3, \dots, n-1\}$. We use the string w to represent the string graph corresponding to w .

Two different types of embedding mechanisms appear in the literature- gluing and connecting. In a graph grammar based on gluing [4, 5, 12, 13], certain nodes of H' are identified for fusion with nodes of D after the removal of M . Certain edges incident on M in H are also chosen to be fused with edges in D .

In the second approach [11], to compensate for those edges of H which were not part of M but were lost due to removal of M , the daughter graph D is connected to H' by introducing new edges.

Node Replacement Graph Grammars In a node replacement graph grammar [3], the mother graph M consists of a single node and each production rule has connection instructions associated with it.

In a *Node Label Controlled (NLC)* [7] graph grammar, embedding is done on undirected graphs with node labels. Application of a production results in the mother node A (in a host graph H) being replaced by the daughter graph D . Here, the label A is a non terminal while D is a graph with nodes whose labels may be terminal or non-terminal connected with undirected edges. Connection instructions are common for all the productions. These are in the form of ordered pairs (α, β) where α and β can be terminal or non terminal symbols. The instruction introduces an undirected edge between each neighbour of A (in H) labelled α and each node in D labelled β . The formal definition is as follows:

Definition 1. [7] An NLC graph grammar is a construct $nG = (\Sigma, \Gamma, P, C, S)$, where:

- Σ is the set of all node labels,
- Γ is the set of all terminal node labels,
- P is the finite set of productions of the form $A \rightarrow D$, where A is the label of the mother node (in the host graph) and D is the daughter graph,
- $C : \Sigma \times \Sigma$ is the set of connection instructions, and
- S is the start/initial graph.

The graph language represented by this grammar [14] is

$$L(nG) = \{G \in G_\Gamma | S \xrightarrow{*} G\}$$

where G_Γ is a set of graphs containing only nodes with labels from Γ .

Another variation of *NLC* known as *boundary NLC* is given in [6, 11]. Here the introduction of an edge between the nodes of host graph and the daughter graph with non-terminal labels is not permitted.

An *NCE* or *Neighbourhood Controlled Embedding* [11] is of the form: ${}_eG = (\Sigma, \Gamma, P, S)$. It is similar to the *NLC* graph grammar definition except that each production has independent connection instructions. In general an *NCE* production p is of the form, $p: A \rightarrow (D, C)$ where $A \rightarrow D$ is a production with mother node A and daughter graph D . $\{C \subseteq \Sigma \times N_D\}$ is the connection relation for p and is of the form (α, β) where α is the node label of one of the neighbours of the mother graph and β is a node in the daughter graph. Here N_D is the set of nodes of D and Σ is the set of all node labels.

The *NLC* embedding [7] can further be extended to undirected graphs with labelled edges. The labelled edges connecting the mother node with its neighbours come into play during the embedding process. This type of embedding mechanisms is called *eNCE* or *edge and node controlled embedding* [11]. An *eNCE* graph grammar is similar to an *NCE* graph grammar with changes only in the connection relations. The formal definition of *eNCE* is as follows:

Definition 2. [18] ${}_eG = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ where:

Σ is the set of all node labels,

Δ is the set of all terminal node labels,

Γ is the set of all edge labels,

Ω is the set of terminal edge labels,

The productions in P are of the form $p: A \rightarrow (D, C)$ where A is the label of the mother node and D is the daughter graph, C is the connection instruction for that production and is of the form: $(a, p \mid q, B)$, where p and q are edge labels, a is the node label of one of the neighbours of the mother node and B is a node in D . The interpretation is that we find an edge labelled p in the host graph which had connected a node x labelled a to the mother node and create a new edge labelled q between x and the node B in the daughter graph. S is the start/initial graph.

The language represented by this grammar [14] is $L({}_eG) = \{G \in G_\Delta \mid S \xRightarrow{*} G\}$ where G_Δ is the set of graphs containing only terminal labelled nodes.

In section 3 we introduce a non-confluent variant of the *eNCE* graph grammars.

2.2 Membrane Computing

The membrane computing computational model was inspired by cell biology, imitating the way in which chemicals interact and cross cell membranes. This model usually referred to as a P system was first introduced by Gheorghe Pun [17] and has been studied extensively [1, 10]. Rewriting P systems consider string objects and manipulate them using rewriting rules. Rules are applied on symbols, sub-strings or even structures to make the objects move from one membrane to another. Rewriting P systems with conditional communications [2] are also extensively used especially for computing structures such as graphs. Other enhancements and developments on graph rewriting P systems has been studied in [19].

3 Non-confluent Edge and Node Controlled Embedding (nc – eNCE) Graph Grammar

Formally, we have:

Definition 3. An *nc – eNCE graph grammar* is a 7 tuple: $ncG = (\Sigma, \Delta, \Gamma, \Omega, P, S, R(P))$ where

- Σ is the set of node labels,
 - Δ is the set of terminal node labels,
 - Γ is the set of edge labels,
 - Ω is the set of terminal edge labels [16],
 - The productions in P are defined as in Definition 2,
 - S is the start node/initial graph,
 - $R(P)$ is a regular expression which specifies the order of application of the productions.
- Hence this grammar becomes restricted or non-confluent.

The language represented by this grammar is $L(ncG) = \{G \in G_\Delta \mid S \xrightarrow{R(P)} G\}$ where G_Δ is the set of graphs containing only terminal nodes. Hence the language of an *nc – eNCE graph grammar* is a set of graphs whose nodes have terminal labels, generated by applying a series of productions in the order p_1, p_2, \dots, p_n where $p_1 p_2 \dots p_n$ is a word in the language represented by the regular expression $R(P)$.

3.1 Generation of Graph language using *nc – eNCE graph grammar*

The following example shows the generation of complete graphs using *nc – eNCE graph grammar* $ncCG$

Example 1. Let $ncCG = (\Sigma, \Delta, \Gamma, \Omega, P, S, R(P))$ be an *nc – eNCE graph grammar* with, $\Sigma = \{S, a\}$, $\Delta = \{a\}$, $\Gamma = \{\alpha\}$, $\Omega = \{\alpha\}$, $P = \{p_1, p_2\}$ and $R(P) = p_1^* p_2$. Figure 1 shows the production rules and the associated connection instructions for generating complete graphs.

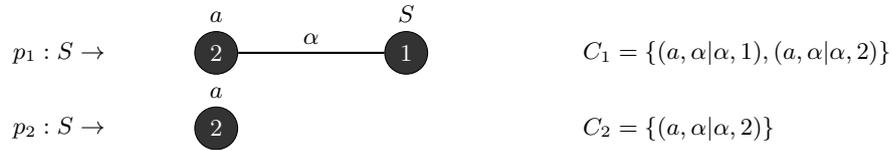


Fig. 1. Productions for generating complete graphs

Figure 2 shows the construction of K_4 , the complete graph on 4 vertices, using the grammar $ncCG$.

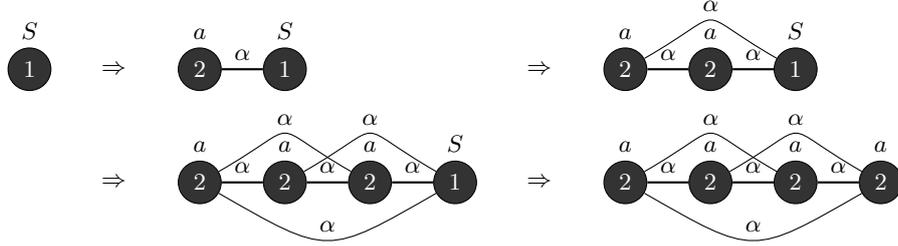


Fig. 2. Generation of K_4 using $ncCG$

4 $nc - eNCE$ Graph Rewriting P Systems

In this section we define an $nc - eNCE$ graph rewriting P systems that is inspired by $nc - eNCE$ graph grammars. The regular expression used to generate the graph languages in $nc - eNCE$ graph grammars is replaced by an equivalent system of nested membranes. The number of membranes required for performing the computation can be calculated using the regular expression $R(P)$ present in the corresponding $nc - eNCE$ graph grammar capable of generating the desired graph. The formal definition of $nc - eNCE$ Graph Rewriting P Systems is as follows:

Definition 4. An $nc - eNCE$ graph rewriting P System is a construct:

$$\Pi_{nc-eNCE} = (N, T, E, M, S, R_1, R_2, \dots, R_n, i_o) \text{ where}$$

- N is the set of non-terminal node labels.
- T is the set of terminal node labels.
- E is the set of edge labels. [16]
- M is a membrane structure with n membranes and depth d , which are labelled by numbers from the set of natural numbers. The skin membrane is labelled as 1.
- S is a non-terminal node over N initially present in each of the regions at maximal depth.
- R_i s are the graph grammar rules in region i and are of the form $P_k(tar)$,
- i_o is the label of the output membrane.

The family of graph language generated by the $nc - eNCE$ graph rewriting P System is denoted by $L(\Pi_{nc-eNCE})_n^d$, where n is the total number of membranes and d is the depth.

4.1 From $nc - eNCE$ graph grammar to $nc - eNCE$ graph rewriting P System

Given an $nc - eNCE$ graph grammar $ncG = (\Sigma, \Delta, \Gamma, \Omega, P, S, R(P))$ capable of generating a graph language $L(ncG)$, an equivalent $nc - eNCE$ graph rewriting P system

$$\Pi_{nc-eNCE} = (N, T, E, M, S, R_1, R_2, \dots, R_n, i_o)$$

with $N = \Sigma - \Delta, T = \Delta, E = \Gamma, i_o = 1$ can be obtained. Suppose Π_X and Π_Y have been constructed corresponding to regular expressions X and Y with k and l membranes respectively. Let $R_i(X)$ be the set of graph grammar rules in the i^{th} membrane of Π_X . Then the graph rewriting P system $\Pi_{R(P)}$ is constructed recursively as follows:

- if $R(P)$ consist of a single literal p , then $M = []_1$ and $R_1 = \{p, (\text{here})\}$.
- if $R(P) = X + Y$, then $M = [[[\dots []_k \dots]_3]_2 [[\dots []_{k+l-1} \dots]_{k+2}]_{k+1}]_1$,

$$R_i(X + Y) = \begin{cases} R_1(X) \cup R_1(Y) & \text{if } i = 1 \\ R_i(X), & \text{if } 2 \leq i \leq k \\ R_{i+1-k}(Y), & \text{if } k + 1 \leq i \leq k + l - 1 \end{cases}$$

and the non-terminal S is initially present in membranes k and $k + l - 1$

- if $R(P) = X \cdot Y$, then $M = [[[\dots [[]_{k+l}]_{k+l-1} \dots]_l \dots]_3]_2]_1$, and the non-terminal S is initially present in membrane $k + l$,

$$R_i(X \cdot Y) = \begin{cases} R_i(Y) & 1 \leq i \leq l \\ R_1(X) \cup \{A \rightarrow A(\text{out}) \mid (A \rightarrow \omega) \in R_1(X)\} & i = l + 1 \\ R_{i-l}(X) & l + 2 \leq i \leq k + l \end{cases}$$

- if $R(P) = X^*$, then $M = [[[\dots [[]_{k+1}]_k \dots]_3]_2]_1$, and the non-terminal S is initially present in membrane $k + 1$,

$$R_i(X^*) = \begin{cases} \emptyset & i = 1 \\ R_1(X) \cup \{A \rightarrow A(\text{out}) \mid (A \rightarrow \omega) \in R_1(X)\} & i = 2 \\ R_{i-1}(X) & 3 \leq i \leq k + 1 \end{cases}$$

It may be noted that the number of membranes in the $nc - eNCE$ graph rewriting P System is one more than the number of concatenations in the expanded form of $R(P)$. The depth of the system is one greater than the maximum number of concatenations present in a single term in $R(P)$.

Example 2. Consider the graph language represented by example 1 of section 3.1. The following $nc - eNCE$ graph rewriting P System Π_{ncCG} inspired from the graph grammar $ncCG$ is capable of generating the same graph language is as follows:

$$\Pi_{ncCG} = (\{S\}, \{a\}, [[[]_3]_2]_1, S, \{R_{11}\}, \{R_{21}\}, \{R_{31}, R_{32}\}, 1)$$

where

R_{11}, R_{21}, R_{31} , and R_{32} are given by

$$\begin{array}{ll} R_{31} : S \rightarrow \begin{array}{c} a \\ \bullet 2 \end{array} \xrightarrow{\alpha} \begin{array}{c} S \\ \bullet 1 \end{array} & C_1 = \{(a, \alpha | \alpha, 1), (a, \alpha | \alpha, 2)\}_{(\text{here})} \\ R_{32} : S \rightarrow S & (\text{out}) \\ R_{21} : S \rightarrow \begin{array}{c} S \\ a \\ \bullet \end{array} & (\text{out}) \\ R_{11} : S \rightarrow \begin{array}{c} \bullet 2 \end{array} & C_2 = \{(a, \alpha | \alpha, 2)\}_{(\text{here})} \end{array}$$

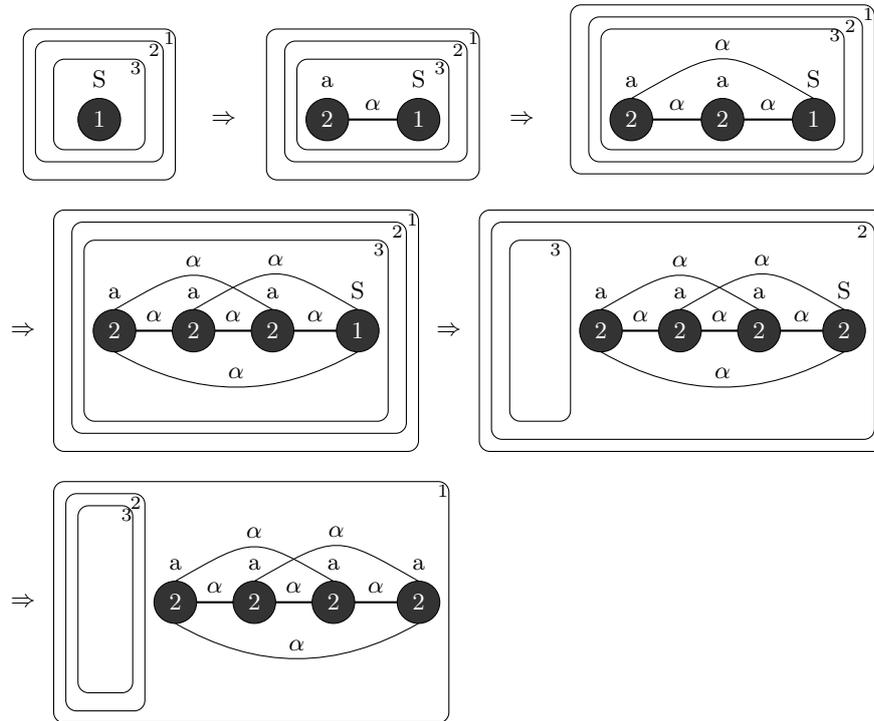


Fig. 3. Generation of K_4 using Π_{ncCG}

5 Conclusion

We have defined some variants of edge and node controlled embedding graph grammars and the equivalent P system. It remains to be seen whether the membrane computing framework improves the generative power of the graph grammar. It will be of interest to apply such computing systems in biochemical modelling and structural analysis of DNA and proteins.

References

1. Administrator, T.P.P.: Home, "http://ppage.psystems.eu/"
2. Bottoni, P., Labella, A., Martín-Vide, C., Paun, G.: Rewriting p systems with conditional communication. In: Formal and Natural Computing (2002)
3. Drewes, F., Kreowski, H.J., Habel, A.: Hyperedge replacement graph grammars. Handbook of Graph Grammars and Computing by Graph Transformation pp. 95–162 (1997)
4. Ehrig, H.: Tutorial introduction to the algebraic approach of graph grammars. Springer, Graph Grammars and Their Application to Computer Science. Graph Grammars 1986. Lecture Notes in Computer Science, vol 291 pp. 3–14 (1986)
5. Ehrig, H., Habel, A., Kreowski, H.J.: Introduction to graph grammars with applications to semantic networks. Computers & Mathematics with Applications **23**(6),

- 557 – 572 (1992). [https://doi.org/https://doi.org/10.1016/0898-1221\(92\)90124-Z](https://doi.org/https://doi.org/10.1016/0898-1221(92)90124-Z), <http://www.sciencedirect.com/science/article/pii/089812219290124Z>
6. Engelfriet, J., Rozenberg, G.: A comparison of boundary graph grammars and context-free hypergraph grammars. *Inf. Comput.* **84**(2), 163206 (Feb 1990). [https://doi.org/10.1016/0890-5401\(90\)90038-J](https://doi.org/10.1016/0890-5401(90)90038-J), [https://doi.org/10.1016/0890-5401\(90\)90038-J](https://doi.org/10.1016/0890-5401(90)90038-J)
 7. Engelfriet, J., Rozenberg, G.: Graph grammars based on node rewriting: an introduction to nlc graph grammars. Springer Berlin Heidelberg pp. 12–23 (1991)
 8. Engelfriet, J., Vereijken, J.J.: Concatenation of graphs. *Graph Grammars and Their Application to Computer Science. Graph Grammars 1994. Lecture Notes in Computer Science*, vol 1073 pp. 368–382 (2005)
 9. Fahmy, H., Blostein, D.: A survey of graph grammars: theory and applications. In: *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems.* pp. 294–298 (1992). <https://doi.org/10.1109/ICPR.1992.201776>
 10. Gheorghe, P., Grzegorz, R., Arto, S.: An introduction to and an overview of membrane computing, pp. 1–27. Oxford University Press (2009), <https://fdslive.oup.com/www.oup.com/academic/pdf/13/9780199556670.pdf>
 11. Grzegorz, R.: Node replacement graph grammars. *Handbook of Graph Grammars and Computing by Graph Transformation* pp. 1–94 (1997)
 12. Habel, A., Kreowski, H.: May we introduce to you: Hyperedge replacement. Springer, *Graph Grammars and Their Application to Computer Science. Graph Grammars 1986. Lecture Notes in Computer Science*, vol 291 pp. 15–26 (1986)
 13. H.Ehrig: Introduction to the algebraic theory of graph grammars (a survey). Springer, *Graph-Grammars and Their Application to Computer Science and Biology, International Workshop, Bad Honnef, October 30 - November 3, 1978. Lecture Notes in Computer Science*, vol 1073 pp. 1–69 (1978)
 14. Janssens, D., Rozenberg, G.: Generating graph languages using hypergraph grammars. Springer Berlin Heidelberg pp. 154–164 (1981)
 15. Murugeswari, V.T., Sheela, J.E.P.: Context-free graph p system. *International Journal of Artificial Intelligence and Soft Computing* **7**(1), 86–94 (2020). <https://doi.org/https://dx.doi.org/10.1504/IJAISC.2019.105021>
 16. Pavlidis, T.: Linear and context-free graph grammars. *Journal of the ACM* **19**(1), 1122 (1972). <https://doi.org/10.1145/321356.321364>
 17. Pun, G., Rozenberg, G.: A guide to membrane computing. *Theoretical Computer Science* **287**(1), 73–100 (2002). [https://doi.org/https://doi.org/10.1016/S0304-3975\(02\)00136-6](https://doi.org/https://doi.org/10.1016/S0304-3975(02)00136-6), <https://www.sciencedirect.com/science/article/pii/S0304397502001366>, natural Computing
 18. Rozenberg, G., Salomaa, A.: *Handbook of formal languages.* Springer (1997)
 19. Sankar, M.P., David, N.G.: On generative power of rewriting graph p system. In: Dash, S.S., Lakshmi, C., Das, S., Panigrahi, B.K. (eds.) *Artificial Intelligence and Evolutionary Computations in Engineering Systems.* pp. 419–429. Springer Singapore, Singapore (2020)

Variants of Tetrahedral Tile Pasting P System Model for 3D Patterns

T. Kalyani^{1*}, T.T. Raman¹, D.G. Thomas²,
K. Bhuvaneshwari³, and P. Ravichandran¹

¹ Department of Mathematics, St. Joseph's Institute of Technology, Chennai - 600119, India,
kalphd02@yahoo.com

² Department of Science and Humanities (Mathematics Division), Saveetha School of Engineering, Chennai - 602105, India

³ Department of Mathematics, Sathayabama Institute of Science and Technology, Chennai - 600119, India

Abstract. K -Tabled Tetrahedral Tile Pasting System (K -TTTPS) and Tetrahedral Tile Pasting P System (TetTPPS) were introduced to generate 3D patterns. Tile pasting system is a parallel generating model which glues two square tiles at its edges to create intricate tiling patterns. It is extended to three dimensions using tetrahedral tiles and the tessellations generated are used to cover floors and walls. In this paper Controlled Tabled Tetrahedral Tile Pasting System (CTTTPS), Extended Tetrahedral Tile Pasting System (ETTPS) and Extended Tetrahedral Tile Pasting P System (ETTPPS) are proposed to generate 3D patterns and they are compared for generative powers. CTTTPS is compared with K -TTTPS and TetTPPS and study some of its properties. It is proved that CTTTPS has more generative capacity than K -TTTPS and TetTPPS.

Keywords: tetrahedral tiles, pasting system, membrane computing, picture languages

1 Introduction

Syntactic methods such as Pasting System and its variants like Extended Pasting System, Tabled Pasting System and Parametric Pasting System plays a vital role in picture generation [5]. Tile Pasting P System model for pattern generation was introduced by Robinson et.al., in [2, 9]. Later on, it was extended to isosceles right triangular tiles to generate two - dimensional tiling patterns in [4]. Many variations of triangular tile pasting system was introduced in [5]. Extending this idea to three dimensions, Tetrahedral Tile Pasting System generating Tetrahedral Picture Languages was proposed in [3, 7, 8].

On the other hand, in the area of membrane computing, Gh. Păun introduced P system [6] which is a new computability model. This model computes patterns to be generated by the system by processing the multi sets of objects present in the region that are defined by a hierarchical arrangement of membranes, by evolution rules associated with the region.

* Corresponding author.

These two areas, namely membrane computing and picture grammars were linked together by Ceterchi et.al. [11], by relating P systems and array-rewriting grammars generating picture languages and proposed array-rewriting P systems [1, 10].

In this paper, we propose controlled Tabled Tetrahedral Tile Pasting System, Extended Tetrahedral Tile Pasting System (ETTPS) and Extended Tetrahedral Tile Pasting P System (ETTPPS) to generate 3D patterns and they are compared for generative powers. CTTTPS is compared with K -Tabled Tetrahedral Tile Pasting System (K -TTTPS) and Tetrahedral Tile Pasting P System (TetTPPS) for generative power and study some of its properties.

2 Preliminaries

We refer [3] for the definitions of tetrahedral tile, K -TTTPS and TetTPPS.

3 Controlled Tabled Tetrahedral Tile Pasting System

In this section, we introduce Controlled Tabled Tetrahedral Tile Pasting System (CTTTPS) to generate 3D patterns and study some of its properties.

Definition 1. A Controlled Tabled Tetrahedral Tile Pasting System (CTTTPS) is a 5-tuple $S = (\Sigma, E, P, C, t_0)$ where Σ is a finite set of tetrahedral tiles. E is a finite set of edge labels of base of tetrahedral tiles. P is a finite set of tables $\{T_1, T_2, \dots, T_k\}$, $k \geq 1$ and each T_i , $i = 1, 2, \dots, k$ is one of left, right, up, down, rightup, righdown, leftup and leftdown rules only. The rules of the tables are applied in parallel to the respective edges of the pattern derived in the previous stage. C is a control language over P and $t_0 \in \Sigma^{++}$ is the axiom pattern.

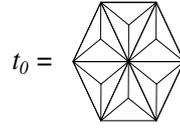
A pattern t_j is generated from t_0 by applying the control word $w = w_1 w_2 \dots w_j \in C$ where $w_i \in C (1 \leq i \leq j)$ if there exists a sequence of derivations $t_0 \Rightarrow_{w_1} t_1 \Rightarrow_{w_2} t_2 \Rightarrow \dots \Rightarrow_{w_j} t_j$ and is denoted by $t_0 \xRightarrow{*}_w t_j$

The set of all patterns generated by the system is $T(S) = \{t_j \in \Sigma^{+++} : t_0 \xRightarrow{*}_C t_j / j \geq 0\}$. The control language C may be either regular, context free or context sensitive, in which case we attach the corresponding name to the control.

Example 1. Consider a CTTTPS, $S_1 = (\Sigma, E, P, C, t_0)$, where

$$\Sigma = \left(\begin{array}{c} \begin{array}{c} \triangle \\ \diagup \quad \diagdown \\ a_1 \quad a_3 \\ \diagdown \quad \diagup \\ a_2 \\ A \end{array} \quad , \quad \begin{array}{c} \triangle \\ \diagdown \quad \diagup \\ b_2 \\ \diagup \quad \diagdown \\ b_3 \quad b_1 \\ B \end{array} \end{array} \right)$$

$E = \{a_1, a_2, a_3, b_1, b_2, b_3\}$, $P = \{L_u, D, R_u, R_d, L_d, U\}$, $L_u = \{(a_1, b_1)\}$, $D = \{(a_2, b_2)\}$, $R_u = \{(a_3, b_3)\}$, $R_d = \{(b_1, a_1)\}$, $U = \{(b_2, a_2)\}$, $L_d = \{(b_3, a_3)\}$, $C = \{(UR_u R_d D L_d L_u)^n / n \geq 1\}$ is a regular control and



The first three members of $T(S_1)$ are shown in Fig. 6.

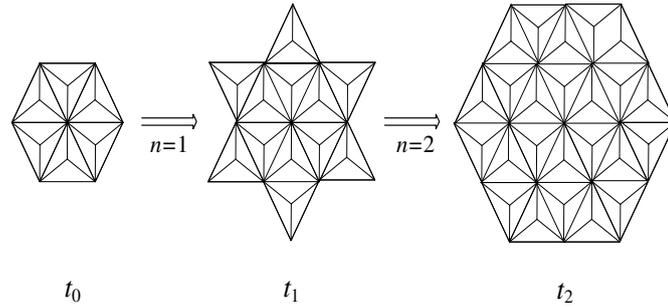
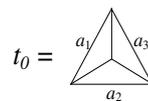


Fig. 1. Star and hexagonal polyhedral

Example 2. Consider a CTTTPS, $S_2 = (\Sigma, E, P, C, t_0)$, where

$$\Sigma = \left(\begin{array}{c} \begin{array}{c} a_1 \quad a_3 \\ \diagdown \quad \diagup \\ a_2 \\ A \end{array}, \quad \begin{array}{c} b_2 \\ \diagdown \quad \diagup \\ b_3 \quad b_1 \\ B \end{array} \end{array} \right)$$

$E = \{a_1, a_2, a_3, b_1, b_2, b_3\}$, $P = \{R_u, R_d, U\}$, $C = \{(R_u R_d U)^n / n \geq 1\}$ $R_u = \{(a_3, b_3)\}$, $R_d = \{(b_1, a_1)\}$, $U = \{(b_2, a_2)\}$, and



A tessellation obtained by S_2 is shown in Fig. 7.

Example 3. In Example 2, if we use context sensitive control $C = \{(R_u R_d U)^{2^n - 1} / n \geq 1\}$, we get equilateral triangular tetrahedral of size $2n, n \geq 1$.

Theorem 1. The family of the set of all patterns generated by K -TTTPS is properly included in the family of the set of all patterns generated by CTTTPS.

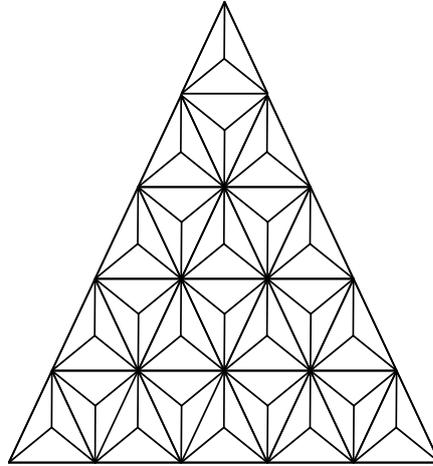


Fig. 2. Triangular polyhedral

Proof. The patterns generated by K -TTTTPS $M = (\Gamma, E, P, t_0)$ can be generated by a Controlled Tabled Tetrahedral Tile Pasting System. We now construct a CTTTTPS $S = \{\Sigma, E_1, P_1, C, t_0\}$ as follows:

Let $\Sigma = \Gamma, E_1 = E, P_1 = P$ and $C = \{(T_1 T_2 \dots T_k)^n / n \geq 1\}$. It is clear that the patterns generated by M can be generated by S . But the patterns generated by CTTTTPS given in Example 3 cannot be generated by any K -TTTTPS. This completes the proof.

Theorem 2. *The family of all patterns generated by TetTPPS is properly included in the family of all patterns generated by CTTTTPS.*

Proof. The patterns generated by TetTPPS $\Pi = (\Gamma, \mu, F_1, \dots, F_m, R_1, \dots, R_m, t_0)$ can be generated by a controlled tabled tetrahedral tile pasting system. We now construct a CTTTTPS $S = (\Sigma, E, P, C, t_0)$ as follows:

Let $\Sigma = \Gamma, E = \{\text{edge labels of tiles in } \Gamma\}, P = \{T_1, T_2, \dots, T_m\}$ where $T_i = \{(x_i, y_i) / (x_i, y_i) \in R_i\} 1 \leq i \leq m, C = \{(T_1 T_2 \dots T_m)^n / n \geq 1\}$ and $t_0 = F_1$. It is clear that the patterns generated by Π can be generated by S . But the patterns generated by CTTTTPS given in Example 3 cannot be generated by any TetTPPS as context sensitive control is used in CTTTTPS.

Definition 2. *In a controlled tabled tetrahedral tile pasting system $S = (\Sigma, E, P, C, t_0)$,*

- (1) *The pasting rule $(a_3, b_3) \in P$ is said to be symmetric if $(a_1, b_1) \in P$, right neighbour of a tile is a left neighbour.*
- (2) *The pasting rule $(a_2, b_2) \in P$ is symmetric if $(b_2, a_2) \in P$.*
- (3) *The pasting rule $(b_1, a_1) \in P$ is symmetric if $(b_3, a_3) \in P$.*
- (4) *The pasting rule $(b_1, a_1) \in P$ is symmetric if $(b_3, a_3) \in P$.*

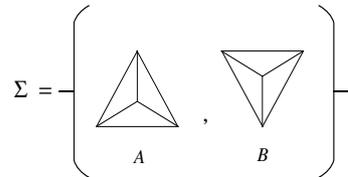
Remark 1. Example 1 is a symmetric pasting system.

Definition 3. *A Controlled Tabled Tetrahedral Tile Pasting System CTTTTPS is said to be a symmetric pasting system if every rule in P is symmetric.*

Proposition 1. *The set of all patterns generated by symmetric pasting systems are strictly included in the family of patterns generated by CTTTPS.*

Proof. The inclusion is straight forward proper inclusion can be seen as follows. The patterns with hole given in Example 2 cannot be generated by any symmetric P system.

Definition 4. *Let*



The reversal of tetrahedral tiles A and B are B and A and they are denoted by A^{rev} , B^{rev} respectively. The reversal of pasting rules re given below: $(a_1, b_1)^{rev} = (b_3, a_3)$, $(a_2, b_2)^{rev} = (b_2, a_2)$, $(a_3, b_3)^{rev} = (b_1, a_1)$

Theorem 3. *The family of all patterns generated by CTTTPS is closed under reversal of patterns.*

Proof. Let $M = T(S) = \{t_j \in \Sigma^{**} / j \geq 0\}$ be generated by a controlled tabled tetrahedral tile pasting system $S = (\Sigma, E, P, C, t_0)$. We construct a CTTTPS $S_1 = (\Sigma_1, E_1, P_1, C_1, t_{01})$ to generate M^{rev} as follows:

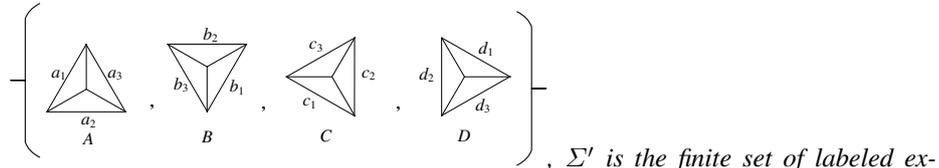
Let $\Sigma_1 = \Sigma^{rev}$, $E_1 = E$, $P_1 = \{T_{11}, T_{12}, \dots, T_{1m}\}$ $T_{1i} = T_i^{rev} = \{(x, y)^{rev} / (x, y) \in T_i\}$, $1 \leq i \leq m$ C_1 is formed by reversing the tables used in C . $t_{01} = t_0^{rev}$. Now S_1 generates the sequence of patterns $T(S_1) = \{t_j^{rev} / t_j \in T(s)\}$. Then $x = t_j^{rev}$ for some j , which implies that $x \in M^{rev}$ and so $T(S_1) \subseteq M^{rev}$.

Conversely let $c \in M^{rev}$. Then $x^{rev} \in M = T(S)$, which implies that $x^{rev} = t_j$ for some j and so $x = t_j^{rev} \in T(S_1)$. This implies that $M^{rev} \subseteq T(S_1)$, which proves that $T(S_1) = M^{rev}$. Hence the family of all patterns generated by CTTTPS is closed under reversal of patterns.

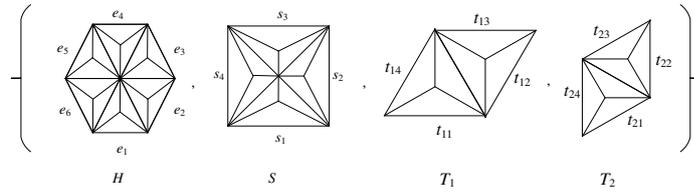
4 Extended Tetrahedral Tile Pasting System and P System

In this section, we propose Extended Tetrahedral Tile Pasting System (ETTPS) and Extended Tetrahedral Tile Pasting P System (ETTPPS).

Definition 5. *An extended tetrahedral tile pasting system ETTPS is a 6 tuple $S = (\Sigma, \Sigma', E, P, t_0, \Delta)$ where Σ is the finite set of labeled tetrahedral tiles*



, Σ' is the finite set of labeled extended tetrahedral tiles formed using glueable tetrahedral tiles of the form



E is the edge labels of tetrahedral and extended tetrahedral tiles in Σ and Σ' . P is a finite set of pasting rules, t_0 is the axiom pattern in $\Sigma \cup \Sigma'$ and Δ is the condition over the pasting rules which is used to complete the tetrahedral picture pattern and it is defined as

$$\Delta = \{(\ell_j m_j, n_k) / \ell_i, m_j \in E(\Sigma'), n_k \in E(\Sigma), i, j, k \geq 1\}.$$

The class of tetrahedral picture languages generated by this system is denoted by \mathcal{L}_S .

Example 4. Consider the extended tetrahedral tile pasting system $S_1 = (\Sigma, \Sigma', E, P, t_0, \Delta)$, where $\Sigma = \{A, B\}$, $\Sigma' = \{H\}$, $E = \{a_1, a_2, a_3, b_1, b_2, b_3, e_1, e_2, e_3, e_4, e_5, e_6\}$, $P = \{(e_1, b_2), (e_2, a_1), (e_3, b_3), (e_4, a_2), (e_5, b_1), (e_6, a_3), (a_2, e_4), (b_2, e_1), (a_3, e_6), (b_3, e_3)\}$ and $\Delta = \{(e_2 e_4, a_2), (e_1 e_3, b_2), (e_2 e_6, a_1), (e_5 e_1, b_1), (e_4 e_6, a_2), (e_3 e_5, b_3)\}$.

The language of hexagonal tetrahedral is generated by parallel mechanism and is shown in Fig. 9.

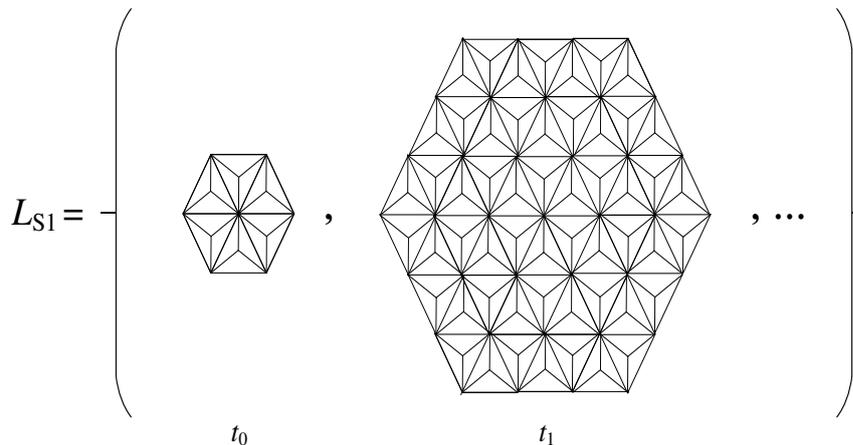


Fig. 3. Language of hexagonal tetrahedral

Theorem 4. The family of the set of all patterns generated by ETTPS intersects the family of the set of all patterns generated by CTTPS.

Proof. The language of hexagonal tetrahedral polyhedral is generated by both systems as in Examples 3 and 7.

Definition 6. An Extended Tetrahedral Tile Pasting P System (ETTPPS) is defined as $\pi = (\Sigma, \Sigma', E, \mu, F_1, F_2, \dots, F_m, R_1, R_2, \dots, R_m, \Delta, i_0)$ where

Σ - Finite set of labeled tetrahedral tiles

Σ' - Finite set of labeled extended tetrahedral tiles formed using gluable tetrahedral tiles

E - edge labels of tetrahedral and extended tetrahedral tiles in Σ and Σ'

μ - The membrane structure having 'm' membranes

F_i - Finite sets of three-dimensional picture patterns over tiles of $\Sigma \cup \Sigma'$, $1 \leq i \leq m$

R_i - Finite sets of pasting rules associated with m regions of the membrane structure, $1 \leq i \leq m$

Δ - The condition over the pasting rules to complete the picture pattern and it is defined in the output region.

$\Delta = \{((\ell_i, m_j, n_k), tar) / \ell_i, m_j \in E(\Sigma'), n_k \in E(\Sigma), i, j, k \geq 1\}$

i_0 - The output membrane.

The computation is successful if there is no possibility of applying the tetrahedral tile pasting rules. Then the computation is stopped and the resultant picture is the halting one, which is collected in the output region. The set of all picture patterns generated by ETTPPS is denoted by $ETTPPL(\pi)$. The family of such languages with atmost m membranes is denoted by $PL_m(ETTPPS)$.

Example 5. The language of hexagonal tetrahedral given in Example 7 is constructed by the ETTPPS $\pi = (\Sigma, \Sigma', E, \mu, F_1, F_2, R_1, R_2, \Delta, 1)$, where $\Sigma = \{A, B\}$, $\Sigma' = \{H\}$, $E = \{a_1, a_2, a_3, b_1, b_2, b_3, e_1, e_2, e_3, e_4, e_5, e_6\}$, $\mu = [1[2]2]_1$, $F_1 = H$, $F_2 = \emptyset$.
 $R_1 = \{((e_1, b_2), here), ((e_2, a_1), here), ((e_3, b_3), here), ((e_4, a_2), here), ((e_5, b_1), here), ((e_6, a_3), in)\}$
 $R_2 = \{((a_2, e_4), here), ((b_2, e_1), here), ((a_3, e_6), here), ((b_3, e_3), out)\}$
 $\Delta = \{((e_2e_4, a_2), here), ((e_1e_3, b_2), here), ((e_2e_6, a_1), here), ((e_5e_1, b_1), here), ((e_4e_6, a_2), here), ((e_3e_5, b_3), here), ((e_3e_5, b_3), in)\}$.

Theorem 5. The family of the set of all patterns generated by ETTPS and ETTPPS intersects.

Proof. The language of hexagonal tetrahedral polyhedral is generated by both systems as in Examples 7 and 8.

5 Conclusion

In this paper we have used tetrahedral tiles to generate three-dimensional patterns using CTTTPS, ETTPS and ETTPPS. CTTTPS is compared with K -TTTPS and TetTPPs and it is proved that CTTTPS has more generative capacity and studied some properties of CTTTPS. The other variants of pasting systems can also be considered and learning of CTTTPS can be done using positive samples. This is our future work.

References

1. D. Giammarresi, A. Restivo, Two - dimensional languages, In: Rozenberg, G., et.al., Handbook of Formal Languages, Vol III, 215-268, Springer, Heidelberg (1997).
2. T. Robinson, Atulya K Nagar and S. Jebasingh, Tile Pasting P System with multiple-edge pasting, $K - Deb$ et.al., (Eds): SEAL 2010, LNCS 6457, 329–338, 2010. Springer - Verlag, Berlin, Heidelberg 2010.
3. T. Kalyani, T.T. Raman, D. Gnanaraj Thomas, Tetrahedral tile pasting P system for 3D patterns, Mathematics in Engineering, Science and Aerospace, Vol 11, No 1, 255–263, 2020.
4. T. Kalyani, K. Sasikala, V.R., Dare, T. Robnison, Triangular Pasting System In : K.G. Subramanian, K. Rangarajan, M. Mukund (Eds.), Formal Models, Languages and Applications, Series in Machine Perception and Artificial Intelligence, Vol.66 (2006), 195-211.
5. K. Bhuvaneswari, T. Kalyani, D. Lalitha, Triangular Tile Pasting P system and array Generating Petri Nets, International Journal of Pure and Applied Mathematics, 107 (1) (2016) 111-128.
6. Gh. Păun, Computing with Membranes, Journal of Computer and System Sciences, 61(1) (2000) 108–143.
7. T. Kalyani, K. Sasikala, D.G. Thomas, T. Robinson, Atulya K Nagar, Meenakshi Paramasivan, 3D-Array token petri nets generating tetrahedral pincture languages, LNCS, 12148, 88–105.
8. T.T. Raman, T. Kalyani, D.G. Thomas, Tetrahedral array grammar system, Mathematics in Engineering, Sciene and Aerospace, Vol 11 No.1, (2020) 215–226.
9. T. Robinson, Extended Pasting Scheme for Kolam Pattern generation, Forma, 22 (2007), 55-64.
10. K.G. Subramanian, R. Saravanan, T. Robinson P Systems for array generation and application to kolam Patterns, Forma 22 (2007) 47–54.
11. R. Ceterchi, M. Mutyam, Gh. Paun and K.G. Subramanian, Array-rewriting P systems, Natural Computing, 2 (2003), 229–249.

Weighted Spiking Neural P Systems with Polarizations and Anti-spikes

Yuping Liu¹, Yuzhen Zhao^{2*}, and Xiyu Liu³

Shandong Normal University, Jinan, Shandong 250014, China
zhaoyuzhen@sdu.edu.cn

Abstract. The spiking neural P systems with polarizations (PSN P systems) take neuron-associated polarizations as the firing condition of neuron rules, effectively simulating the biological fact that each neuron contains electrical charges. However, in biological neural systems, neurons are always connected by multiple synapses. Therefore, in this study, the weighted synapses and anti-spikes are introduced into the PSN P systems and get a novel variant of SN P systems called weighted spiking neural P systems with polarizations and anti-spikes (PAWSN P systems), in which the application of anti-spikes can further optimize the computational performance of the systems. We investigate the computational universality of PAWSN P systems. It is proved that PAWSN P systems are Turing universal as number generating devices. By comparing several SN P systems with polarizations, it can be found that the PAWSN P systems can achieve computational universality using fewer resources.

Keywords: membrane computing; spiking neural P systems; computational universality; weight; anti-spike; polarization

1 Introduction

Membrane systems, also known as P systems, are a class of distributed parallel computational models that can be divided into three main types according to the cell membrane structure or cell distribution: cell-like P systems, tissue-like P systems, and neural-like P systems. In the study of membrane computing theory, various types of P systems have been extended in the past studies and many extended P systems with computational universality have been obtained [1–6]. The computational complexity of different extended P systems has also been investigated [7–12].

The more widely studied are the spiking neural P systems (SN P systems) recently, in which information is transmitted between neurons using spikes. Based on different biological characteristics, SN P systems have been extended in terms of objects, rules, system structures and operation methods. In terms of system objects extension, Aman et al. [13] proposed the spiking neuron P systems with astrocytes of producing calcium. In terms of rules extension, Peng et al. [14] proposed dynamic threshold neural P systems (DTNP systems) using a dynamic threshold mechanism to control neurons firing. Wu et al. [15] introduced a novel spikes distribution mechanism in the rules of SN P systems and investigated the computational power. In terms of system structures extension, Yang et al. [16] proposed the spiking neural P systems with structural plasticity and anti-spikes (SNP-SPA systems). In terms of systems operation methods extension, Lv et al. [17] proposed the spiking neural P systems with extended channels (SNP-ECR

* Corresponding author: Yuzhen Zhao

systems). Song et al. [18] proposed the spiking neural P systems with delay on synapses (SNP-DS systems). Garcia et al. [19] proposed the spiking neural P systems with dendritic and axonal computation (DASN P systems). Lv et al. [20] investigated the Turing universality of sequential coupled neural (SCNP) systems working under four sequential strategies.

The spiking neural P systems with polarizations were proposed by Wu et al. [21], in which the firing condition for neurons is no longer a regular expression judging the number of spikes contained in neurons, but the neuron-associated polarizations: $+, 0, -$. Then Wu et al. [22] proposed the spiking neural P system with polarizations and anti-spikes by introducing anti-spikes in the SNP systems with polarizations for simplification. Jiang et al. [23] proposed the spiking neural P systems with polarizations and rules on synapses.

Based on the following motivation, we propose a novel variant of the SN P systems that is designed to more closely match the characteristics of biological neural systems.

- In biological nervous system, each neuron contains a certain electrical charge (either positive or negative) to maintain its current potential. The firing behavior of a neuron depends on its potential change, which in turn transmits electrical signals via synapses for the exchange of information between neurons. Therefore, the design of the spiking neural systems with polarizations [21] makes good use of the feature that all neurons contain a certain amount of charge.
- Synaptic connections between neurons build bridges for neuronal information exchange and transmission. In fact, the number of synapses present between neurons in brain often varies and multiple synapses exist between two neurons. Then, Pan et al. [24] achieved a simulation of this biological observation by assigning weights to the synapses and proposed the spiking neural P systems with weighted synapses.

Considering the above motivations, we introduce polarizations and weighted synapses into the SN P systems, which makes our computational model more consistent to with biological facts. In addition, we introduce anti-spikes to simplify the computation and improve the computational performance of the system. As a result, we obtain a novel variant of SN P systems called weighted spiking neural P systems with polarizations and anti-spikes (PAWSN P systems). In the PAWSN P systems, the application of weighted synapses can cause multiple spikes and anti-spikes with multiple charges to be sent between neurons. Therefore, it becomes necessary to perform charge calculations inside the neuron to determine the polarization of the neuron. Based on the above idea, this paper investigates the computational universality. It can be demonstrated that PAWSN P systems are Turing-universal as number generating devices. The rest of this paper is organized as follows. Section 2 presents a formal definition of PAWSN P systems and details how it works with an example. Section 3 demonstrates the Turing universality of the PAWSN P systems as number generating devices. Section 4 presents conclusions and outlook.

2 Weighted spiking neural P systems with polarizations and anti-spikes

A PAWSN P system containing $m \geq 1$ neurons can be expressed as a tuple:

$$\Pi = (O, \sigma_1, \dots, \sigma_m, syn, in, out)$$

(1) $O = \{a, \bar{a}\}$ is an alphabet consisting of two types of symbols, where a and \bar{a} represent spike and anti-spike, respectively.

(2) $\sigma_1, \dots, \sigma_m$ are neurons in system Π , and every neuron can be expressed as a tuple: $\sigma_i = (\alpha_i, c_i, n_i, R_i)$, $1 \leq i \leq m$.

(a) $\alpha_i \in \{+, 0, -\}$ is the initial polarization of neuron σ_i , and $+, 0, -$ represent positive, neutral, and negative polarizations, respectively.

(b) $c_i \geq 0$ is the number of charges contained in neuron σ_i . If $\alpha_i = +$, it means that the neuron σ_i contains c_i positive charges. If $\alpha_i = 0$, then $c_i = 0$, it means that the neuron σ_i contains no charge. If $\alpha_i = -$, it means that the neuron σ_i contains c_i negative charges.

(c) n_i is the number of initial spikes or anti-spikes contained in neuron σ_i . If $n_i > 0$, it means that the neuron σ_i contains n_i spikes. If $n_i = 0$, it means that the neuron σ_i contains no spike or anti-spike. If $n_i < 0$, it means that the neuron σ_i contains n_i anti-spikes.

(d) R_i is the set of rules contained in neuron σ_i , with of in the following three forms:
 i. Spiking rule: $\alpha/b^c \rightarrow b'; \beta$, where $\alpha, \beta \in \{+, 0, -\}, b, b' \in O, c \geq 1$;
 ii. Forgetting rule: $\alpha'/b^s \rightarrow \lambda; \beta'$, where $\alpha', \beta' \in \{+, 0, -\}, b \in O, s \geq 1$;
 iii. Annihilating rule: $a\bar{a} \rightarrow \lambda$. The execution of this rule takes precedence over the above spiking rule and forgetting rule.

(3) $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\} \times W$ represents the synapses between neurons, where $W = \{w | w \in N_+\}$ is the set of weights, and N_+ is the set of positive integers. For any synapse $(i, j, w) \in syn$, $1 \leq i \neq j \leq m, w \in W$.

(4) in, out are the input and output neurons, respectively.

The following conventions are given for the calculation of the number of charges in neurons:

i. Multiple positive (negative) charges that appear in the same neuron can be accumulated.

ii The same number of positive and negative charges in the same neuron can cancel each other out.

Note that the calculation of the number of charges contained in neuron σ_i is only used to determine the polarization of the neuron σ_i . Positive charges and negative charges cannot coexist in the same neuron. We can determine the polarization of a neuron by comparing the number of positive charges in the neuron with the number of negative charges. For example, if the number of positive charges in neuron σ_i is more than the number of negative charges, the polarization of neuron σ_i is positive after charge cancellation.

In PAWSN P systems, every neuron contains one or more rules. For the spiking rule: $\alpha/b^c \rightarrow b'; \beta$, when the polarization of neuron σ_i is α , i.e., $\alpha_i = \alpha$, and it contains at least c spikes (if b is a spike) or c anti-spikes (if b is an anti-spike), this rule can be applied to consume c spikes or anti-spikes in neuron σ_i to generate a spike or anti-spike and a charge β , then through the weighted synapse (i, j, w) , w spikes or anti-spikes with w charges are sent to subsequent neurons. For the forgetting rule: $\alpha'/b^s \rightarrow \lambda; \beta'$, when $\alpha_i = \alpha'$, and σ_i contains at least s spikes (if b is a spike) or s anti-spikes (if b is an anti-spike), this rule can be applied to consume s spikes or anti-spikes, then through the weighted synapse (i, j, w) , w charges are sent to subsequent neurons. For

the annihilating rule: $a\bar{a} \rightarrow \lambda$, when the same number of spikes and anti-spikes meet in the same neuron σ_i , they can cancel each other out. In addition, the execution of this rule does not need time and is not controlled by neuron-associated polarization. According to our system design, it makes sense to receive spikes from neuron σ_{out} , so that the environment will not receive anti-spikes and charges.

The configuration of the system Π at a certain time t can be expressed as $C_t = \langle (\alpha_1, c_1)/n_1, (\alpha_2, c_2)/n_2, \dots, (\alpha_m, c_m)/n_m \rangle$. The computation of the system is the transition of the system configuration. Starting from the initial configuration C_0 , the neurons can apply three types of rules to achieve transition, which is denoted by $C_0 \Rightarrow C_1 \Rightarrow \dots \Rightarrow C_h$. When all the neurons in the system cannot apply any rules, the computation stops.

3 Computational universality of PAWSN P systems

This section investigates the Turing Universality of PAWSN P systems as number generating devices. By simulating the register machines, it is shown that all recursively enumerable number sets can be generated the PAWSN P systems.

When the register machine M works in generation mode, initially, each register of M is empty, and the machine executes the instruction l_0 to start calculation. Eventually, the register machine M runs to the halt instruction l_h , at which point the number n stored in register 1 is the number generated by M . The set of numbers generated by M is denoted by $N(M)$ and the family of all sets of numbers generated by register machines is equal to the family NRE of recursively enumerable sets of numbers. The set of numbers to be generated by the PAWSN P systems using at most three kinds of polarization $p \leq 3$, with m neurons, and containing at most n rules per neuron is denoted as $N_2PAWSNP_m^n(ch_p)$. If there is no restriction on the number of neurons and the maximum number of rules in each neuron, the symbol $*$ is used instead.

Theorem 1. $N_2SNP_*^2(ch_3) = NRE$

Proof. According to the Turing-Church thesis, the inclusion $N_2SNP_*^2(ch_3) \subseteq NRE$ holds. We only need to prove the inclusion $NRE \subseteq N_2SNP_*^2(ch_3)$. Therefore, considering the register machine $M = (m, H, l_0, l_h, I)$ working in the generating mode, a PAWSN P system Π_1 is designed to simulate M . Three type of modules are included: the ADD module, the SUB module and the FIN module.

In order to interrelate the system Π_1 with the register machine M , we set each register r in M to correspond to a neuron σ_r in Π_1 , and the value in register r corresponds to the same number of spikes in the neuron. Similarly, each instruction l_i corresponds to a neuron σ_{l_i} in system Π_1 , and some auxiliary neurons σ_{c_i} ($i = 1, 2, 3, \dots$) are introduced.

Initially, each neuron in the system Π_1 is provided with a polarization and the number of charges contained. We denote the state of the neuron in diagrams as $(\sigma_i, (\alpha_i, n_i))$, where σ_i is the label of the neuron, α_i represents the polarization and n_i represents the number of charges contained in the neuron. In addition, the rest of the neurons in the system are empty, except for the neuron σ_{l_0} containing one spike, which is used to trigger the simulation. During the simulation, once the neuron σ_{l_i} receives one spike, the system starts the simulation instruction $l_i : (OP(r), l_j, l_k)$ (OP represents the ADD operation or the SUB operation). When the neuron σ_{l_h} receives one spike, it means

that the system successfully simulates the calculation in register machine M . The result is output by the neuron σ_{out} in FIN module. The result $t_2 - t_1$ is defined as the time interval between the first two spikes sent by the neuron σ_{out} to the environment, corresponding to the number contained in register 1.

In order to verify the system Π_1 can correctly simulate the calculations of the register machine M , we have listed the simulation process of each module in the table.

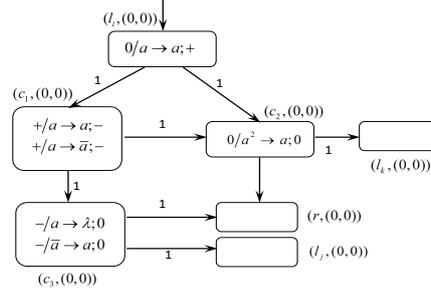


Fig. 1. ADD module in Π_1 to simulate instruction $l_i : (ADD(r), l_j, l_k)$.

(1) ADD module (as shown in Figure 1.): simulating an ADD instruction $l_i : (ADD(r), l_j, l_k)$.

Table 1. ADD instruction simulation process: case (1)

Step	Firing neurons	Rules executing	Observations
t	$l_i, (0, 0), a$	$0/a \rightarrow a; +$	the simulation of the ADD module begins
$t + 1$	$c_1, (+, 1), a$	$+/a \rightarrow a; -$	this simulation branch continues
$t + 2$	$c_2, (0, 0), a^2$	$0/a^2 \rightarrow a; 0$	simulates the increase of the number in register r by 1
	$c_1, (-, 1), a$	$-/a \rightarrow \lambda; 0$	activates the module associated with neuron σ_{l_k}

Table 2. ADD instruction simulation process: case (2)

Step	Firing neurons	Rules executing	Observations
t	$l_i, (0, 0), a$	$0/a \rightarrow a; +$	the simulation of the ADD module begins
$t + 1$	$c_1, (+, 1), a$	$+/a \rightarrow \bar{a}; -$	$+/a \rightarrow \bar{a}; -$
$t + 2$	$c_2, (0, 0), a$	$a\bar{a} \rightarrow \lambda$	restores neuron σ_{c_2} to be neutral and empty
	$c_3, (-, 1), \bar{a}$	$-/\bar{a} \rightarrow a; 0$	simulates the increase of the number in register r by 1 and activates the module associated with neuron σ_{l_j}

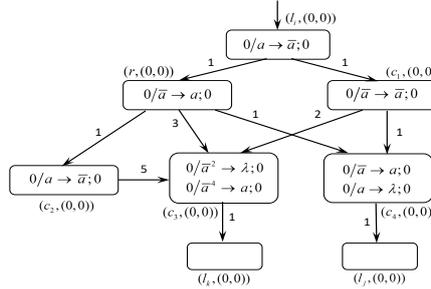


Fig. 2. SUB module in Π_1 to simulate instruction $l_i : (SUB(r), l_j, l_k)$.

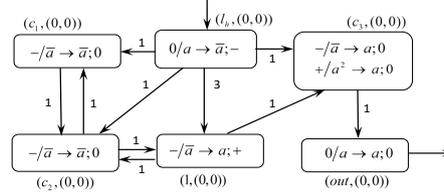
(2) SUB module (as shown in Figure 2.): simulating a SUB instruction $l_i : (SUB(r), l_j, l_k)$.

Table 3. SUB instruction simulation process: $n > 0$

Step	Firing neurons	Rules executing	Observations
t	$l_i, (0, 0), a$	$0/a \rightarrow \bar{a}; 0$	the simulation of the SUB module begins
$t + 1$	$r, (0, 0), a^{n-1}$ $c_1, (0, 0), \bar{a}$	$a\bar{a} \rightarrow \lambda$ $0/\bar{a} \rightarrow \bar{a}; 0$	$n > 0$ and simulates the reduction of the number in register r by 1 this simulation branch continues
$t + 2$	$c_3, (0, 0), \bar{a}^2$ $c_4, (0, 0), \bar{a}$	$0/\bar{a}^2 \rightarrow \lambda; 0$ $0/\bar{a} \rightarrow a; 0$	restores neuron σ_{c_3} to be neutral and empty activates the module associated with neuron σ_{l_j}

Table 4. SUB instruction simulation process: $n = 0$

Step	Firing neurons	Rules executing	Observations
t	$l_i, (0, 0), a$	$0/a \rightarrow \bar{a}; 0$	the simulation of the SUB module begins
$t + 1$	$r, (0, 0), \bar{a}$ $c_1, (0, 0), \bar{a}$	$0/\bar{a} \rightarrow a; 0$ $0/\bar{a} \rightarrow \bar{a}; 0$	$n = 0$ and this execution continues this simulation branch continues
$t + 2$	$c_2, (0, 0), a$ $c_4, (0, 0), 0$	$0/a \rightarrow \bar{a}; 0$ $a\bar{a} \rightarrow \lambda$	this simulation branch continues restores neuron σ_{c_4} to be neutral and empty
$t + 3$	$c_3, (0, 0), \bar{a}^4$	$0/\bar{a}^4 \rightarrow a; 0$	activates the module associated with neuron σ_{l_k}


Fig. 3. FIN module of the system II_1 .

(3)FIN module (as shown in Figure 3): outputting the result of the computation.

Table 5. FIN instruction simulation process

Step	Firing neurons	Rules executing	Observations
t	$l_h, (0, 0), a$	$0/a \rightarrow \bar{a}; -$	the simulation of the FIN module begins
$t + 1$	$c_1, (-, 1), \bar{a}$ $c_2, (-, 1), \bar{a}$ $1, (-, 3), a^{n-3}$ $c_3, (-, 1), \bar{a}$	$-\bar{a} \rightarrow \bar{a}; 0$ $-\bar{a} \rightarrow \bar{a}; 0$ $a\bar{a} \rightarrow \lambda$ $-\bar{a} \rightarrow a; 0$	this simulation branch continues the number in register r is decreased by 3 this simulation branch continues
$t + 2$	$c_1, (-, 1), \bar{a}$ $c_2, (-, 1), \bar{a}$ $1, (-, 3), a^{n-4}$ $out, (0, 0), a$	$-\bar{a} \rightarrow \bar{a}; 0$ $-\bar{a} \rightarrow \bar{a}; 0$ $a\bar{a} \rightarrow \lambda$ $0/a \rightarrow a; 0$	this simulation branch continues the number in register r is decreased by 1 sends the first spike to environment
...
$t + n - 1$	$c_1, (-, 1), \bar{a}$ $c_2, (-, 1), \bar{a}$ $1, (-, 3), \bar{a}$	$-\bar{a} \rightarrow \bar{a}; 0$ $-\bar{a} \rightarrow \bar{a}; 0$ $-\bar{a} \rightarrow a; +$	this simulation branch continues this simulation branch continues
$t + n$	$c_1, (-, 1), \bar{a}$ $c_2, (0, 0), 0$ $1, (-, 3), \bar{a}$	$-\bar{a} \rightarrow \bar{a}; 0$ $a\bar{a} \rightarrow \lambda$ $-\bar{a} \rightarrow a; +$	restores neuron σ_{c_1} to be empty restores neuron σ_{c_2} to be empty restores neuron σ_1 to be empty
$t + n + 1$	$c_2, (+, 1), 0$ $c_3, (+, 1), a^2$	$a\bar{a} \rightarrow \lambda$ $+/\bar{a}^2 \rightarrow a; 0$	restores neuron σ_{c_2} to be empty this simulation branch continues
$t + n + 2$	$out, (0, 0), a$	$0/a \rightarrow a; 0$	sends the second spike to environment

In summary, the analysis of the simulation process in tables leads to the conclusion that the system II_1 correctly simulates the work of the register machine M in the gen-

erating mode. The system Π_1 applies three types of polarizations, and each neuron in the system contains at most two rules. Therefore, Theorem 1 holds.

Table 6. Comparison of the extended SN P systems with polarizations as number generating devices

Computing models	Configuration				
	Polarizations	Rules	Auxiliary neurons (ADD)	Auxiliary neurons (SUB)	Auxiliary neurons (FIN)
PAWSN P systems	3	2	3	4	3
PSN P systems[21]	3	2	5	5	7
PASN P systems[22]	2	2	4	6	6
PANRS P systems[23]	3	2	7	8	5

Table 6 gives the comparison of four kinds of extended SN P systems with polarizations working in the generating mode, mainly comparing the number of types of polarizations used, the maximum number of rules contained in neurons or on the synapses (for PSRS P systems), and the number of auxiliary neurons used in the every module. The comparison shows that the PAWSN P systems use the least number of auxiliary neurons for each module. Therefore, PAWSN P systems can achieve computational universality using fewer computational resources.

4 Conclusion

Based on the biological phenomenon that neurons contain a certain number of charges and the existence of multiple synapses between neurons, we introduce the weighted synapses and anti-spikes into the spiking neural P systems with polarizations and propose a variant of SN P systems called weighted spiking neural P systems with polarizations and anti-spikes (PAWSN P systems). The application of weighted synapses has triggered the discussion of the transmission of charges between neurons. At the same time, the calculation of the internal charges of neurons has an important role in the judgment of neuronal polarizations and the execution of rules. In this paper, we give the formal definition of the PAWSNP system. Then the computational universality of PAWSN P system as number generating devices is demonstrated. By comparing different variants of SN P systems with polarizations, it can be found that the PAWSN P systems can achieve computational universality using fewer neuronal resources.

The next research could focus on further reducing the number of polarization types and the number of neurons required for PAWSN P systems with computational universality. In addition, applying the PAWSN P systems to solve NP problems is a research direction.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (No.61806114, 61472231, 61876101, 61602282, 61402187, 61502283, 61802234, 61703251), the China Postdoctoral Science Foundation (No.2018M642695, 2019T120607).

References

1. Jiang, S.X., Wang, Y.F., Xu, J.B., Xu, F.: The Computational Power of Cell-like P Systems with Symport/Antiport Rules and Promoters. *Fundamenta Informaticae* 164(2-3), 207–225 (2019)

2. Liu, L., Yi, W., Yang, Q., Peng, H., Wang, J.: Small Universal Numerical P Systems with Thresholds for Computing Functions. *Fundamenta Informaticae* 176(1), 43–59 (2020)
3. Pan, L.Q., Song, B.S.: P Systems with Rule Production and Removal. *Fundamenta Informaticae* 171(1-4), 313–329 (2020)
4. Liu, L., Yi, W., Yang, Q., Peng, H., Wang, J.: Numerical P systems with Boolean condition. *Theoretical Computer Science* 785, 140–149 (2019)
5. Jiang, S.X., Wang, Y.F., Xu, F., Deng, J.L.: Communication P Systems with Channel States Working in Flat Maximally Parallel Manner. *Fundamenta Informaticae* 168(1), 1–24 (2019)
6. Luo, Y., Zhao, Y., Chen, C.: Homeostasis Tissue-Like P Systems. *IEEE Trans Nanobiotechnology* 20(1), 126–136 (2021)
7. Jiang, S.X., Wang, Y.F., Su, Y.S.: A uniform solution to SAT problem by symport/antiport P systems with channel states and membrane division. *Soft Computing* 23(12), 3903–3911 (2019)
8. Ye, L., Zheng, J.H., Guo, P., Perez-Jimenez, M.J.: Solving the 0-1 Knapsack Problem by Using Tissue P System With Cell Division. *Ieee Access* 7, 66055–66067 (2019)
9. Song, B.S., Zeng, X.X., Rodriguez-Paton, A.: Monodirectional tissue P systems with channel states. *Information Sciences* 546, 206–219 (2021)
10. Christinal, A.H., Diaz-Pernil, D., Mathu, T.: A uniform family of tissue P systems with protein on cells solving 3-coloring in linear time. *Natural Computing* 17(2), 311–319 (2018)
11. Buno, K.C., Cabarle, F.G.C., Calabia, M.D., Adorna, H.N.: Solving the N-Queens problem using dP systems with active membranes. *Theoretical Computer Science* 736, 1–14 (2018)
12. Guo, P., Zhu, J., Chen, H.Z., Yang, R.L.: A Linear-Time Solution for All-SAT Problem Based on P System. *Chinese Journal of Electronics* 27(2), 367–373 (2018)
13. Aman, B., Ciobanu, G.: Spiking Neural P Systems with Astrocytes Producing Calcium. *International Journal of Neural Systems* 30(12) (2020)
14. Peng, H., Wang, J., Perez-Jimenez, M.J., Riscos-Nunez, A.: Dynamic threshold neural P systems. *Knowledge-Based Systems* 163, 875–884 (2019)
15. Wu, T.F., Zhang, L.P., Pan, L.Q.: Spiking neural P systems with target indications. *Theoretical Computer Science* 862, 250–261 (2021)
16. Yang, Q., Li, B., Huang, Y., Peng, H., Wang, J.: Spiking neural P systems with structural plasticity and anti-spikes. *Theoretical Computer Science* 801, 143–156 (2020)
17. Lv, Z., Bao, T., Zhou, N., Peng, H., Huang, X., Riscos-Nunez, A., Perez-Jimenez, M.J.: Spiking Neural P Systems with Extended Channel Rules. *International Journal of Neural Systems* 31(1) (2021)
18. Song, X., Valencia-Cabrera, L., Peng, H., Wang, J., Perez-Jimenez, M.J.: Spiking Neural P Systems with Delay on Synapses. *International Journal of Neural Systems* 31(1) (2021)
19. Garcia, L., Sanchez, G., Vazquez, E., Avalos, G., Anides, E., Nakano, M., Sanchez, G., Perez, H.: Small universal spiking neural P systems with dendritic/axonal delays and dendritic trunk/feedback. *Neural networks : the official journal of the International Neural Network Society* 138, 126–139 (2021)
20. Lv, Z.Q., Kou, J.H., Yi, W.M., Peng, H., Song, X.O.X., Wang, J.: Sequential Coupled Neural P Systems. *International Journal of Unconventional Computing* 15(3), 157–191 (2020)
21. Wu, T., Paun, A., Zhang, Z., Pan, L.: Spiking Neural P Systems With Polarizations. *IEEE transactions on neural networks and learning systems* 29(8), 3349–3360 (2018)
22. Wu, T., Zhang, T., Xu, F.: Simplified and yet Turing universal spiking neural P systems with polarizations optimized by anti-spikes. *Neurocomputing* 414, 255–266 (2020)
23. Jiang, S., Fan, J., Liu, Y., Wang, Y., Xu, F.: Spiking Neural P Systems with Polarizations and Rules on Synapses. *Complexity* 2020 (2020)
24. Pan, L.Q., Zeng, X.X., Zhang, X.Y., Jiang, Y.: Spiking Neural P Systems with Weighted Synapses. *Neural Processing Letters* 35(1), 13–27 (2012)

Domino Recognizable Three Dimensional Picture Languages and P System

F. Sweety¹, S. Annadurai¹, T. Kalyani^{2*}, D.G. Thomas³, and K. Bhuvaneswari⁴

¹ Department of Mathematics, St. Joseph's College of Engineering, Chennai-119, India

² Department of Mathematics, St. Joseph's Institute of Technology, Chennai - 600119, India
kalphd02@yahoo.com

³ Department of Science and Humanities (Mathematics Division), Saveetha School of Engineering, Chennai - 602105, India

⁴ Department of Mathematics, Sathayabama Institute of Science and Technology, Chennai - 600119, India

Abstract. In two dimensions, many families like rectangular, hexagonal and iso triangular picture languages were proposed. Tiling a plane in two dimensions play a vital role in image processing and pattern matching. Three dimensional tetrahedral picture languages were introduced and its properties were studied. Here we propose recognizability of tetrahedral picture languages using dominoes and it is compared with local and recognizable tetrahedral picture languages. Further, we propose a tetrahedral $v \rightarrow e_p$ and $e_p \rightarrow v$ pasting system and pasting P system and these two systems are compared with local tetrahedral picture languages.

Keywords: Dominoes, P system, pasting system, picture languages

1 Introduction

Generation or recognition of two dimensional languages by several approaches like pattern recognition, pattern matching, etc. have been proposed in the literature [1, 3–5]. A two dimensional language is a set of pictures. Dora Giammarresi and Antonio Restivo [2] have proposed two-dimensional picture languages and its recognizability through local picture languages. M. Latteux and Simplot [6] have proposed hv -local picture languages using horizontal and vertical dominoes.

Extending these ideas hexagonal picture languages, iso-picture languages and octagonal picture languages were introduced [1, 3, 7]. The idea of recognizability in two-dimensions is extended to three-dimensional picture languages namely 3D-rectangular picture languages and tetrahedral picture languages in [8–10].

On the other hand, in the area of membrane computing, Gh. Paun introduced P system [13] which is a new computability model. This model computes patterns to be generated by the system by processing the multi sets of objects present in the region that are defined by a hierarchial arrangement of membranes, by evolution rule associated with the region.

These two areas, namely membrane computing and picture grammars were linked together by Ceterchi et.al. [15], by relating P systems and array-rewriting grammars

* Corresponding author.

generating picture languages and proposed array-rewriting P systems [16]. Pasting system and its variants like extended pasting system and parametric P system were introduced in [11, 12, 14].

In this paper domino recognizability of tetrahedral picture languages is introduced and it is compared with local and recognizable tetrahedral picture languages. Further, we propose a tetrahedral $v \rightarrow e_p$ and $e_p \rightarrow v$ pasting system and pasting P system and these two systems are compared with local tetrahedral picture languages.

2 Preliminaries

We refer [8] for the definitions of tetrahedral tile and tetrahedral picture languages. Here we refer [9] for the definitions of local and recognizable tetrahedral picture languages.

3 Domino Recognizable Tetrahedral Picture Languages over $v \rightarrow e_p$

In this section, we define vertical, right and left dominoes of the types shown in Fig. 5.

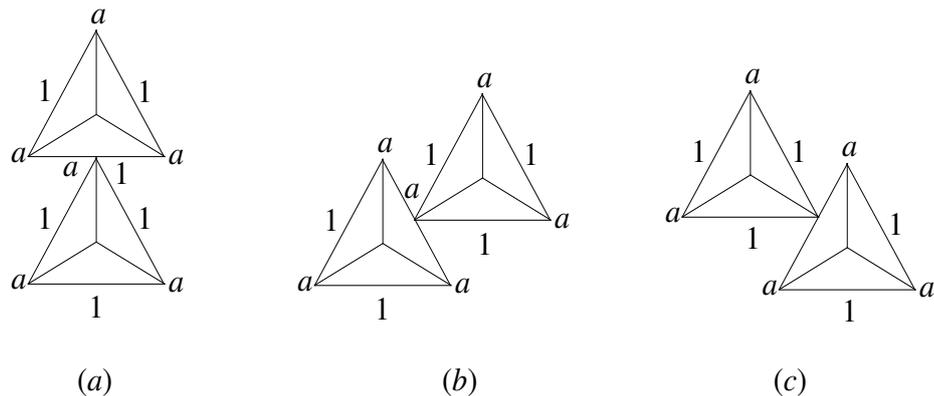


Fig. 1. Vertical, right and left dominoes of size (2, 1)

We have defined local tetrahedral picture languages as languages given by a finite set of authorized tetrahedral pictures of size 2. Here, vertical, right and left controls are done at the same time. In domino system the three controls are done separately.

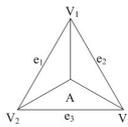
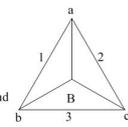
Given a tetrahedral picture p of size n , we denote by $B_{h,k}(p)$, the set of all sub pictures of p of size (h, k) , i.e., h tetrahedral pictures of size k , $1 \leq h, k \leq n$ are generated by the production rule $v \rightarrow e_p$.

Definition 1. A three dimensional tetrahedral picture language $L \subseteq S^{**T}$ is *vrl-local* if there exists a finite set Δ of dominoes over the alphabet $\Sigma \cup \{\#\}$ such that the

4 Tetrahedral Tile $v \rightarrow e_p$ and $e_p \rightarrow v$ Pasting System and P System

In this section, we propose $v \rightarrow e_p$ and $e_p \rightarrow v$ pasting rules, pasting system and P system. We compare Tetrahedral tile $v \rightarrow e_p$ and $e_p \rightarrow v$ pasting system and P system with local tetrahedral picture languages.

Definition 3. A $v \rightarrow e_p$ pasting rule is a pair (x, y) , where x belongs to the vertex set of tetrahedral tile and y belongs to the edge positions of tetrahedral tiles. The pasting rule (x, y) glues the vertex of a base of a tetrahedral tile with the midpoint of edge of a base of another tetrahedral tile.

Example 1. Consider the two tetrahedral tiles  and . The $v \rightarrow e_p$

pasting rules by which the two tetrahedral tiles A and B can be glued are $\{(v_1, 3), (v_2, 2), (v_3, 1)\}$.

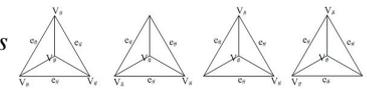
Similarly the tetrahedral tile A can be glued with A by $v \rightarrow e_p$ pasting rules $\{(v_1, e_3), (v_2, e_2), (v_3, e_1)\}$.

The tetrahedral tile B can be glued with A by $v \rightarrow e_p$ pasting rules $\{(a, e_3), (b, e_2), (c, e_1)\}$. The tetrahedral tile B can be glued with B by $v \rightarrow e_p$ pasting rules $\{(a, 3), (b, 2), (c, 1)\}$.

Definition 4. A $e_p \rightarrow v$ pasting rule is a pair (x, y) , where x belongs to the set of edge positions of tetrahedral tiles and y belongs to the set of vertices of a tetrahedral tiles. The pasting rule (x, y) glues the midpoint of edge of a tetrahedral tile with the vertex of another tetrahedral tile.

Example 2. For the two tetrahedral tiles A and B given in Example 4, the $e_p \rightarrow v$ pasting rules for the combinations of tiles A and A , A and B , B and A and B and B are as follows:

- (i) $\{(e_1, v_3), (e_2, v_2), (e_3, v_1)\}$
- (ii) $\{(e_1, c), (e_2, b), (e_3, a)\}$
- (iii) $\{(1, v_3), (2, v_2), (3, v_1)\}$
- (iv) $\{(1, c), (2, b), (3, a)\}$

Definition 5. The tetrahedral tiles A and B can be glued with the special tetrahedral boundary tiles  by the following $v \rightarrow e_p$ and

$e_p \rightarrow v$ pasting rules:

- (i) $\{(v_1, e_{\#}), (v_2, e_{\#}), (v_3, e_{\#})\}$
- (ii) $\{(a, e_{\#}), (b, e_{\#}), (c, e_{\#})\}$
- (iii) $\{(e_1, v_{\#}), (e_2, v_{\#}), (e_3, v_{\#})\}$
- (iv) $\{(1, v_{\#}), (2, v_{\#}), (3, v_{\#})\}$

Definition 6. A controlled tabled tetrahedral $v \rightarrow e_p$ & $e_p \rightarrow v$ tile pasting system ($CTTv \rightarrow e_p$ & $e_p \rightarrow vTPS$) is a 5-tuple $M = (\Sigma, V, E, P, c, t_0)$ where Σ is a finite set of tetrahedral tiles, V is a set of vertex labels of base of tetrahedral tiles in Σ , E is a set of edge labels of base of tetrahedral tiles in Σ . P is a finite set of tables $\{T_1, T_2, \dots, T_k\}$, $k \geq 1$ and each T_i , $i = 1, 2, \dots, k$ is $v \rightarrow e_p$ or $e_p \rightarrow v$ pasting rules. The rules of the tables are applied in parallel to the respective vertices or edges of the pattern derived in the previous stage. C is a control language over P and $t_0 \in \Sigma^{++}$ is the axiom pattern.

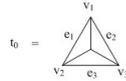
A pattern t_j is generated from t_0 by applying the control word $w = w_1w_2 \dots w_j \in C$ where $w_i \in C$ ($1 \leq i \leq j$) if there exists a sequence of derivations $t_0 \Rightarrow_{w_1} t_1 \Rightarrow_{w_2} t_2 \Rightarrow \dots \Rightarrow_{w_j} t_j$ and is denoted by $t_0 \xRightarrow{*}_w t_j$.

The set of all patterns generated by the system is $T(M) = \{t_j \in \Sigma^{++} : t_0 \xRightarrow{*}_c t_j / j \geq 0\}$. The control language C may be either regular, context-free or context-sensitive in which case we attach the corresponding control.

Example 3. Consider a $CTTv \rightarrow e_p$ & $e_p \rightarrow vTPS$ $M_1 = (\Sigma, V, E, P, C, t_0)$, where

$$\Sigma = \left\{ \begin{array}{c} v_1 \\ e_1 \quad e_2 \\ v_2 \quad e_3 \quad v_3 \end{array} \right\}, \quad V = \{v_1, v_2, v_3, a, b, c\}, \quad E = \{e_1, e_2, e_3, 1, 2, 3\},$$

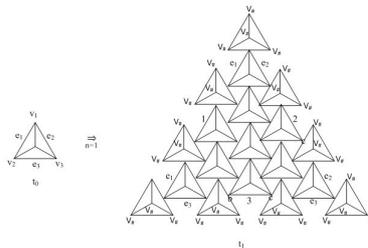
$$P = \{T_1, T_2, T_3, T_4, T_5\}, T_1 = \{(e_1, c), (e_2, b), (e_3, a)\}, T_2 = \{(v_1, e_3), (v_2, e_2), (v_3, e_1)\}, T_3 = \{(2, b), (1, c), (3, a)\}, T_4 = \{(v_3, e_{\#}), (v_2, e_{\#}), (v_1, e_{\#})\}, T_5 = \{(2, v_{\#}), (1, v_{\#}), (3, v_{\#})\}, C = \{(T_1T_2)^n T_4T_5 / n \geq 1\},$$



This system generates a tetrahedral picture language where the tetrahedral tiles

along the triads are  and the tetrahedral tiles in the remaining places are .

The first member t_1 by applying the control language C is shown in the following figure.



Theorem 2. The family of tetrahedral picture languages generated by $CTTv \rightarrow e_p$ & $e_p \rightarrow vTPS$ intersects the family of local tetrahedral picture languages $TrLOC(v \rightarrow e_p)$.

Proof. The language of tetrahedral pictures, where the tetrahedral tiles along the triad are  and the tetrahedral tiles in the remaining places are  is a local tetrahedral picture language as given in Example 2, which is also generated by $CTTv \rightarrow e_p$ & $e_p \rightarrow vTPS$ (Example 6).

Definition 7. A tetrahedral tile $v \rightarrow e_p$ & $e_p \rightarrow v$ pasting P system ($TetTv \rightarrow e_p$ & $e_p \rightarrow vPPS$) $\pi = (\Sigma, \mu, F_1, F_2, \dots, F_m, R_1, R_2, \dots, R_m, i_0)$ where Σ is a finite set of labeled tetrahedral tiles, μ is a membrane structure with ‘ m ’ membranes, labeled in an one-to-one way with $1, 2, \dots, m$; F_1, F_2, \dots, F_m are finite sets of three-dimensional tetrahedral picture patterns over tiles of Σ associated with the m regions of μ . R_1, R_2, \dots, R_m are finite sets of $v \rightarrow e_p$ and $e_p \rightarrow v$ pasting rules of the type $((x_i, y_i), tar)$, $1 \leq i \leq n$ associated with the m regions of μ and i_0 is the output membrane which is an elementary membrane.

The computation process is similar to the computation process of $TetTPPS$ [11].

Example 4. Consider a tetrahedral tile $v \rightarrow e_p$ & $e_p \rightarrow v$ pasting P system generating tetrahedral picture languages where the tetrahedral tiles along the triad are  and

the tetrahedral tiles in the remaining places are  .
 $\pi = (\Sigma, \mu = [1[2[3[4[5]4]3]2]1, F_1, F_2, F_3, F_4, F_5, R_1, R_2, R_3, R_4, R_5, 5)$

where $\Sigma = \left\{ \begin{array}{c} v_1 \\ e_1 \quad e_2 \\ v_2 \quad e_3 \\ v_3 \end{array} \right\}, \left\{ \begin{array}{c} a \\ 1 \quad 2 \\ b \quad 3 \\ c \end{array} \right\}$

$$F_1 = \begin{array}{c} v_1 \\ e_1 \quad e_2 \\ v_2 \quad e_3 \\ v_3 \end{array}, F_2 = F_3 = F_4 = F_5 = \emptyset$$

$R_1 = \{((e_1, c), here), ((e_2, b), here), ((e_3, a), in)\}$
 $R_2 = \{((v_1, e_3), here), ((v_2, e_2), here), ((v_3, e_1), in), ((v_3, e_1), out)\}$
 $R_3 = \{((v_1, e_3), here), ((v_2, e_2), here), ((v_3, e_1), in)\}$
 $R_4 = \{((v_e, e_\#), here), ((v_2, e_\#), here), ((v_1, e_\#), in), ((v_1, e_\#), here), ((2, v_\#), here), ((1, v_\#), here), ((3, v_\#), in)\}$
 $R_5 = \emptyset.$

Beginning with the initial object F_1 in region 1, the pasting rule R_1 is applied, where the rules in R_1 are applied in parallel to the boundary edges of the pattern present in region 1. Once the rule $((e_3, a), in)$ is applied, the generated 3D-pattern is sent to the inner membrane 2, and in region 2, the rules of R_2 are applied in parallel to the vertices of the pattern generated in region 1. If the rule $((v_3, e_1), out)$ is applied, the 3D-picture pattern generated is sent to region 1 and rule R_1 is applied. Whereas if the rule $((v_3, e_1), in)$ is applied, the 3D-picture pattern generated is sent to the inner membrane 3, and in region 3, the rule R_3 is applied.

In region 3, if the rule $((v_3, e_1), in)$ is applied, the 3D-picture pattern generated is sent to the inner membrane 4, and in region 4, the boundary tiles are glued and once the rules $((v_1, e_{\#}), in)$ and $((3, v_{\#}), in)$ are applied the final tetrahedral picture pattern is sent to membrane 5, wherein the picture pattern is collected, which is the elementary membrane.

Theorem 3. *The family of tetrahedral picture languages generated by $TetTv \rightarrow e_p$ & $e_p \rightarrow vPPS$ intersects the family of tetrahedral picture languages generated by $CTTv \rightarrow e_p$ & $e_p \rightarrow vTPS$.*

Proof. It is evident from examples 6 and 7.

Theorem 4. $\mathcal{L}(TetTv \rightarrow e_p \ \& \ e_p \rightarrow vPPS) \cap TrLOC(v \rightarrow e_p) \neq \emptyset$.

Proof. It is evident from examples 2 and 6.

5 Conclusion

In this paper we have defined domino recognizability of tetrahedral picture languages over the production rule $(v \rightarrow e_p)$. It is compared with local and recognizable tetrahedral picture languages. We have proposed $CTTv \rightarrow e_p \ \& \ e_p \rightarrow vTPS$ and $TetTv \rightarrow e_p \ \& \ e_p \rightarrow vPPS$ and these two systems are compared with local tetrahedral picture languages. This work can also be applied to other production rules of PSFG. Wang recognizability of tetrahedral picture languages can be studied. This is our future work.

References

1. Dersanambika K S, Krithivasan K, Martin-Vide C and Subramanian K G 2005, Local and recognizable hexagonal picture languages, *International Journal of Pattern Recognition and Artificial Intelligence* **19** 853–871
2. Giammarresi D and Restivo A 1992, Recognizable picture languages, *International Journal of Pattern Recognition and Artificial Intelligence* **6** 231–256
3. Kalyani T, Dare V R and Thomas D G 2004, Local and recognizable iso picture languages, *Lecture Notes in Computer Science* **3316** 738–743
4. Siromoney G, Siromoney R and Krithivasan K 1972, Abstract families of matrices and picture languages, *Computer Graphics and Image Processing* **1** 284–307
5. Siromoney G, Siromoney R and Krithivasan K 1973, Picture languages with array rewriting rules, *Information and Control* **22** 447–470
6. Latteux M and Simplot D 1997, Recognizable picture languages and domino tiling, *Theoretical Computer Science* **178** 275–283
7. Kuberal S, Kamaraj T, Kalyani T and Bhuvanewari K 2017, Octagonal tile rewriting grammar and picture languages, *International Journal of Computer & Mathematical Sciences* **6**(9)
8. Dharani A, Maragatham R S and Siromoney R 2017, Picture generation of rectangular blocks using tetrahedral, *Global Journal of Mathematical Sciences: Theory and Practical* **9**(2) 189–203

9. Sweety F, Kalyani T, Maragatham R S and Thomas D G 2019, Recognizability of tetrahedral picture languages, *International Journal of Recent Technology and Engineering* **8**(4) 7379–7383
10. Thomas D G, Sweety F, Dare V R and Kalyani T 2008, 3D rectangular array acceptors and learning, *Image Analysis from Theory to Applications* 205–214
11. T. Kalyani, T.T. Raman, D. Gnanaraj Thomas, Tetrahedral tile pasting P system for 3D patterns, *Mathematics in Engineering, Science and Aerospace*, Vol 11, No 1, 255–263, 2020.
12. T. Kalyani, K. Sasikala, V.R., Dare, T. Robnison, Triangular Pasting System In : K.G. Subramanian, K. Rangarajan, M. Mukund (Eds.), *Formal Models, Languages and Applications*, Series in Machine Perception and Artificial Intelligence, Vol.66 (2006), 195-211.
13. Gh. Păun, *Computing with Membranes: An introduction*, Springer-Verlag, Berlin, 2002.
14. T. Robnison, Extended Pasting Scheme for Kolam Pattern generation, *Forma*, 22 (2007), 55-64.
15. R. Ceterchi, M. Mutyam, Gh. Paun and K.G. Subramanian, Array-rewriting P systems, *Natural Computing*, 2 (2003), 229–249.
16. K.G. Subramanian, R. Saravanan, T. Robnison P Systems for array generation and application to kolam Patterns, *Forma* 22 (2007) 47–54.

Short Papers: Nature-Inspired Computing

Prediction of Grain Temperature Based on Intelligent Algorithm Optimized BP Neural Network

Jing Yu¹[0000-0002-9075-1734], Shuo Liu¹[0000-0002-3724-0353]*, Tianqi Liu¹[0000-0001-5826-8702], Kang Zhou¹[0000-0002-0205-768X], Yan Zhang², Wenbin Ma³, and Pinpin Wu³

¹ Wuhan Polytechnic University, Wuhan, Hubei, 430000, China

² Zhixing College of Hubei University, Wuhan, Hubei, 430000, China

³ Wuhan National Rice Trading Center Co., Ltd, Wuhan, Hubei, 430000, China

Abstract. China is a country with a large population and a large consumption of grain. It is a very important event to ensure the long-term storage of grain in the granary. Grain temperature has an important impact on grain security. Effective monitoring and prediction of grain temperature is an important means to ensure grain quality. In this paper, neural network is used to predict grain temperature. In the process of experiment, in view of some defects of BP neural network in grain temperature prediction, genetic algorithm (GA), particle swarm optimization (PSO), improved genetic algorithm optimized BP neural network (IGA-BP) and improved particle swarm optimization algorithm optimized BP neural network (IPSO-BP) were put forward to predict grain temperature, so as to improve the prediction accuracy and speed. By comparing the prediction results of the five models, it is concluded that IPSO-BP model can better predict grain temperature.

Keywords: grain temperature prediction · BP neural network · intelligent optimization algorithm

1 Introduction

Food is a country's strategic material, is also the fundamental for human survival, food affects national security, political stability, economic development and social harmony, so the storage of food is a very important issue. In the long-term storage of grain, temperature is closely related to the phenomena such as heating of grain heap in the granary, grain mildew and microbial reproduction[1]. Effective monitoring and prediction of grain temperature is an important means to ensure grain quality. Since grain is a poor conductor of heat, it is difficult to directly measure the temperature of grain heap[2], and the temperature of grain heap can only be predicted by the temperature, humidity and other factors at different locations of grain barn.

After long-term research by domestic and foreign scholars, there are many methods for grain temperature prediction. As for the application status of numerical simulation in the research of grain storage, Abe T[3] et al. proposed a two-dimensional digital model based on the finite difference method and simulated the change of grain

* Shuo Liu:874477154@qq.com, Jing Yu:1294390279@qq.com

warehouse temperature with the external environment temperature during the whole storage period by numerical simulation. And the simulated data and real data can get a good fitting effect. Jian F[4] et al. monitored the temperature and moisture content in a metal silo. Through numerical simulation of the temperature in the metal silo, they confirmed that even in a small silo, there is a large enough temperature gradient to drive the air movement, and that convection may lead to moisture migration. Hammami F[5] et al. simulated the temperature of the grain when the air flow passes through the cylindrical granary, and the final experimental results were basically consistent with other open data. Zhang Qian[6] et al. monitored the temperature in the granary for a long time and recorded the data, and used quadratic function for fitting analysis to obtain the quantitative relationship between the granary temperature and grain temperature. Jin Libing[7] et al. adopted the method of combining underground simulation experiment, engineering experiment and numerical simulation. The change law of grain temperature in underground ecological granary was studied. It was found that underground ecological granary has the advantage of storing grain at constant low temperature, which is beneficial to guarantee grain quality. Wang Yuancheng[8] et al. conducted numerical simulation on the temperature and moisture of the grain pile in the Granary Based on the numerical simulation method. Through the experiment, it was found that the combination of oblique flow and transverse ventilation can better achieve the effect of cooling and moisture retention.

The research of the above scholars has made some achievements in the prediction of grain temperature. Due to the large number of parameters, large amount of calculation and long consumption of time in the prediction model, the prediction accuracy is not high. Therefore, in view of the above problems, this paper proposes BP neural network optimized based on swarm intelligence algorithm to predict grain temperature. The intelligent optimization algorithm and neural network are combined to build a grain heap temperature prediction system, improve the efficiency of grain temperature control, and ensure the grain quality safety in the granary.

2 Introduction of Algorithm

2.1 BP Neural Network

Since all the data sets processed in this experiment are of numerical type, the most basic BP neural network is selected for prediction[9]. Traditional BP neural network is composed of input layer, hidden layer and output layer, and the process is mainly divided into two stages. The first stage is forward propagation, The second stage is back propagation.

2.2 Intelligent Algorithm

As a part of evolutionary computing, Genetic Algorithm (GA) is a computational model simulating Darwin's Genetic selection and natural selection of biological evolution process[10-11], and is a method to search for the optimal solution by simulating natural evolution process. The algorithm is a way to convert the natural evolution process into mathematical language, and to convert the problem solving process into the

process of biological evolution by using computers. In a word, genetic algorithm is an elite strategy[12], mainly including selection operator, crossover operator and mutation operator, through which the optimal individual is selected, that is the optimal solution.

Particle Swarm optimization (PSO) was proposed by Dr. J.Kennedy and Dr. R.c. Eberhart in 1995[13-14]. Particle swarm optimization (PSO), similar to genetic algorithm (GA), is a bionic algorithm for global random search based on swarm intelligence. Particle swarm optimization (PSO) converges quickly and easily, and only a few parameters need to be adjusted. It does not need the complex operators such as selection, crossover and mutation in genetic algorithm, so it has been widely used in various neighborhoods.

The speed and position update formula are as follows:

$$V_{id} = V_{id} + c1 * r1 * (P_{id} - X_{id}) + c2 * r2 * (P_{gd} - X_{id}) \quad (1)$$

$$X_{id} = V_{id} + X_{id} \quad (2)$$

V_{id} represents the velocity of the particle, P_{id} represents the individual extreme value, P_{gd} represents the population extreme value of the population, X_{id} represents the position of the particle, $c1$ and $c2$ are learning factors and non-negative constants, $r1$ and $r2$ are random numbers in the range [0,1].

2.3 Improved Intelligent Algorithm

2.3.1 Improved Genetic Algorithm

In genetic algorithms, crossover probability and mutation probability are important factors that determine algorithm performance. In the evolutionary process, at the beginning of the iteration, in order to expand the search range, the crossover probability of the population should be larger. In the late iteration, in order to avoid falling into local optimality, a greater probability of variation should be used to generate new individuals[15]. Therefore, on the basis of the traditional genetic algorithm, this experiment sets an adaptive adjustment mechanism for the crossover probability and mutation probability respectively. During the experiment, the crossover probability and mutation probability can be adjusted dynamically to improve the search efficiency of the algorithm.

The adaptive cross probability formula is established as follows:

$$pcross = \begin{cases} P_{cmax} - \frac{(P_{cmax} - P_{cmin}) * (f - f_{avg})}{f_{max} - f_{avg}} & f \leq f_{avg} \\ P_{cmax} & f > f_{avg} \end{cases} \quad (3)$$

Where f is the fitness value of the current individual, f_{avg} represents the average fitness value of all the current individual, f_{min} represents the minimum fitness value of all current individual, P_{cmax} is the maximum crossover probability and P_{cmin} is the minimum crossover probability.

The adaptive mutation probability formula is established as follows:

$$pmutation = \begin{cases} P_{mmin} + \frac{(P_{mmax} - P_{mmin}) * (f - f_{avg})}{f_{max} - f_{avg}} & f \leq f_{avg} \\ P_{mmin} & f > f_{avg} \end{cases} \quad (4)$$

Where P_{min} represents the minimum probability of variation, P_{max} represents the maximum probability of variation, f represents the fitness value of the current individual, f_{avg} represents the average fitness value of all current individuals, f_{min} represents the minimum fitness value of all current individuals.

2.3.2 Improved Particle Swarm Optimization algorithm

The traditional particle swarm optimization algorithm has the defects of slow convergence rate and easy to fall into local optima when it is applied, the researchers have proposed many improved methods[16].The adaptive ω is mainly used to update the speed and position[17]. Inertia weight ω is the ability of particles to maintain the inertia of motion and represents the global search ability. In the process of iteration, the value of ω should be adjusted according to different periods. Based on the traditional PSO algorithm, this experiment adds the concave function decreasing inertia weight. In addition, this experiment also improved learning factors c_1 and c_2 , which regulate individual learning ability and group learning ability respectively.

The formula for updating ω is as follows:

$$\omega(i) = \omega_{max} - (\omega_{max} - \omega_{min}) * ((\frac{2i}{T_{max}}) - (\frac{i}{T_{max}})^2) \quad (5)$$

Where ω_{max} represents the maximum value of ω , ω_{min} represents the minimum value of ω , T_{max} represents the maximum number of iterations, and i represents the current number of iterations.

The update formula of learning factors c_1 and c_2 is as follows:

$$\begin{cases} c1 = c1_{max} - (c1_{max} - c1_{min}) * \sin(\frac{i*\pi}{2*T_{max}}) \\ c2 = c2_{min} + (c2_{max} - c2_{min}) * \sin(\frac{i*\pi}{2*T_{max}}) \end{cases} \quad (6)$$

Where T_{max} represents the maximum number of iterations, and i represents the current number of iterations, and $c1_{max}$ represents maximum number of c_1 , and $c1_{min}$ represents the minimum number of c_1 , and $c2_{max}$ represents the maximum number of c_2 , and $c2_{min}$ represents the minimum number of c_2 .

3 Results Analysis

3.1 Data Source and Model Performance Evaluation

The data in this paper are from the monitoring results of Wuhan National Rice Trading Center Co., Ltd., a total of 162 sets of sample data. In this experiment, mean absolute error (MAE), mean square error (MSE) and root mean square error (RMSE) were used to evaluate the performance of the model. That is:

$$MAE = \frac{1}{N} \sum_{i=1}^N |X_i - Y_i| \quad (7)$$

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (X_i - Y_i)^2} \quad (8)$$

Where X_i is the predicted value, Y_i is the real value, N and is the number of samples.

The root mean square error (RMSE) is used to reflect the degree of dispersion between the predicted value and the true value of the model. The smaller the RMSE value is, the smaller the dispersion between the predicted value and the true value is, and the higher the accuracy of the prediction result is.

3.2 Model Comparative Analysis

The experimental test platform is: Windows 10 and MATLAB R2016a. In the experiment, 162 pieces of data were obtained after processing. Select the first 130 data pieces (80%) as the training set and the last 32 data pieces (20%) as the test set.

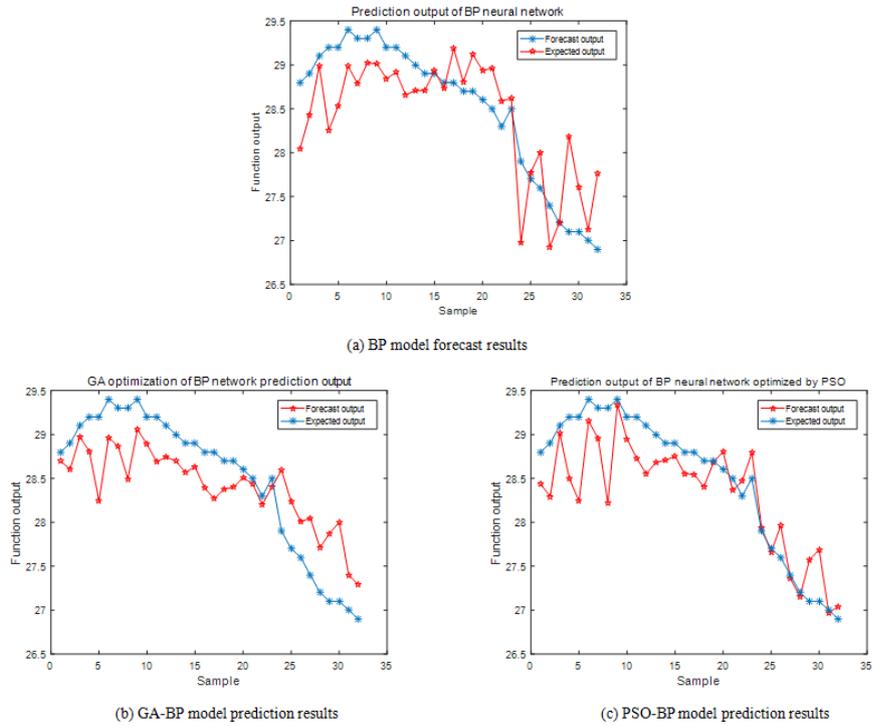


Fig. 1. Comparison of prediction results of three models

It can be seen from the fig 1 that the prediction error of the BP model is very large, which is closely related to the initial weight and threshold value. After optimizing the initial weights and thresholds using an optimization algorithm. As can be seen from the

figure, the prediction results of GA-BP model and PSO-BP model are closer to the true value than those of BP model. The predicted output curve of the model fits better with the actual output curve. It shows that the intelligent algorithm is effective to optimize the BP neural network and can improve the prediction effect of the model.

Table 1. Prediction results evaluation of BP model,GA-BP model and PSO-BP model

Characteristic index	BP model	GA-BP model	PSO-BP model
Average absolute error (MAE)	0.39964	0.41129	0.3056
Root mean square error (RMSE)	0.48551	0.4713	0.40094

According to Table 1, when the average surface temperature is predicted, the RMSE predicted by the BP model is 0.48551, and the RMSE predicted by GA-BP model is 0.4713, which is 2.93% higher than that of BP model. The RMSE predicted by PSO-BP model is 0.40094, which is 17.42% higher than that of BP model. Therefore, it can be judged that PSO-BP model can predict the average temperature of grain heap surface more accurately.

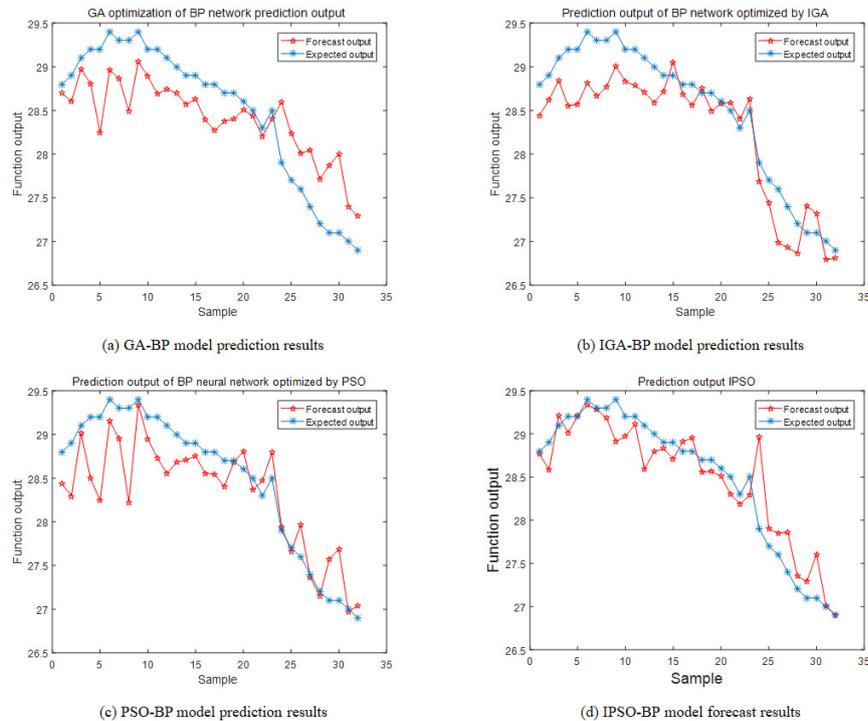


Fig. 2. Comparison of prediction results of four models

According to fig 2, it can be seen that the prediction results of BP neural network model optimized based on improved intelligent algorithm are closer to the actual value than the traditional BP neural network model optimized based on intelligent algorithm, and the model fitting effect is better.

Table 2. Evaluation of prediction results of four models

Characteristic index	GA-BP model	IGA-BP model	PSO-BP model	IPSO-BP model
Average absolute error (MAE)	0.41129	0.3102	0.3056	0.20698
Root mean square error (RMSE)	0.4713	0.35967	0.40094	0.29295

According to Table 2, when GA-BP model predicts the average temperature of grain heap surface, RMSE is 0.4713. When the IGA-BP model was used for prediction, the RMSE was 0.35967, 23.69% higher than GA-BP model; when the PSO-BP model was used for prediction, the RMSE was 0.40094; when the IPSO-BP model was used for prediction, the RMSE was 0.29295, 26.93% higher than PSO-BP model. Therefore, it can be judged that the BP neural network model optimized based on the improved intelligent algorithm can predict the average temperature of the surface layer of grain heap more accurately.

4 Conclusion

The weights and thresholds in BP neural network have obvious influence on the prediction effect of the whole model, so the selection of initial weights is directly related to the convergence speed and error accuracy of the whole algorithm, and is a crucial step. By comparing the five models, the following conclusions are drawn:

(1)The BP neural network optimized based on the intelligent optimization algorithm speeds up the convergence rate of the whole model, improves the error accuracy of the model, and can predict grain temperature more effectively.

2)Compared with the traditional intelligent algorithm, the BP neural network model optimized by the improved intelligent algorithm has better prediction effect and is more accurate for the average temperature of grain heap surface;

(3)Compared with IGA-BP model, IPSO-BP model is more accurate in predicting the average surface temperature of grain heap, and the predicted value is closer to the real value, indicating that IPSO-BP model is a better method to predict the average surface temperature of grain heap compared with the other four models.

References

1. Wu Cunrong, Tang Huaijian, Wang Yanyan. Current situation and Prospect of grain storage standard system in China [J]. Journal of China Grain and oil journal, 2010, 25 (11): 124-128.
2. Jian F, Jayas DS, White NDG. Specific heat, thermal diffusivity, and bulk density of genetically modified canola with high oil content at different moisture contents, temperatures, and storage times[J]. Transaction of the ASABE,2013,56(3):1077-1083.
3. Abe T, Basunia MA. Simulation of Temperature and Moisture Changes During Storage of Rough Rice in Cylindrical Bins Owing to Weather Variability[J]. Journal of Agricultural Engineering Research, 1996, 65(3):223-233.
4. Jian F, Jayas DS,White NDG. Temperature fluctuations and moisture migration in wheat stored for 15 months in a metal silo in Canada[J]. Journal of Stored Products Research,2008,45(2):82-90.

5. Hammami F, Ben Mabrouk, Mami. Modelling and simulation of heat exchange and moisture content in a cereal storage silo[J]. Mathematical and Computer Modelling of Dynamical Systems,2016,22(3):207-220.
6. Zhang Qian, Zhou Yongjie, Xin Liyong, Liu Zhiqiang, Cao Yang. Study on Grain Temperature Change Law and Mathematical Model of Tall Bungalow Storage [J]. Grain Storage, 2003,06: 25-30.
7. Jin Li Bing, Duan Haijing, Xue Yaqi, Wang Zhenqing, Wu Qiang. Experiment and Numerical Simulation of Grain Temperature Field in Underground Ecological Grain Grain [J]. Modern Food Technology, 2021,37 (01): 111-116.
8. Wang Yuancheng, Shi Tianyu, Qu Anandi, et al. Numerical simulation and experiment of bilateral suction inclined circulation air in tall bungalow barn [J]. Chinese Journal of Grain and Oil, 2020,35 (03): 139-146.
9. Dong Zhaok, GoFeng. Insect population data analysis and its implementation on the SPSS software [J]. Journal of Applied Entomology, 2013,50 (04): 1163-1169.
10. Bao Jinsong, Li Zhiqiang, Zhou Yaqin. Optimization on Ship-borne Equipment Based on Genetic Algorithm [J]. System Simulation Journal, 2019,31 (5): 901-908.
11. Glover F. Future paths for integer programming and links to artificial intelligence[J]. Computers&operations research,1986,13(5):533-549.
12. Glover F. Tabu search-part I[J]. ORSA Journal on computing,1989,1(3):190-206.
13. Liu Jianhua. Research on the Basic Theory and Improvement of Particle Group Algorithms [D]. Central South University, 2009.
14. Yang Fan, Yu Ming, Li Dan, Ren Hongge. Optimization of BP neural network for CO_2 flux prediction based on particle swarm optimization [J]. Journal of Natural Sciences, Heilongjiang University, 2017,34 (4): 481-485.
15. Zhang Lian, Gong Yu, Yang Hongjie,et al. Based on the adaptive genetic particle group algorithm [J]. Journal of Chongqing University of Technology (Natural Science). <https://kns.cnki.net/kcms/detail/50.1205.T.20210516.1202.002.html>
16. He Dan, Lin Laipeng, Li Xiaoyong,et al. Soft Measurement Model for Wastewater Treatment Based on Improving BP Particle Club Algorithm [J]. Journal of South China Normal University (Natural Science Edition), 2021,53 (2): 114-120.
17. Wang Feng, Zhou Yihong, Zhao Chunju, Zhou Huawei,et al. Inversion Analysis of Hot Blood Parameters of Different Materials of Ulgao Arch Dam Based on Hybrid Particle Group algorithm [J]. Journal of Tsinghua University (Natural Sciences Edition), 2021,61 (07): 747-755.

Abstracts

Revisiting Sorting and Selection Applied to Median Filtering Extended Abstract

James Cooper¹[0000-0001-9954-3280] and Radu Nicolescu¹[0000-0003-2498-1002]

School of Computer Science, University of Auckland
Private Bag 92019, Auckland 1142, New Zealand
jcoo092@aucklanduni.ac.nz

1 Introduction

Sorting is a common and important operation across Computer Science, including in Membrane Computing [1]. One of the most attractive properties of P systems is the capacity for unbounded parallelism, which has been used well in the past [11]. cP systems [3, 8, 9] combines this with logical pattern matching on associative data objects through multiset unification to enable succinct fixed-size rulesets to solve problems [7]. Motivated by the image processing problem of median filtering [5, Chap. 3.4.1], cP systems rulesets for sorting and statistical operations have been developed. All these operations require fixed-size rulesets and have a time complexity of $O(1)$. *None* of these rulesets are uniform or semi-uniform families of rules. They need no form of customisation to a particular problem, nor do they rely on any preprocessing. This extended abstract provides a ruleset of just two fixed rules to sort arbitrary numerical sets in a constant time of two steps, which can be followed by one more one-step rule to select an arbitrary n th (such as a median).

Problem	# of rules	# of steps
Minimum & Maximum	1	1
Counting elements	2	2
Counting frequency of elements	3	3
Sum	2	2
Mean	5	4
Mode	4	4
Sorting Sets	2	2
Sorting Multisets into ranges	4	4
Sorting Multisets with unique identifiers	3	3
Selection over Sets	2	2
Selection over Multisets	5	5
Selection over Multisets with unique identifiers	4	4

The table above summarises our cP models for various statistical operations, listing the name of each and the number of rules and steps it requires — note the boldfaced Sorting Sets and Selection over Sets, briefly presented in the sequel. Currently, these operations focus only on nonempty sets and multisets of natural numbers greater than zero, i.e., a particular case of total ordering, where the order relation can be straightforwardly checked. Statistics such as

the sum and mean may make little sense in non-numeric contexts, but sorting is more widely applicable. Further research might investigate non-numerical cases of total ordering or scenarios where a more general partial ordering is desirable. Example partial orderings include division-based orders and ordering (potentially non-intersecting) subsets of a given set or multisets over a given alphabet. A good starting point might be to systematise the construction of abstract representations of Hasse diagrams. Added rules to deal with zeros and empty sets are reasonably simple to formulate but dealing with the full range of integers has not been addressed yet. Nor, for that matter, have encompassing sets such as rational numbers or (useful finite approximations of) the entire class of real numbers. A good representation of rational and real numbers in cP systems is an open problem. Something based on Gustafson’s “unums” [6] may be a reasonable starting point.

2 Sorting a Set to Selecting the Median

The two rules needed to sort sets with cP systems are listed below. The final sorted order is found in the second value of each element of a set of indexed r objects. At the start (“step 0”) of the operation, assume the presence in the cP systems top-level cell of an arbitrary nonempty set of numerical s objects. These rules count the occurrence of values strictly less than the current value and add that many copies of the unary digit to the index value for the r object. In each instance, this number plus one is equal to the value’s correct index in the total ordering, thus sorting the values.

$$s_1 \quad \rightarrow_+ s_2 r(R)(1) \mid s(R) \quad (1)$$

$$s_2 r(Y)\{\} \rightarrow_+ s_3 r(Y)\{1\} \mid s(X) \mid X \subsetneq Y \quad (2)$$

For example, consider the set $\{s(6), s(3), s(7), s(2), s(5)\}$. After following the sorting rules, each element of the set has an associated index in the range $[1, |S|]$: $\{r(6)(4), r(3)(2), r(7)(5), r(2)(1), r(5)(3)\}$. One further rule thus suffices to select the n^{th} (and therefore the median) value:

$$s_3 \rightarrow_1 s_4 t(T) \mid r(T)(N) \mid n(N) \quad (3)$$

For instance, with $n(3)$, $t(5)$ is selected. We use an extended version of these rules to implement median filtering on a grid representing an image’s pixels.

3 Related Work

Unlike our cP solutions, all other P systems solutions known to us propose uniform or semi-uniform families of P systems that require preprocessing (sometimes more than linear). Ceterchi and Sburlan gave an overview of approaches to sorting in P systems at the time of their writing [1]. All the discussed approaches are interesting and innovative approaches to sorting that fit themselves to the properties of P systems. In all instances, however, the best results had a linear time complexity in either the number of elements in the multiset to sort or the size of the largest element. More recent work included [4]. The best time complexity among the procedures described there comes from Section 3.3, which describes a method to sort in constant time (three steps). The authors comment,

however, that this relies on “some precomputed resources of size $O(n^2)$.” The cP systems sorting rules need no such preprocessing.

Of course, the field of Membrane Computing is not the only one to have investigated improving the time complexity of parallel sorting. Both [10] and [2] presented highly parallel theoretical implementations of quicksort based on concurrent-read concurrent-write parallel random access machines. These works concluded that their time complexity is on the order $O(\log_2 n)$. It appears that the main limitation of those methods compared to cP systems is that the comparison operation is only performed between pairs of elements in a single step. In contrast, cP systems permits (effectively) an unbounded number of comparisons.

References

1. Ceterchi, R., Sburlan, D.: Membrane Computing and Computer Science. In: Pun, G., Rozenberg, G., Salomaa, A. (eds.) *Oxford Handbook of Membrane Computing*, chap. 22, pp. 553–583. Oxford University Press, Inc., New York, USA (2010)
2. Chlebus, B.S., Vro, I.: Parallel quicksort. *Journal of Parallel and Distributed Computing* **11**(4), 332–337 (Apr 1991). [https://doi.org/10.1016/0743-7315\(91\)90040-G](https://doi.org/10.1016/0743-7315(91)90040-G)
3. Cooper, J., Nicolescu, R.: Neighbourhood Message Passing on a Lattice with cP systems: Part One (2021), submitted for publication
4. Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S.: Kernel P Systems Modelling, Testing and Verification - Sorting Case Study. In: *Lecture Notes in Computer Science*, vol. 10105 LNCS, pp. 233–250. Springer Berlin Heidelberg (2017). https://doi.org/10.1007/978-3-319-54072-6_15
5. Gimel'farb, G., Delmas, P.: *Image Processing and Analysis: A Primer*, *Primers in Electronics and Computer Science*, vol. 03. World Scientific (Europe) (2019). <https://doi.org/10.1142/q0173>
6. Gustafson, J.L.: *The End of Error*. Chapman and Hall/CRC, New York, New York, USA (Jun 2017). <https://doi.org/10.1201/9781315161532>
7. Henderson, A., Nicolescu, R., Dinneen, M.J.: Solving a PSPACE-complete problem with cP systems. *Journal of Membrane Computing* **2**(4), 311–322 (Dec 2020). <https://doi.org/10.1007/s41965-020-00064-w>
8. Liu, Y., Nicolescu, R., Sun, J.: An efficient labelled nested multiset unification algorithm. *Journal of Membrane Computing* (May 2021). <https://doi.org/10.1007/s41965-021-00076-0>
9. Nicolescu, R., Henderson, A.: An Introduction to cP Systems. In: Graciani, C., Riscos-Núñez, A., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Enjoying Natural Computing*, *Lecture Notes in Computer Science*, vol. 11270, pp. 204–227. Springer International Publishing (2018). https://doi.org/10.1007/978-3-030-00265-7_17
10. Powers, D.M.W.: Parallelized QuickSort and RadixSort with Optimal Speedup. In: *PROCEEDINGS OF INTERNATIONAL CONFERENCE ON PARALLEL COMPUTING TECHNOLOGIES. NOVOSIBIRSK*. pp. 167–176. World Scientific, Novosibirsk, Russia (1991), <https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.9071>
11. Sosik, P.: P systems attacking hard problems beyond NP: a survey. *Journal of Membrane Computing* **1**(3), 198–208 (Sep 2019). <https://doi.org/10.1007/s41965-019-00017-y>

Spiking neural P systems with energy and threshold

Qianqian Ren and Xiyu Liu *

Shandong normal university, Shandong Jinan 250014, China

Abstract. Inspired by the energy function in Boltzmann machine, a new variant of neural membrane system, called spiking neural P systems with energy and threshold (ETSN P systems), is proposed. And assume that the temperature is a constant, it is not going to affect the system. According to the laws of physics, the less energy the system has, the more stable it will be. Therefore, the change of the system will develop towards the direction of less energy. Inspired by this law, the system we proposed has been improved on the firing rules. Extend firing rule with probability. And the connections between neurons are made through synapses, which has weights. In addition, the systems have two kinds of data, input data and threshold.

Keywords: energy function, Boltzmann machine, spiking neural P systems, threshold

According to the Boltzmann machine, the actual output of the neuron will occur according to a probability, and the output can only be in two states: 0 or 1. An important characteristic is that the system is always changing in the direction of less energy, that means the neuron produces values with high probability. In other words, when the probability is greater than 0.5, the output tends to be 1, and When less than 0.5, the output tends to be 0. Therefore, in the ETSN P system, the neuron either produces a spike or produce empty according to the firing rules. The probability function that outputs 1 is

$$P_j(1) = \frac{1}{1 + e^{-\sum_i (w_{ij}u_i - b_j)/T}} \quad (1)$$

Where b_j is the threshold of neuron j , and u_i is the number of spikes from neuron i to neuron j .

An ETSN P system with degree $m \geq 1$ is a construct of the form

$$\Pi = (O, \sigma_1, \dots, \sigma_m, W, I_0, syn, in, out) \quad (2)$$

Where,

* Corresponding author.

- 1) $O = \{a\}$ is the singleton alphabet, a is the spike.
- 2) $\sigma_1, \dots, \sigma_m$ are neurons, of the form $\sigma_j = (u_i, b_j, P, R_j)$. Where,
 - $u_i \geq 0 \in R$ is the number of spikes which comes from the connected neuron σ_i .
 - b_j is the threshold in neuron .
 - P is the probability function such as equation (1). It determines whether or not an spike is generated.
 - R_j is a finite set of rules, with the form $a^P \rightarrow a^S$. Where $S \in \{0, 1\}$. If $S=0$, the rule can be written as $a^P \rightarrow \lambda$.
- 3) W is the set of weights on synapses. If $w < 0$, then synapses are inhibitory synapses. Otherwise, the synapses are excitatory synapses. When $w = 0$, there is no connection between neurons.
- 4) I_0 is the set of initial states.
- 5) $syn \subseteq \{1, 2, \dots, n\} \times \{1, 2, \dots, m\} \times W$ are synapses. For each $(i, j) \in syn$, $1 \leq i \leq n$, $1 \leq j \leq m$, and $(i, j) \neq (j, i)$.
- 6) in, out denotes the input neuron and output neuron, respectively.

Acknowledgment

This research was funded by the National Natural Science Foundation of China (Nos. 61876101, 61802234 and 61806114), the Social Science Fund Project of Shandong (16BGLJ06, 11CGLJ22), China Postdoctoral Science Foundation Funded Project (2017M612339, 2018M642695), Natural Science Foundation of the Shandong Provincial (ZR2019QF007), China Postdoctoral Special Funding Project (2019T120607) and Youth Fund for Humanities and Social Sciences, Ministry of Education (19YJCZH244).

Two New Variants of Spiking Neural P Systems

Xiaoxiao Song

School of Electrical Engineering and Electronic Information, Xihua University,
Chengdu, Sichuan 610039, P.R. China
songxx_xhu@163.com

Abstract. Motivated by some biological considerations, two new variants of spiking neural P systems (SN P systems) are proposed. By considering the length of axons in nervous systems, spiking neural P systems with delay on synapse were proposed, which move the delay time onto synapses. Thus, the spikes leaving a presynaptic neuron could reach the postsynaptic neurons at different moments. By adopting autapses, spiking neural P systems with autapses were proposed. With this special synapses, a neuron can transmit information back to itself. It was proved that both of these two kinds of SN P systems are universal as number generators, and some small universal systems based on the variants in computing functions were constructed.

Keywords: Spiking neural P systems · Delay on synapses · Autapses.

Spiking neural P systems (SN P systems), were proposed as a branch of membrane computing, which are usually represented by directed graphs. SN P systems are restricted to concepts mimicking the transmission of electrical pulses (spikes) among neurons. Over the last 10 years, hundreds of papers have been published focusing on SN P systems. These papers are focus on researching the computation ability of SN P systems in generating and accepting numbers, computing functions, generating languages, and also solving some computationally hard problems and some combinatorial optimization problems. Motivated by biological and mathematical considerations, dozens of variants of SN P systems were explored.

In this year, inspired from some biological observation, we proposed two interesting variants of SN P system, named *spiking neural P systems with delay on synapse* (SNP-DS systems) [1] and *spiking neural P systems with autapses* (SNP-AU systems) [2].

1 Spiking Neural P systems with Delay on Synapse

In the human brain there are approximately 86×10^9 neurons and 10^{15} synapses among them. It is worth pointing out the diversity of the axons with respect its length: whereas some axons have a length less than a millimeter, others have a meter or more. Besides, the information travels at different speeds within different neurons. Consequently, spikes leaving a presynaptic neuron reach the

postsynaptic neurons at different moments. Inspired by these biochemical facts, a new kind of computing models, called SNP-DS systems, are introduced in such manner that they incorporate delays (in the form of natural numbers) associated with synapses. In these models, when some spikes are sent from a presynaptic neuron to different postsynaptic neurons, the spikes will arrive to these neurons at certain time instants, depending on the delays associated with them.

Definition 1 An SNP-DS system with a degree of $q \geq 1$, is a tuple $\Pi = \{O, \sigma_1, \dots, \sigma_q, syn, D_{syn}, in, out\}$, where:

- (1) $O = \{a\}$ is the singleton alphabet (a denotes spike);
- (2) $\sigma_i = (n_i, R_i)$, $1 \leq i \leq q$, where n_i indicates the number of initial spikes and R_i denotes the rules, which can be of two types:
 - (a) Spiking/Firing rules: $E/a^c \rightarrow a^p$, where $E \in REG(O)$, the set of regular expressions over O , and c, p are natural numbers such that $c \geq 1$ and $p \geq 0$;
 - (b) Forgetting rules: $E/a^c \rightarrow \lambda$, for some natural number $c \geq 1$;
- (3) $syn = \{(i, j) | 1 \leq i, j \leq (q \cup environment) \wedge i \neq j\}$;
- (4) D_{syn} is a mapping from syn onto the set N of natural numbers;
- (5) $in, out \in \{1, 2, \dots, q\}$.

Unlike the traditional SN P systems, where all the postsynaptic neurons receive spikes at a same instant from their presynaptic neuron, the postsynaptic neurons in SNP-DS systems would receive spikes at different instants, depending on the delay time on the synapses connecting them.

It is proved that the SNP-DS systems are universal as number generators. Two small universal SNP-DS systems, with standard or extended rules, are constructed to compute functions, using 56 and 36 neurons respectively.

2 Spiking Neural P systems with Autapses

Autapses are special synapses connecting the axon of a neuron onto itself. Autapses are often observed in cultured neurons, but over the last century they were considered as redundant synaptic connections, seemingly unproductive. However, within the last 10 years, some experiments have started exposing the important role that autapses play in neural networks. Autaptic excitation was found to be important in persistent activities of some neurons. Autapses were also found to inhibit subsequent action potential generation in fast-spiking neurons in the human and rat epileptic neocortex. It was also revealed that autapses promote neuronal responsiveness, burst firing, and coincidence detection in neocortical pyramidal cells. In addition, autapses associated with single neurons can significantly promote synchronization in the network. Consequently, these studies indicate that autapses are functional structures playing an important role in neural systems. Based on the structure and function of autapses, we proposed a new class of computation devices SNP-AU systems.

Definition 2 An SNP-AU system is defined as the following tuple:

$$\Pi = \{O, \sigma_1, \dots, \sigma_m, syn, in, out\},$$

where:

- (1) $O = \{a\}$ is the singleton alphabet, with its single symbol a representing a spike;
- (2) $\sigma_1, \dots, \sigma_m$, for $1 \leq i \leq m$, denote neurons of the form $\sigma_i = (n_i, R_i)$, with:
 - (a) $n_i \geq 0$ indicating the initial spikes stored in σ_i ;
 - (b) R_i being the rules assigned to neuron σ_i , with the form $E/a^c \rightarrow a^p$, where $E \in REG(O)$ ($REG(O)$ denotes the regular expression over the singleton alphabet O) and $c \geq p \geq 0$ (with c and p denoting the number of spikes consumed and produced, respectively, by using the spiking rule);
- (3) $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\} \times K$ represents the synapses among neurons, including autapses for some of such neurons; all the synapses will be of the form (i, j, k) , with $1 \leq i, j \leq m$ (with $i = j$ in the case of autapses) and $k \in K$ (with K representing the set of number of synapses, and k representing the number of synapses from neuron σ_i to neuron σ_j , being such number of synapses from i to j equal to 1 if $i \neq j$ (regardless of k), and k if $i = j$);
- (4) in denotes the input neuron;
- (5) out denotes the output neuron.

For the SNP-AU systems proposed, a synapse can connect the axon and a dendrite of the same neuron. This means that a synapse with the form (i, j) would allow $i = j$; that is, a neuron can transmit information back to itself (i.e., self-stimulation).

Statistically, every randomly chosen pair of nearby excitatory neurons has between 1 and 6 synaptic connections, and between 4 and 5 ones for inhibitory basket neurons. Something similar happens to autapses. In this way, the neurons in SNP-AU systems could have more than one autapse. Therefore, with one autapse, a neuron can preserve the number of its inner spikes constant after firing a rule sending one spike, given the fact that the spike could also come back to such neuron using its autapse. On the other hand, by including more autapses in the same neuron, the number of spikes contained in the neuron can grow, with the rule generating more spikes (through its autapses) than the ones previously stored.

We proved that SNP-AU systems can generate Turing-computable numbers, through the simulation of the modules of universal register machines. This result improves significantly the results given by classical SNP systems in terms of the number of neurons and rules, while preserving simplicity and power to a reasonable extent. Moreover, we construct an SNP-AU system using 53 neurons, proving its universality for computing functions.

References

1. X. Song, L. Valencia-Cabrera, H. Peng, J. Wang, M. J. Pérez-Jiménez: Spiking neural P systems with delay on synapses. *International Journal of Neural Systems* **31**, 2050042 (2021).
2. X. Song, L. Valencia-Cabrera, H. Peng, J. Wang: Spiking neural P systems with autapses. *Information Sciences* **570**, 383–402 (2021).

Enzymatic Spiking Neural P Systems*

Xiang Tian^{1,2} and Xiyu Liu^{1,2} **

¹ Business School, Shandong Normal University, Jinan 250358, China

² Academy of Management Science, Shandong Normal University, Jinan 250358, China

In organisms, only red blood cells do not have enzymes, and other living cells have enzymes, and nerve cells (or neurons) are no exception. Pavel et al. [1] proposed an Enzyme Numerical P System (ENPS) in 2010, but its structure and rules do not apply to the SN P system. This work is based on the following biological facts, that is, neurotransmitters are synthesized by specific synthase in neurons. From this perspective, a new SN P system is proposed, called the Enzymatic SN P system (ESNP). Different from the traditional SN P system, ESNP has two objects: pulse and enzyme, and the number of enzymes controls the number of reactions.

The proposed ESNP structure is as follows.

$$\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn, in, out)$$

Where:

1. $O = \{a, e\}$ defines the object alphabet (a stands for spike and e stands for enzyme).
2. $\sigma_1, \sigma_2, \dots, \sigma_m$ represent m neuron cells, $\sigma_i = (a_i, e_i, R_i), 1 \leq i \leq m$, where
 - a. a_i and e_i represent the initial spike and enzyme in the neuron σ_i respectively.
 - b. R_i represents a finite set of rules.
3. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ represents the synaptic connection between neurons.
4. $in, out \in \{1, 2, \dots, m\}$ correspond to the input and output neurons of the system, respectively.

* This work was partly supported by the National Natural Science Foundation of China (Nos. 61876101, 61802234, 61806114), the Social Science Fund Project of Shandong (Nos. 11CGLJ22, 16BGLJ06), the Natural Science Foundation of the Shandong Province (No. ZR2019QF007), the Youth Fund for Humanities and Social Sciences, Ministry of Education (No. 19YJCZH244), the China Postdoctoral Special Funding Project (No. 2019T120607), and the China Postdoctoral Science Foundation Funded Project (Nos. 2017M612339, 2018M642695).

** Corresponding author.

Evolution-Communication Spiking Neural P Systems with Energy on Neurons

Liping Wang¹ and Xiyu Liu² *

¹ Shandong Normal University, Shandong Jinan 250014, CHINA

² Shandong Normal University, Shandong Jinan 250014, CHINA

Abstract. The evolutionary communication spiking neural P system (ECSNP) is a variant of the spiking neural P system (SNP), which is used to describe the integration-fire behavior of neurons and the distribution of spikes generated, and to describe a mechanism of information conversion between neurons. In addition, the processes that occur in living cells, including the manipulation of matter, energy, and information, can be considered as computational processes. So far, almost all material evolution has been captured in the P system through the method of object evolution. However, energy as an implicit condition drives information transmission, and studies have found that neurons themselves carry energy. In this paper, we use “ e ” to quantify energy, and a new system, called evolution communication spiking neural P systems with energy on neurons (ECSN Pe), is proposed. Evolution rules release energy while achieving integrate-and-fire behavior of neurons. The execution of communication rules is driven by energy and consumes energy. The computing power of the new system has been proven.

Keywords: spiking neural P system, evolutionary-communication, energy, universality

1 ECSN Pe systems

The new system (ECSN Pe system) of degree $m \geq 1$ is a construct:

$$\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn, in, out) \quad (1)$$

where

1. $O = a$ is the singleton alphabet of a unique object a , which is usually called a spike;
2. $\sigma_1, \sigma_2, \dots, \sigma_m$ are neurons, of the form $\sigma_i = (n_i, R_i)$, for each $1 \leq i \leq m$, where

* Corresponding author: Xiyu Liu

- a) $n_i \geq 0$ refers to the quantity of spikes placed in neuron σ_i in the initial state;
b) R_i) refers to a set of rules, each rule has one of the following two forms:
(i) spike-evolution rules:

$$E|a^r \rightarrow a^s e^t$$

where E is a regular expression over O , $r \leq 1$, $s \geq 0$, $t \leq 1$ and $r \geq s$, e^t represents the energy released by the rule;

- (ii) spike-communication rules:

$$(a^p, go, e^q)$$

for some $p \geq 1, q \geq 1$, e^q represents the energy consumed by the rule;

3. $syn \in 1, \dots, m \times 1, \dots, m$ is a synaptic expression, for any $(i, j) \in syn$, $1 \leq i, j \leq m$, $i \neq j$;

4. in, out indicate the input and output neurons of the system, respectively.

A Review on the Research of Rough Set Membrane Computing

Qiang Yang, Gexiang Zhang*

School of Control Engineering, Chengdu University of Information Technology,

Chengdu, 610225, China

zhgxdylan@126.com

Abstract Membrane computing is a biological computational model with great potential in distributed and parallel computing, inspired by the membrane structure of living cells. As so far, membrane computing is mainly used to deal with deterministic information. But in fact, problems in real life often become uncertain due to lack of definite information or based on dynamic information. Therefore, the rough set membrane computing is proposed. In this paper, we introduce the basic concept of rough set membrane computing firstly. Then we summarize the recent years' research progress of rough set membrane computing in two aspects: theoretical research, application research. Finally, the paper gives the existing problems and research prospect of rough set membrane computing.

Keywords: rough set membrane computing; definite information; dynamic information

Numerical P systems of directed graph structure with threshold and plasticity

Xiu Yin¹, Xiyu Liu¹, Jie Xue¹, and Yuzhen Zhao¹ *

Shandong Normal University, Shandong Jinan 250014, CHINA

Wu et al.[1] inspired by the numerical properties of Numerical P (NP) systems[2], and combined NP systems and SN P systems for the first time and proposed Numerical Spiking Neural P (NSN P) systems. At the end of this article, it is mentioned that the semantics of the production function and the repartition protocol in NSN P systems are different from those in NP systems, which means that the NSN P system is not a NP system with a directed graph structure. Therefore, this paper proposes numerical P systems of directed graph structure with threshold and plasticity.

$$\Pi = (\sigma_1, \sigma_2, \dots, \sigma_m, syn, in, out)$$

where

- 1) $\sigma_1, \sigma_2, \dots, \sigma_m$ are neurons of the form $\sigma_i = (\theta_i, Var_i, Pr_i, Var_i(0)), 1 \leq i \leq m$, where
 - a) θ_i is the threshold of neuron σ_i ;
 - b) $Var_i = \{x_{q,i} | 1 \leq q \leq k_i\}$ is a set of variables in neuron σ_i
 - c) $Var_i(0) = \{x_{q,i}(0) | x_{q,i}(0) \in R, 1 \leq q \leq k_i\}$ refers to the set of initial values of the variables $x_{q,i}$ in set Var_i .
 - d) Pr_i is a finite set of programs associated with neuron σ_i . Program l from neuron σ_i has the following two forms:
 - General form: $pr_{l,i} = F_{l,i}(x_{1,i}, \dots, x_{k_i,i})$, where F is called “production function” .
 - Plasticity form: $pr_{l,i} = F_{l,i}(x_{1,i}, \dots, x_{k_i,i}) \rightarrow \alpha k(i, N_j), \alpha \in \{+, -, \pm, \mp\}, k \geq 1, 1 \leq j \leq m, \text{ and } N_j \subseteq \{1, 2, \dots, m\}$.
- 2) $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ are synapses between neurons, for each $(i, j) \in syn, 1 \leq i, j \leq m, \text{ and } i \neq j$.

* Corresponding author Xiyu Liu, xyliu@sdu.edu.cn. Research is supported by the Natural Science Foundation of China(No.61170038), the Natural Science Foundation of Shandong Province(No.ZR2011FM001)

3) $in, out \in \{1, \dots, m\}$ indicate the input and output neurons respectively.

Each neuron in DGNP-TP systems contains a threshold θ , only when the production value of the production function in the neuron is not less than the threshold, the neuron will fire. Taking neuron σ_i as an example, if the production value $pr_{l,i}(t) = F_{l,i}(x_{1,i}(t), \dots, x_{k_i,i}(t))$ of neuron σ_i at time t is not less than threshold θ_i , then neuron σ_i will fire at time t .

For a plasticity program $pr_{l,i} = F_{l,i}(x_{1,i}, \dots, x_{k_i,i}) \rightarrow \alpha k(i, N_j)$, $\alpha \in \{+, -, \pm, \mp\}$, the part on the right of \rightarrow is called “repartition protocol”, N_j is a set of neurons, and neuron σ_i can connect to neurons in N_j (synapse creation) or not (synapse deletion), which depends on α and k .

For α and k , we intend to use α and k in spiking neural P systems with structural plasticity.

References

1. Wu, T., et al., Numerical Spiking Neural P Systems. *IEEE Trans Neural Netw Learn Syst*, 2021. 32(6): p. 2443-2457.
2. Paun, G., Numerical P Systems (After Ten Years). 2017: p. 26-29.

Tissue Neural Like P System

Xiaoling Zhang and Xiyu Liu

Shandong Normal University, Shandong Jinan 250014, CHINA

Abstract: Artificial neural network abstracts the human brain neuron network from the perspective of information processing. It is a computing model composed of a large number of interconnected nodes. Similarly, the Tissue-like P system regards cells as the vertices of the graphics in the system, which is a type of parallel computing model. In cell biology, molecules move from high concentration to low concentration on the membrane, and the osmotic pressure of the extracellular fluid refers to the attraction of solute particles in the solution. Inspired by this biological knowledge, we introduced the concept of penetration rate and proposed a tissue neural P system (TNP System) based on the general model of artificial neural networks(Fig 1). We proved the universality of the TNP system. In addition, we also proved that the proposed organization P system can solve the SAT problem.

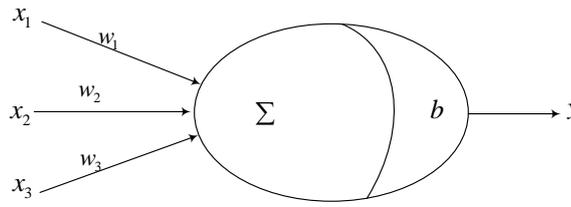


Figure1: the general model of artificial neural network

The structure of the Tissue Neural P system (TNP System) based on the general model of the neural network is as follows:

$$\Pi = (O, b, \varepsilon, M_0, \dots, M_q, P_r, R(i, j), i_{out})$$

- O is the alphabet, which contains all objects in the system;
- b is the threshold;
- $\varepsilon \subseteq O$ is the alphabet of objects that originally existed in the environment;
- $M_i, 1 \leq i \leq q$ is a multi-set of objects that exist in cell q at the initial time;
- P_r is the penetration rate equivalent to the weight in the artificial neural network;
- $R(i, j)$ is the set of symport rules;
 - Symport rule $(s, u/\lambda, s')$
 - Activation rule $\begin{cases} 1, \sum Pr_i x_i < b \\ 0, \sum Pr_i x_i \geq b \end{cases}$
- i_{out} is the output region

Author Index

- Adorna H., 84, 352
Alhazov A., 21, 39, 54
Aman B., 69
Annadurai S., 606
Atulya K Nagar, 323
- Ballesteros K., 84, 264
Bera S., 102
Bhuvaneshwari K., 590, 606
- Cabarle F.G., 84, 264, 316, 352, 400
Cailipan D.P., 84
Ceterchi R., 102
Chan TN, 296
Chen H., 117
Chen Y., 443
Ciobanu G., 128
Cooper J., 627
- Dinneen M.J., 296
Deng X., 167
Dong J., 145, 167
Duan Y., 191, 243
Dupaya A.G., 264
- Feng H., 167
Freund R., 21, 39
- Galano A.C., 264
Gayathri Lakshmi M., 573
Gnanaraj Thomas D., 323, 573
Gheorghe M., 343
Guo D., 145, 475
- Han T., 390, 567
Happe H., 296
Henderson A., 296
Hernández-Tello J., 316
Hinze T., 296
Huang Y., 390, 567
- Ivanov S., 21, 39
Ipate F., 343
- James Immanuel S., 323, 573
Jayakrishna V., 582
Jayasankar S., 323, 573
Jie X., 639
- Kalyani T., 590, 606
- de La Cruz R.T., 84, 264, 352
Lazo P.P., 264, 352
Leporati A., 54
Li T., 475
Ling S., 475
Liu M., 497
Liu Q., 413
Liu S., 617
Liu X., 598, 630, 635, 636, 639
Liu Y., 598
Lisa Mathew, 582
Liu T., 510, 617
Liu X., 117, 413
Luo B., 145, 167
- Ma H., 475
Ma W., 617
Macabayao I.C., 264, 352
Manzoni L., 54
Martínez-Del-Amor M.Á., 316, 400
Mauri G., 54
Meenakshi Paramasivan, 573
- Nicolescu I.M., 343
Nicolescu R., 296, 627
Niu Y., 497
- Orellana-Martín T., 316, 379
Oswald M., 39
- Pérez-Jiménez M. J., 191, 243, 379
- Qi D., 191, 243
- Raman T. T., 590
Ravichandran P., 590
Ren Q., 630
Robinson T., 323
Rong H., 145, 167, 191, 243
- Song H., 390, 567
Song Q., 390, 567
Song X., 632, 635
Sriram S., 102
Subramanian K.G., 102
Sweety F., 606

Tian X., 635
Thomas D. G., 590, 606
Todoran E.N., 128

Valdez A.A., 400
Valencia-Cabrera L., 191, 243, 379
Verlan S., 21

Wang L., 636
Wang S., 167
Wee F., 400
Wu J., 443
Wu P., 617
Wu Q., 413

Xu F., 457
Xu S., 390, 567

Yang H., 510
Yang Q., 145, 638, 432
Yang X., 432
Yin J., 529
Yin X., 639
Yu J., 510, 443
Yu W., 443

Zandron C., 54
Zhang L., 457
Zhang G., 145, 167, 191, 243, 529, 638, 432
Zhang Y., 617
Zhao Y., 413, 598, 639
Zhou J. 529
Zhou K., 510, 529, 617
Zhu M., 145