CDMTCS Research Report Series

Proceedings of the Asian Branch of International Conference on Membrane Computing (ACMC2018)

Michael J. Dinneen
Radu Nicolescu
(Editors)

Department of Computer Science
University of Auckland
Auckland, New Zealand

CDMTCS-530
December 2018

Centre for Discrete Mathematics and
Theoretical Computer Science

Contents

Network Controllability: algorithmics for cancer medicine

Ion Petre

Computational Biomodeling Laboratory, Åbo Akademi University and Turku Centre for Computer Science, Turku, Finland and the National Institute for Research and Development in Biological Science, Bucharest, Romania

Abstract. The intrinsic robustness of living systems against perturbations is a key factor that explains why many single-target drugs have been found to provide poor efficacy or to lead to significant side effects. Rather than trying to design selective ligands that target individual receptors only, network polypharmacology aims to modify multiple cellular targets to tackle the compensatory mechanisms and robustness of disease-associated cellular systems, as well as to control unwanted off-target side effects that often limit the clinical utility of many conventional drug treatments. However, the exponentially increasing number of potential drug target combinations makes the pure experimental approach quickly unfeasible, and translates into a need for algorithmic design principles to determine the most promising target combinations to effectively control complex disease systems, without causing drastic toxicity or other side-effects. Building on the increased availability of disease-specific essential genes, we concentrate on the target structural controllability problem, where the aim is to select a minimal set of driver/driven nodes which can control a given target within a network. That is, for every initial configuration of the system and any desired final configuration of the target nodes, there exists a finite sequence of input functions for the driver nodes such that the target nodes can be driven to the desired final configuration. We investigated this approach in some pilot studies linking FDA-approved drugs with cancer cell-line-specific essential genes, with some very promising results.

Designing the Square Kilometre Array Computer System

Andrew Ensor

High Performance Computing Research Laboratory
Auckland University of Technology, Auckland, New Zealand

Abstract. The Square Kilometre Array is the largest mega-Science project of the next decade aiming to build enormous radio telescope arrays across Western Australia and Southern Africa. With 160 TeraByte/s data generated in just stage one and over 260PetaFLOP compute requirements it presents unprecedented data movement and processing challenges. This talk will outline the project, the progress made by the design team toward overcoming its computing challenges, and some of the key components being led by New Zealand researchers.

Andrew is the Director of the High Performance Computing Research Laboratory. His research interests include HPC and GPU computing, distributed and mobile systems, algorithms, concurrency and computer graphics. Andrew is also the Director of the New Zealand SKA Alliance, a group of 35 NZ academic and industry partners working on the Exascale computer design for the Square Kilometre Array Project.

An Improved Universal Spiking Neural P System with Generalized Use of Rules

Yun Jiang[1,2] *, Yansen Su[3], and Fen Luo[1,2]

[1] Chongqing Engineering Laboratory for Detection, Control and Integrated Systems
Chongqing Technology and Business University, Chongqing 400067, China
[2] School of Artificial Intelligence,
Chongqing Technology and Business University, Chongqing 400067, China
[3] Key Lab of Intelligent Computing and Signal Processing of Ministry of Education,
School of Computer Science and Technology,
Anhui University, Hefei 230039, China
jiangyun@email.ctbu.edu.cn

Abstract. Spiking neural P systems (SN P systems, for short) are a class of distributed and parallel computing devices inspired from the way neurons communicate by means of spikes. In most of the SN P systems investigated so far, the spiking rules are usually used in a sequential way or an exhaustive way. Recently, a generalized way of using rules, applicable for both spiking rules and forgetting rules, is considered in SN P systems. This new strategy of using rules is a combination of the two ways mentioned above: if a rule is used at some step, it can be applied any possible number of times, nondeterministically chosen. In this work, we investigate the computational completeness of SN P systems with generalized use of rules. Specifically, we construct a universal SN P systems with generalized use of rules, where each neuron contains at most five rules, each spiking rule consumes at most four spikes for each time, and each forgetting rule removes at most four spikes for each time. Compared to the original work, this result makes an improvement to the related parameters, thus provides an answer to the open problem.

Keywords: Membrane computing, Spiking neural P system, Generalized use of rules, Computational completeness

1 Introduction

Brain is a rich source of inspiration for informatics. Specifically, it has provided plenty of ideas to construct high performance computing models, as well as to design efficient algorithm. Inspired from the biological phenomenon that neurons cooperate in the brain by exchanging spikes via synapses, various neural-like computing models have been proposed. In the framework of membrane computing, a kind of distributed and parallel neural-like computing model were proposed in 2006 [1], which is called spiking neural P systems (SN P systems, for short).

* Corresponding author.

Since the human brain and biological neurons are rich sources of computing ideas, many variants of SN P systems have been introduced, taking inspiration from biological phenomena, e.g. synapse weight, neuron division, astrocytes, inhibitory synapses, as in [2–19]. Investigation on the theoretical and practical usefulness have also been applied to these variants: their computing power in relation to well-known models of computation, e.g. finite automata, register machines, grammars, computing numbers or strings as in [20–32]; computing efficiency in solving hard problems, as in [33, 34].

Moreover, practical applications and software for simulations have been developed for SN P systems and their variants: to design logic gates, logic circuits [35] and databases [36], to represent knowledge [37], to diagnose fault [38–40], to approximately solve combinatorial optimization problems [41].

Briefly, SN P systems have neurons that process only one type of symbols, the spike, based on the indistinct signal used by biological neurons. Neurons are placed on nodes of a directed graph, and the edges between neurons are called synapses, again based on synapses of biological neurons. SN P systems processes spikes by applying rules, and two of the most common types are spiking rules and forgetting rules. Spiking rules are of the form $E/a^c \to a^p; d$, where E is a regular expression over $\{a\}$, and c, p, d are natural numbers, $c \geq p \geq 1$, $d \geq 0$. Using a rule $E/a^c \to a^p; d$ means that c spikes are consumed in the neuron and p spikes are produced after a delay of d steps. The produced spikes are sent to all neurons connected by an outgoing synapses from the neuron where the rule was applied. Forgetting rules are of the form $E/a^c \to \lambda$, whose application removes c spikes from the neuron and generates no spike.

During the evolution of SN P systems, the way of applying rules plays a crucial role. In the earlier version of SN P systems, the spiking rules are usually used in a sequential way [9, 20, 22, 23, 33, 34], which means that in a neuron, at a step, one of the applicable rules is nondeterministically chosen for the application, and the chosen rue is applied only once, so the system works in a sequential way at the level of the neuron. Later the exhaustive way of using rules is proposed [4], which is inspired by the biological fact that an enabled chemical reaction consumes as many related substances as possible. The exhaustive use of rules means that in a neuron, at a step, one of the applicable rules is nondeterministically chosen for the application, and the chosen rule is applied as many times as possible. So the exhaustive use of rules is a kind of local parallelism at the level of a neuron [30, 31].

Recently, generalized use of rules, which is similar to the minimal parallelism mode in P systems [42], is considered for SN P systems [43]. This new way of using rules is a combination of the sequential use of rules and the exhaustive use of rules. Specifically, generalized use of rules means that in a neuron, at a step, one of the applicable rules is nondeterministically chosen for the application, and the chosen rule is applied for any l, $1 \leq l \leq m$ times, where m is the maximum number of times for which the chosen rule can be applied under the exhaustive way of using rules. In [43], it is proved that SN P systems with generalized use of rules are Turing universal as number computing devices. The computational

completeness is achieved when each neuron of the system contains at most seven rues, each spiking rule consumes at most nine spikes for each time, and each forgetting rule removes at most seven spikes for each time. Also in [43], it is mentioned that these parameters in the above result may be optimized without losing the universality.

In this work, we improve the related parameters mentioned above. Specifically, we construct a universal SN P systems with generalized use of rules, where each neuron contains at most five rules, each spiking rule consumes at most four spikes for each time, and each forgetting rule removes at most four spikes for each time. This universality result provides an answer to the open problem mentioned in [43].

This work is organized as follows. In section 2, we simply review the computing models investigated in this work: SN P systems with generalized use of rules. The computational completeness of SN P systems with generalized use of rules is investigated in section 3. Conclusions and remarks are given in section 4.

2 Spiking Neural P Systems with Generalized Use of Rules

In this section, we simply review SN P systems with generalized use of rules. For more details, readers can look up in [43].

Formally, an SN P system with generalized use of rules, of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, out),$$

where:

1. $O = \{a\}$ is a singleton alphabet (a is called spike);
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons, of the form

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

where:

(1) $n_i \geq 0$ is the initial number of spikes placed in the neuron σ_i;
(2) R_i is a finite set of rules of the following two forms:
- Firing rule: $E_1/a^{c_1} \rightarrow a^p; d$, where E_1 is a a regular expression over $\{a\}$, and $c_1 \geq p \geq 1$, $d \geq 0$ (called a delay). Specifically, when $d = 0$, it can be omitted;
- Forgetting rule: $E_2/a^{c_2} \rightarrow \lambda$, where E_2 is a a regular expression over $\{a\}$, and $c_2 \geq 1$, with the restriction that for each rule $E_1/a^{c_1} \rightarrow a^p; d$ of type (1) from R_i, we have $E_1 \cap E_2 = \emptyset$;

3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ is the set of synapses between neurons, with restriction $(i, i) \notin syn$ for $1 \leq i \leq m$ (no self-loop synapse);

4. $out \in \{1, 2, \ldots, m\}$ indicates the output neuron, which can emit spikes to the environment.

If a spiking rule $E/a^c \to a^p; d$ has $E = a^c$ and $d = 0$, then it is simply written as $a^c \to a^p$. Similarly, if a forgetting rule $E/a^c \to \lambda$ has $E = a^c$, then it is simply written as $a^c \to \lambda$. In the following sections, the feature of delay is not used in all SN P systems, so the spiking rules are always of the form $E/a^c \to a^p$.

In an SN P system working in a generalized use of rules, the rules are applied as follows.

For a spiking rule $E/a^{c_1} \to a^p \in R_i$, if neuron σ_i contains k_1 spikes, $a^{k_1} \in L(E)$ and $k_1 \geq c_1$, then the rule can be applied. However, the essential we consider here is not the form of the rules, but the way they are used. Using the rule in a generalized manner, as suggested in the Introduction, means the following. We assume that $k_1 = s_1 c_1 + r_1$, for some $s_1 \geq 1$ (this means that we must have $k_1 \geq c_1$) and $0 \leq r_1 < c_1$ (the remainder of dividing k by c), then $n_1 c_1$ spikes can be consumed, where n_1 is nondeterministically chosen from the set $\{1, 2, \ldots, s_1\}$. If the rule consumes $n_1 c_1$ spikes, $1 \leq n_1 \leq s_1$, then $k_1 - n_1 c_1$ spikes remain in neuron σ_i, while $n_1 p$ spikes are produced and sent to each of the neurons σ_j such that $(i, j) \in syn$. In the case of σ_i being the output neuron, $n_1 p$ spikes are also sent to the environment. In the case of σ_i having no synapse leaving from it, the produced spikes are lost.

For a forgetting rule $E/a^{c_2} \to \lambda \in R_i$, if neuron σ_i contains k_2 spikes, $a^{k_2} \in L(E)$ and $k_2 \geq c_2$, then the rule can be applied. We assume that $k_2 = s_2 c_2 + r_2$, with $s_2 \geq 1$ and $0 \leq r_2 < c_2$, then $n_2 c_2$ spikes can be removed, where n_2 is nondeterministically chosen from the set $\{1, 2, \ldots, s_2\}$. If the rule removes $n_2 c_2$ spikes, $1 \leq n_2 \leq s_2$, then $k_2 - n_2 c_2$ spikes remain in neuron σ_i.

In each time unit, in each neuron which can use a rule we have to use a rule, either a firing or a forgetting one. In some time, in a neuron, it is possible that several rules can be applied. In this case, only one of the rules is nondeterministically chose to be applied, and the chosen rule will be applied in a generalized way as mentioned above.

The configuration of the system is described by the numbers of spikes present in each neuron. Thus, the initial configuration is $\langle n_1/0, n_2/0, \ldots, n_m/0 \rangle$. Using the rules as described above, we can define *transitions* among configurations. Any sequence of transitions starting from the initial configuration is called a *computation*. A computation halts if it reaches a configuration where no rule in the neuron can be used. This configuration is usually called the halting configuration.

The result of a computation can be defined in several ways. In this work, we consider SN P systems with generalized use of rules as number generators: the computation result is defined as the total number of spikes sent to the environment by the output neuron during the computation, i.e., this computation should be a halting one, otherwise, the computation is considered as an invalid computation and gives no result. For an SN P system Π with generalized use of rules, the set of all numbers computed in this way is denoted by $N^{gen}(\Pi)$, with the subscript *gen* indicating that the rules are applied in a generalized

way. We denote by $SpikP_m^{gen}(rule_k, cons_r, forg_q)$ the family of all sets $N^{gen}(\Pi)$ computed as above by SN P systems with at most $m \geq 1$ neurons, using at most $k \geq 1$ rules in each neuron, with all spiking rules $E/a^{c_1} \rightarrow a^p$ having $c_1 \leq r$, and all forgetting rules $E/a^{c_2} \rightarrow \lambda$ having $c_2 \leq q$. When any of these parameters m, k, r, q is not bounded, it is replaced with $*$.

In the next sections SN P systems are represented graphically, which is easy to understand. A neuron is represented by an oval with the initial spikes and rules inside. Each neuron has incoming and outgoing arrows which indicates their communications with other neurons, and the output neuron has an outgoing arrow pointing to environment, suggesting that it can send spikes to the environment.

3 An Improved Universal Spiking Neural P Systems with Generalized Use of Rules

In this section we present an improved universal spiking neural P systems with a generalized way of using rules, where each neuron of the SN P system contains at most five rules, each spiking rule consumes at most four spikes for each time, and each forgetting rule removes at most four spikes for each time. Our universality proof will use the characterization of NRE by means of register machine.

A register machine is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labeling an ADD instruction), l_h is the halt label (assigned to instruction HALT), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. The labeled instructions are of the following forms:

- $l_i : (ADD(r), l_j)$ (add 1 to register r and then go to the instruction with label l_j),
- $l_i : (SUB(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- $l_h : HALT$ (the halt instruction).

A register machine M generates a set of number $N(M)$ in the following way. Starting with all registers empty (i.e., storing the number zero), the system applies the instruction with label l_0 and continues to apply instructions as indicated by the labels (and made possible by the contents of registers); if the system reaches the halt instruction, the number n present in register 1 at that time is said to be generated by M. The set of all numbers generated by M is denoted by $N(M)$. It is known that the register machine can generates each recursively enumerable set of numbers, even with only three registers [44]. Hence, register machines characterizes NRE, i.e., $N(M) = NRE$. Without loss of generality, We follow the convention that when comparing the power of two number generating devices, number zero is ignored.

In the proof below, we use the characterization of NRE by means of register machine, with an additional care paid to the number of rules in each neuron, and the number of spikes consumed and removed in each rule.

Theorem 1. $Spik^{gen}P_*(rule_5, cons_4, forg_2) = NRE$.

Proof. In view of the Turing-Church thesis, the inclusion in NRE can be proved directly, so we only have to prove the inclusion $NRE \subseteq Spik^{gen}P_*(rule_5, cons_4, forg_4)$. The proof is achieved in a constructive way, that is, an SN P system with generalized use of rules is constructed to simulate the universal register machine.

Let $M = (m, H, l_0, l_h, I)$ be a universal register machine. Without lose of generality, we assume that the result of a computation is the number stored in register 1 and this register is never decremented during the computation.

In what follows, We construct an SN P system Π working in a generalized way of using rules to simulate the register machine M.

For each register r of M we consider a neuron σ_r in Π whose contents correspond to the contents of the register. Specifically, if the register r holds the number $n \geq 0$, then the neuron σ_r will contain $2n$ spikes. Therefore, the contents of a register r increasing by 1 means the number of spikes from the neuron σ_r increasing by 2; the contents of a non-empty register decreasing by 1 means the number of spikes decreasing by 2; checking whether the register is empty amounts at checking whether σ_r has no spike inside.

With each label l of an instruction in M we also associate a neuron σ_l. Initially, all these neurons are empty, except for the neuron σ_{l_0} associated with the start label of M, which contains 2 spikes. This means that this neuron is "activated". Additional neurons will be associated with the registers and the labels of M, in a way which will be described immediately. During the computation, the neuron σ_l which receives 2 spikes will become active and start to an instruction $l_i : (OP(r), l_j, l_k)$ of M (OP is ADD or SUB). Simulating the instruction $l_i : (OP(r), l_j, l_k)$ of M means starting with neuron σ_{l_i} activated, operating the register r as requested by OP, then introducing 2 spikes in one of the neuron σ_{l_j}, σ_{l_k}, which becomes active in this way. When the neuron σ_{l_h}, associated with the halting label of M, is activated, the computation in M is completely simulated in Π.

Note that in both the ADD and the SUB modules, the rules from neurons σ_{l_j}, σ_{l_k} are written in the form $a^2 \to a^{\delta(l_q)}$ ($q = j$ or k), because we do not know whether σ_{l_j} or σ_{l_k} is a label of ADD, SUB, or halting instruction. That is why we use the function δ, defined on H as follows:

$$\delta(l) = \begin{cases} 2, \text{ if } l \text{ is the label of an ADD instruction,} \\ 1, \text{ otherwise.} \end{cases}$$

In what follows, the work of system Π is described (that is how system Π simulates the ADD, SUB instructions of register machine M and outputs the computation result).

Module ADD: simulating an ADD instruction $l_i : (ADD(r), l_j, l_k)$

As shown in Figure 1, the ADD module works as follows.

Let us assume that at some step t, the system starts to simulate an ADD instruction $l_i : (ADD(r), l_j, l_k)$ of M and register r holds the number n. At that moment, neuron σ_{l_i} contains two spikes, and the other neurons are empty,

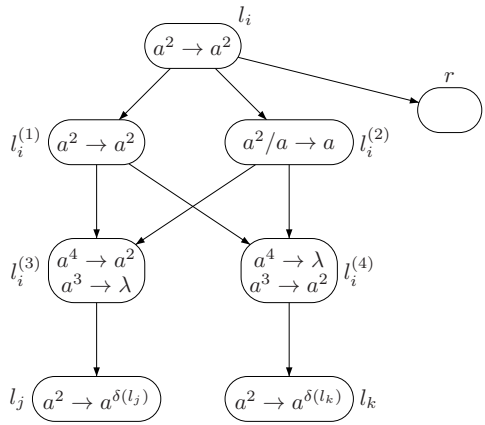


Fig. 1. Module ADD for simulating $l_i : (ADD(r), l_j, l_k)$

except for neurons associated with the registers. With two spikes in neuron σ_{l_i}, rule $a^2 \to a^2$ in neuron σ_{l_i} is enabled at step t, and the ADD module is initiated.

At step t, the rule $a^2 \to a^2$ in σ_{l_i} is enabled and is used for only once, sending two spikes to each of neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and σ_r. At step $t+1$, neuron σ_r receives two spikes, which means the content in register r is incremented by one. Also at step $t+1$, both neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ are activated. In neuron $\sigma_{l_i^{(1)}}$, rule $a^2 \to a^2$ is enabled and can be applied for only once, sending two spikes to each of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$. In neuron $\sigma_{l_i^{(2)}}$, rule $a^2/a \to a$ is enabled and can be applied for once or two times nondeterministically, sending one spike or two spikes to each of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$, respectively. Consequently, each of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ receives three spikes if rule $a^2/a \to a$ is used for once or four spikes if rule $a^2/a \to a$ is used for two times.

If each of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ accumulates three spikes, the three spikes in neuron $\sigma_{l_i^{(3)}}$ are removed by applying rule $a^3 \to \lambda$, while the three spikes in neuron $\sigma_{l_i^{(4)}}$ enables rule $a^3 \to a^2$, and makes the neuron firing, sending two spikes to neuron σ_{l_k}. With two spikes inside, neuron σ_{l_k} becomes active, which corresponds to the fact that the system starts to simulate the instruction l_k of the machine M.

If each of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ accumulates four spikes, the four spikes in neuron $\sigma_{l_i^{(4)}}$ are removed by applying rule $a^4 \to \lambda$, while the four spikes in neuron $\sigma_{l_i^{(3)}}$ enables rule $a^4 \to a^2$, and makes the neuron firing, sending two spikes to neuron σ_{l_j}. With two spikes inside, neuron σ_{l_j} becomes active, which corresponds to the fact that the system starts to simulate the instruction l_j of the machine M.

Therefore, from firing neuron σ_{l_i}, the system adds two spikes to neuron σ_r and fires neuron σ_{l_j} or σ_{l_k} nondeterministically, which simulates the ADD instruction $l_i : (ADD(r), l_j, l_k)$ correctly.

Module SUB: simulating a SUB instruction $l_i : (SUB(r), l_j, l_k)$

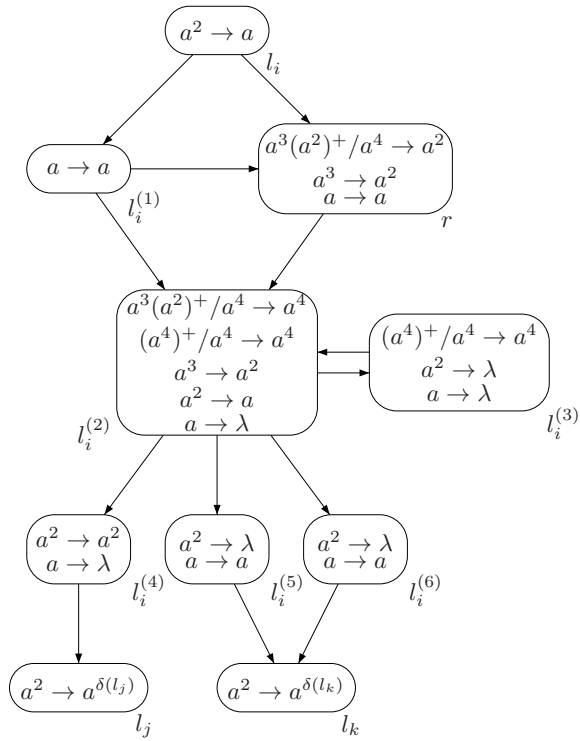


Fig. 2. Module SUB for simulating $l_i : (SUB(r), l_j, l_k)$

As shown in Figure 2, a SUB instruction $l_i : (SUB(r), l_j, l_k)$ is simulated in the following way.

Let us assume that at some step t, the system starts to simulate an SUB instruction $l_i : (SUB(r), l_j, l_k)$ of M and register r holds the number n. At that moment, neuron σ_{l_i} contains two spikes, and no spike is present in other neurons except for neurons associated with the registers. With two spikes in neuron σ_{l_i}, rule $a^2 \to a$ in neuron σ_{l_i} is enabled at step t, and the SUB module is initiated.

At step t, the rule $a^2 \to a$ in σ_{l_i} is enabled and is used for only once, sending one spikes to each of neurons $\sigma_{l_i^{(1)}}$ and σ_r. At step $t+1$, rule $a \to a$ in neurons $\sigma_{l_i^{(1)}}$ is used, sending one spike to neuron $\sigma_{l_i^{(2)}}$. Also at step $t+1$, neuron σ_r receives one spike from neuron σ_{l_i}, and the rules in it can be applied. There exists three cases for neuron σ_r.

In case 1, neuron σ_r has no spike at step t, which corresponds to the fact that the number stored in register r is zero. In this case, neuron σ_r has one spike at step $t+1$, and the rule $a \to a$ is enabled, sending one spike to neuron $\sigma_{l_i^{(2)}}$. Consequently, neuron $\sigma_{l_i^{(2)}}$ accumulates two spikes at step $t+2$ (one spike is received from neurons $\sigma_{l_i^{(1)}}$ and the other one from neuron σ_r), and the rule $a^2 \to a$ is enabled and applied, sending one spike to each of neurons $\sigma_{l_i^{(s)}}$, $3 \leq s \leq 6$. The spike in neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ will be removed by rule $a \to \lambda$ at the next step, while the spike in neuron $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ enables rule $a \to a$, and makes the neurons fire, sending one spike to neuron σ_{l_k}. Neuron σ_{l_k} accumulates two spikes at step $t+4$. Note that the spike in neuron σ_r received from neuron $\sigma_{l_i^{(1)}}$ at step $t+2$ will be consumed in the following two steps: this spike is sent to neuron $\sigma_{l_i^{(2)}}$ by applying rule $a \to a$; then the spike in neuron $\sigma_{l_i^{(2)}}$ is consumed by rule $a \to \lambda$. Therefore, neuron σ_{l_k} is activated, and the number of spikes stored in neuron σ_r is still zero, which correctly simulates SUB instruction l_i.

In case 2, neuron σ_r has two spikes at step t, which corresponds to the fact that the number stored in register r is 1. In this case, neuron σ_r has three spikes at step $t+1$, and the rule $a^3 \to a^2$ is enabled, sending two spikes to neuron $\sigma_{l_i^{(2)}}$ and leaving no spike in σ_r. Consequently, neuron $\sigma_{l_i^{(2)}}$ accumulates three spikes at step $t+2$ (one spike is received from neurons $\sigma_{l_i^{(1)}}$ and two from neuron σ_r), and the rule $a^3 \to a^2$ is enabled and applied, sending two spikes to each of neurons $\sigma_{l_i^{(s)}}$, $3 \leq s \leq 6$. The two spikes in neurons $\sigma_{l_i^{(3)}}$, $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ will be removed by rule $a^2 \to \lambda$ at the next step, while the two spikes in neuron $\sigma_{l_i^{(4)}}$ enables rule $a^2 \to a^2$, and makes the neuron fires, sending two spikes to neuron σ_{l_j}. Neuron σ_{l_j} accumulates two spikes at step $t+4$. Similarly, the spike in neuron σ_r received from neuron $\sigma_{l_i^{(1)}}$ at step $t+2$ will be removed by using rule $a \to a$ in neuron σ_r and rule $a \to \lambda$ in neuron $\sigma_{l_i^{(2)}}$. In this way, neuron σ_{l_j} is activated, and the number of spikes stored in neuron σ_r becomes zero. Therefore, SUB instruction l_i is correctly simulated.

In case 3, neuron σ_r has $2n$ ($n \geq 2$) spikes at step t, which corresponds to the fact that the number stored in register r is greater than 1. In this case, neuron σ_r accumulates $2n+1$ ($n \geq 2$) spikes at step $t+1$, and rule $a^3(a^2)^+/a^4 \to a^2$ is enabled. Under a generalized way of using rules, rule $a^3(a^2)^+/a^4 \to a^2$ can be applied for several times, nondeterministically chosen.

To illustrate the simulation of SUB instruction l_i, we use a simple example as follows. We assume that the number stored in register r is 6. So at step $t+1$, neuron σ_r contains 13 spikes, and rule $a^3(a^2)^+/a^3 \to a^2$ can be used for once or twice or three times, nondeterministically chosen.

If rule $a^3(a^2)^+/a^3 \to a^2$ is used for once, then nine spikes remain in neuron σ_r, and two spikes are sent to neuron $\sigma_{l_i^{(2)}}$. Note that one spike is also sent to neuron σ_r from neuron $\sigma_{l_i^{(1)}}$ at step $t+2$, so neuron σ_r accumulates ten spikes in total at step $t+2$, and no rule in this neuron can be used again. For neuron $\sigma_{l_i^{(2)}}$, three spikes are accumulated at step $t+2$ (one spike is received from neurons

$\sigma_{l_i^{(1)}}$ and two from neuron σ_r), and the rule $a^3 \to a^2$ is enabled and applied, sending two spikes to each of neurons $\sigma_{l_i^{(s)}}$, $3 \leq s \leq 6$. The two spikes in neurons $\sigma_{l_i^{(3)}}$, $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ will be removed by rule $a^2 \to \lambda$ at the next step, while the two spikes in neuron $\sigma_{l_i^{(4)}}$ enables rule $a^2 \to a^2$, and makes the neuron fires, sending two spikes to neuron σ_{l_j}. In this way, neuron σ_{l_j} is activated, and the number of spikes stored in neuron σ_r becomes ten, which means that the number stored in register r becomes five.

If rule $a^3(a^2)^+/a^4 \to a^2$ is used for twice, then four spikes are sent to neuron $\sigma_{l_i^{(2)}}$ and five spikes remain in neuron σ_r. At the next step, neuron σ_r will accumulates 6 spikes (one of them is received from neuron $\sigma_{l_i^{(1)}}$), and the 6 spikes cannot be consumed by any rule in neuron σ_r. For neuron $\sigma_{l_i^{(2)}}$, it accumulates 5 spikes in total at step $t+2$ (four spikes are received from σ_r and one from neurons $\sigma_{l_i^{(1)}}$), enabling rule $a^3(a^2)^+/a^4 \to a^4$. By using rule $a^3(a^2)^+/a^4 \to a^4$, 4 spikes are sent to neuron $\sigma_{l_i^{(3)}}$, and 1 spike is left in neuron $\sigma_{l_i^{(2)}}$, where the spike will be removed by rule $a \to \lambda$. In this case, neuron $\sigma_{l_i^{(3)}}$ accumulate 4 spikes, enabling $(a^4)^+/a^4 \to a^4$. Rule $(a^4)^+/a^4 \to a^4$ in neurons $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$ will be used forever, which means that the computation cannot halt and thus gives no result. Such a computation will be ignored.

If rule $a^3(a^2)^+/a^4 \to a^2$ is used for three times, then six spikes are sent to neuron $\sigma_{l_i^{(2)}}$ and one spike remains in neuron σ_r. At the next step, neuron σ_r will accumulates two spikes (one of them is received from neuron $\sigma_{l_i^{(1)}}$), and the 2 spikes cannot be consumed by any rule in neuron σ_r. For neuron $\sigma_{l_i^{(2)}}$, it accumulates 7 spikes in total at step $t+2$ (six spikes are received from σ_r and one from neurons $\sigma_{l_i^{(1)}}$), enabling rule $a^3(a^2)^+/a^4 \to a^4$. By using rule $a^3(a^2)^+/a^4 \to a^4$, 4 spikes are sent to neuron $\sigma_{l_i^{(3)}}$, and 3 spikes are left in neuron $\sigma_{l_i^{(2)}}$, where the spike will be consumed by rule $a^3 \to a^2$. In this case, each of neurons $\sigma_{l_i^{(4)}}$, $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ accumulates 6 spikes, thus cannot be consumed by any rule in these neurons; neuron $\sigma_{l_i^{(3)}}$ receives 4 spikes from neuron $\sigma_{l_i^{(2)}}$ at step $t+3$, enabling rule $(a^4)^+/a^4 \to a^4$, then receives 2 spikes from neuron $\sigma_{l_i^{(2)}}$ at step $t+3$, enabling rule $a^2 \to \lambda$. Consequently, the computation also enters an endless loop by repeatedly using rule $(a^4)^+/a^4 \to a^4$ in neurons $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$. Such a computation will also be ignored.

So in case 3, under a generalized use of rules, the simulation has two possibilities: neuron σ_{l_i} is activated and the number of spikes in neuron σ_r is decremented by two, or the simulation does not halt and it gives no result. This means that SUB instruction l_i is still correctly simulated in case 3.

Therefore, the simulation of SUB instruction i correct: system Π starts from neuron σ_{l_i}, and ends in σ_{l_j} (if the number stored in register r is greater than 0), or in σ_{l_k} (if the number stored in register r is 0). Note that the non-halting simulation gives no result, and it is ignored.

Module FIN: Outputting the result of computation

As shown in Figure 3, the result of computation is output in the following way.

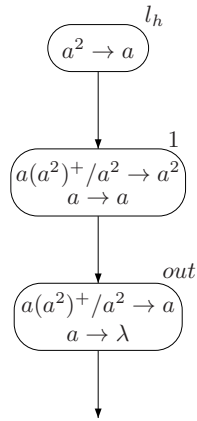


Fig. 3. Module FIN for outputting the result

Assume that the halting instruction l_h of M is reached, and thus the computation of M halts. This means that neuron σ_{l_h} receives two spikes. We assume that at that moment, neuron σ_1 has $2n$ spikes, corresponding to the fact that the number stored in register 1 is n, $n \geq 0$. With two spikes in neuron σ_{l_h}, rule $a^2 \to a$ is applied, sending one spike to neuron σ_1, and making rule $a(a^2)^+/a^2 \to a^2$ in neuron σ_1 enabled. Under a generalized way of using rules, rule $a(a^2)^+/a^2 \to a^2$ can be used for several times, nondeterministically chosen. No matter how many times rule $a(a^2)^+/a^2 \to a^2$ is used at first, the rule will be used repeatedly until one spike remains in neuron σ_1. In this way, $2n$ spikes are transferred from neuron σ_1 to neuron σ_{out}. The remaining one spike is then consumed by rule $a \to a$ in neuron σ_1, which makes rule $a(a^2)^+/a^2 \to a$ in neuron σ_{out} enabled. It is not difficult to find out that for each time rule $a(a^2)^+/a^2 \to a$ is applied, two spikes are consumed and one spike is sent to environment. Besides, the remaining spike is finally removed by rule $a \to \lambda$ in neuron σ_{out}. Therefore, the system sends n spikes to the environment in total during the computation, which is the exact number stored in register 1 of M when the computation of M halts.

Based on the explanations as above, it is clear that the register machine M is correctly simulated by SN P system Π working in a generalized way of using rules. Therefore, $N^{gen}(\Pi) = N(M)$, which completes the proof.

4 Remarks and Conclusion

In this work, we present an improved universal SN P system working in a generalized way of using rules. In the proof of the universality result, each neuron

of the constructed SN P system contains at most five rules, each spiking rule consumes at most four spikes for each time, and each forgetting rule removes at most four spikes for each time. Compared with the construction in [43], these parameters are optimized without losing the universality. These parameters may be further optimized by constructing the modules differently. This task is left as an open problem to the readers.

Acknowledgments. This work was supported by National Natural Science Foundation of China (61502063 and 61502004), Natural Science Foundation Project of CQ CSTC (cstc2018jcyjAX0057), Science and Technology Research Program of Chongqing Municipal Education Commission (KJQN201800814), and Chongqing Social Science Planning Project (2017YBGL142).

References

1. Ionescu, M., Păun, G., Yokomori, T. (2006). Spiking neural P systems. *Fundamenta Informaticae*, 71, 279-308.
2. Cavaliere, M., Ibarra, O.H., Păun, G., Egecioglu, O., Ionescu, M., Woodworth, S. (2009). Asynchronous spiking neural P systems. *Theoretical Computer Science*, 410, 2352–2364.
3. Pan, L., Păun, G. (2009). Spiking neural P systems with anti-spikes. *International Journal of Computers Communications & Control*, IV, 273–282.
4. Ionescu, M., Păun, G., Yokomori, T. (2007) Spiking neural P systems with an exhaustive us of rules. *International Journal of Unconventional Computing*, 3, 135–154.
5. Ionescu, M., Păun, G., Pérez-Jiménez, M.J., Yokomori, T. (2011). Spiking neural dP systems. *Fundamenta Informaticae*, 111, 423–436
6. Pan, L., Wang, J., Hoogeboom, H.J. (2012). Spiking neural P systems with astrocytes. *Neural Computation*, 24, 805–825.
7. Pan, L., Zeng, X., Zhang, X., Jiang, Y. (2012). Spiking neural P systems with weighted synapses. *Neural Processing Letters*, 35, 13–27.
8. Song, T., Pan, L., Păun, G. (2014). Spiking neural P systems with rules on synapses. *Theoretical Computer Science*, 529, 82–95.
9. Wang, J., Hoogeboom, H.J., Pan, L., Păun, G., Pérez-Jiménez, M.J. (2014). Spiking neural P systems with weights. *Neural Computation*, 22, 2615–2646.
10. Song, T., Liu, X., Zeng, X. (2015). Asynchronous spiking neural P systems with anti-spikes. *Neural Processing Letters*, 42, 633–647.
11. Song, T., Pan, L. (2015). Spiking neural P systems with rules on synapses working in maximum spiking strategy. *IEEE Transactions on Nanobioscience*, 14, 465–477.
12. Song, T., Pan, L. (2015). Spiking neural P systems with rules on synapses working in maximum spikes consumption strategy. *IEEE Transactions on Nanobioscience*, 14, 38–44.
13. Zhao, Y., Liu, X., Wang, W., Adamatzky, A. (2016). Spiking neural P systems with neuron division and dissolution. *Plos One*, 11, e0162882.
14. Wu, T., Zhang, Z., Păun, G., Pan, L. (2016). Cell-like spiking neural P systems. *Theoretical Computer Science*, 623, 180–189.
15. Jiang, K., Chen, W., Zhang, Y., Pan, L. (2016). Spiking neural P systems with homogeneous neurons and synapses. *Neurocomputing*, 171, 1548–1555.

16. Song, T., Pan, L. (2016). Spiking neural P systems with request rules. *Neurocomputing*, 193, 193–200.
17. Pan, L., Păun, G., Zhang, G., Neri, F. (2017). Spiking neural P systems with communication on request. *International Journal of Neural Systems*, 27(8), 1750042.
18. Pan, L., Wu, T., Su, Y., Vasilakos, A.V. (2017). Cell-Like spiking neural P systems with request rules. *IEEE Transactions on Nanobioscience*, 16(6), 513-522.
19. Wu, T., Păun, A., Zhang, Z., Pan, L. (2018). Spiking neural P systems with polarizations. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8), 3349-3360.
20. Ibarra, O.H., Păun, A., Rodríguez-Patón, A. (2009). Sequential SNP systems based on min/max spike number. *Theoretical Computer Science*, 410, 2982–2991.
21. Neary, T. (2009). A boundary between universality and non-Universality in extended spiking neural P systems. *Lecture Notes in Computer Science*, 6031, 475–487.
22. Song, T., Pan, L., Jiang, K., Song, B., Chen, W. (2013). Normal forms for some classes of sequential spiking neural P systems. *IEEE Transactions on Nanobioscience*, 12, 255–264.
23. Zhang, X., Zeng, X., Luo, B., Pan, L. (2014). On some classes of sequential spiking neural P systems. *Neural Computation*, 26, 974–997.
24. Wang, X., Song, T., Gong, F., Zheng, P. (2016). On the computational power of spiking neural P systems with self-organization. *Scientific Reports*, 6, 27624.
25. Chen, H., Freund, R., Ionescu, M. (2007). On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75, 141–162.
26. Krithivasan, K., Metta, V.P., Garg, D. (2011). On string languages generated by spiking neural P systems with anti-spikes. *International Journal of Foundations of Computer Science*, 22, 15–27.
27. Zeng, X., Xu, L., Liu, X. (2014). On string languages generated by spiking neural P systems with weights. *Information Sciences*, 278, 423–433.
28. Song, T., Xu, J., Pan, L. (2015). On the universality and non-universality of spiking neural P systems with rules on synapses. *IEEE Transactions on Nanobioscience*, 14, 960–966.
29. Wu, T., Zhang, Z., Pan, L. (2016). On languages generated by cell-like spiking neural P systems. *IEEE Transactions on Nanobioscience*, 15, 455–467.
30. Zhang, X., Zeng, X., Pan, L. (2008). On string languages generated by spiking neural P sysstems with exhaustive use of rules. *Natural Computing*, 7, 535–549.
31. Pan, L., Zeng, X. (2011). Small universal spiking neural P systems working in exhaustive mode. *IEEE Transactions on Nanobioscience*, 10, 99-105.
32. Wu, T., Bîlbîe, F-D., Păun, A., Pan, L., Neri, F. (2018). Simplified and yet Turing universal spiking neural P systems with communication on request. *International Journal of Neural Systems*, 28(8), 1850013.
33. Ishdorj, T.-O., Leporati, A., Pan, L., Zeng, X., Zhang, X. (2010). Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. *Theoretical Computer Science*, 411, 2345–2358.
34. Pan, L., Păun, Gh., Pérez-Jiménez, M.J. (2011). Spiking neural P systems with neuron division and budding. *Science China – Information Science*, 54, 1596–1607.
35. Song, T., Zheng, P., Wong, M.L., Wang, X. (2016). Design of logic gates using spiking neural P systems with homogeneous neurons and astrocytes-like control. *Information Sciences*, 372, 380–391.
36. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A. (2017). Semantics of deductive databases with spiking neural P systems. *Neurocomputing*. http://doi.org/10.1016/j.neucom.2017.07.007.

37. Wang, J., Shi, P., Peng, H., Pérez-Jiménez, M.J., Wang, T. (2013). Weighted fuzzy spiking neural P systems. *IEEE Transactions on Fuzzy Systems*, 21, 209–220.
38. Peng, H., Wang, J., Pérez-Jiménez, M.J., Wang, H., Shao, J., Wang, T. (2013). Fuzzy reasoning spiking neural P systems for fault diagnosis. *Information Sciences*, 235, 106–116.
39. Wang, J., Peng, H. (2013). Adaptive fuzzy spiking neural P systems for fuzzy inference and learning. *International Journal of Computer Mathematics*, 90, 857–868.
40. Wang, T., Zhang, G., Zhao, J., He, Z., Wang, J., Pérez-Jiménez, M.J. (2015). Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. *IEEE Transactions on Power Systems*, 30, 1182–1194
41. Zhang, G., Rong, H., Neri, F., Pérez-Jiménez, M.J. (2014). An optimization spiking neural P system for approximately solving combinatorial optimization problems. *International Journal of Neural Systems*, 24, 1440006.
42. Ciobanu, G., Pan, L., Păun, Gh., Pérez-Jiménez, M.J. (2007). P systems with minimal parallelism. *Theoretical Computer Science*, 378(1), 117–130.
43. Zhang, X., Wang, B., Pan, L. (2014). Spiking neural P systems with a generalized use of rules. *Neural Computation*, 26, 1–19.
44. Minsky, M. (1967). *Computation: Finite and Infinite Machines*. Upper Saddle River, NJ: Prentice Hall.

Generating Context-Free Languages using Spiking Neural P Systems with Structural Plasticity

Ren Tristan A. de la Cruz[1], Francis George C. Cabarle[1], and Henry N. Adorna[1]

Algorithms and Complexity Laboratory
Department of Computer Science
College of Engineering
University of the Philippines
P. Velasquez Street, Diliman, Quezon City, Philippines
{radelacruz,fccabarle,hnadorna}@up.edu.ph

Abstract. *Spiking neural P system* (SNP system) is a model of computation inspired by networks of spiking neurons. An SNP system is a network of neurons than can send an object, known as a *spike*, to each other. *Spiking neural P system with structural plasticity* (SNPSP system) is a variant of the classical SNP system. SNPSP system incorporates the ideas of *synaptogenesis* (creating new synapses) and *synaptic pruning* (deletion of existing synapses), collectively known as structural plasticity, as features of the model. This gives SNPSP systems the ability to change their own structure/topology. In this work, we used SNPSP systems to generate context-free languages. We created a procedure for constructing an SNPSP system given a context-free grammar in *Greibach* normal form (GNF). The resulting SNPSP system essentially simulates the way a context-free grammar in GNF is used to generate languages. We used modules known as *arithmetic-memory* modules, also created using SNPSP systems, to perform arithmetic operations which are needed for the simulation.

Keywords: Membrane Computing · SNPSP Systems · SNP System Variant · Context-free Languages · Arithmetic-Memory Module.

1 Introduction

Membrane computing studies family of related models of computation known as *P systems*. P systems are biologically-inspired unconventional models of computation [19]. They were formally introduced in [18]. P systems include models inspired by the mechanisms of biological cell. They include models of a group of cells (tissue-like models) and models of group of neurons (neural-like models).

Spiking neural P systems (SNP systems) [7] are neural-like P systems whose main computing elements (processors) are *spiking neurons*. The neurons also serve as storage, storing a single whole known as *spike count*. An SNP system is

a collection of such neurons connected to each other using *synapses*. We refer to this model as *classic* SNP system.

Classic SNP system is based on a simplified and abstracted spiking neurons. Different structures and mechanisms in the brain have been good sources of inspiration for different variants of the classic SNP systems. *e.g.* weighted synapses, neuron division, threshold mechanism, astrocytes, neuron budding, axons, etc. After being introduced, many other variants of SNP systems were able to incorporate these structures and mechanisms. Some of these SNP variants are in [15,17,21,22,12,13,28,20,27,23,16,14,26,25].

Spiking neural P system with structural plasticity (SNPSP system) is a variant of the classical SNP system that incorporates the ideas of *synaptogenesis* (creating new synapses) and *synaptic pruning* (deletion of existing synapses), collectively known as structural plasticity, as features of the model.This gives SNPSP systems the ability to change their own structure/topology.

SNP variants have been used as language generator [24,4,10,11,29,8,9,30,1]. For this work, we used SNPSP systems to generate context-free languages. We created a procedure for constructing an SNPSP system given a context-free grammar in *Greibach* normal form (GNF). The resulting SNPSP system essentially simulates the way a context-free grammar in GNF is used to generate languages. We used modules known as *arithmetic-memory* modules, also created using SNPSP systems, to perform arithmetic operations which are needed for the simulation.

The structure of the document is as follows: In Section 2, we discuss some prelimenary notions and notations like string, languages, morphisms, SNPSP systems, AM modules. In Section 3.1 we discuss GNF grammar, string encoding and stack operations. In Sections 3.2-3.6, we build in parts the SNPSP system for generating CFL. In Section 4, we give some concluding remarks.

2 Preliminaries

We assume the reader has basic knowledge of formal language and automata theory and some background on membrane computing. In this section we only recall relevant notions and notations from these areas.

2.1 Strings and Languages

An alphabet V is a finite set of symbols, a string s is a concatenation of symbols from some V, so that the string is said to be *over the alphabet* V. If s is a string over V, we denote as $|s|$ the length of s while $|s|_a$ where $a \in V$ denotes the number of occurrences of symbol a in s.

A language L is a set of strings. When talking about languages , the term *word* can be used as a synonym for string. For some alphabet V we have V^* as the language that contains strings of all lengths over V including the empty string, denoted as λ. Further, $V^+ = V^* - \{\lambda\}$.

If X and Y are alphabets, a morphism $h : X^* \to Y^*$ is a mapping that satisfies the condition $h(uv) = h(u)h(v)$ for $u, v \in X^*$. For a morphism $h : X^* \to Y^*$ and a string $y \in Y^*$, the inverse morphism from h is defined as $h^{-1}(y) = \{x \in X^* \mid h(x) = y\}$. These mappings are extended in the natural way to languages, i.e. for some language L we have $h(L) = \{h(w) \mid w \in L\}$. A morphism $h : X^* \to X^*$ is a *projection* if $h(a) \in \{a, \lambda\}$ for each $a \in X$.

2.2 Spiking Neural P Systems with Structural Plasticity

Spiking neural P systems with structural plasticity (or SNPSP systems) introduced in [2] are variants of classic spiking neural P systems (or SNP systems) from [7].

Formally, an SNPSP system Π of degree $m \geq 1$ is a construct $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out)$, where:

- $O = \{a\}$ is a singleton alphabet containing the spike symbol a.
- $\sigma_1, \ldots, \sigma_m$ are the *neurons* of the system. A neuron σ_i, $1 \leq i \leq m$, has the form (n_i, R_i) : n_i is a non-negative integer that indicates the initial number of spikes in σ_i represented by the string a^{n_i} over O; R_i is a finite set of rules with the following forms:
  1. **Spiking Rule:** $E/a^c \to a$ where E is a regular expression over O and $c \geq 1$. When $E = a^c$, the rule can be written as $a^c \to a$.
  2. **Plasticity Rule:** $E/a^c \to \alpha k(i, N)$ where $c \geq 1$, $\alpha \in \{+, -, \pm, \mp\}$, $N \subseteq \{1, \ldots, m\}$, and $k \geq 1$. When $E = a^c$, the rule can be written as $a^c \to \alpha k(i, N)$.
- $syn \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$, with $(i, i) \notin syn$, is the set of initial *synapses* between neurons.
- in, out are neuron labels that indicate the input and output neurons, respectively.

The semantics of SNPSP system are as follows. For every time step, each neuron of Π checks if any of their rules are applicable. Activation requirements of a rule are specified as E/a^c at the left-hand side of every rule. A rule $r \in R_i$ of σ_i is applicable if the following conditions are met: (1) the number of spikes in σ_i represented by a^n, is in the language described by E (i.e. $a^n \in L(E)$), and (2) σ_i contains n spikes and $c \leq n$.

It is possible that multiple rules are applicable in a neuron at a given time. This possibility occurs when the languages described by the regular expressions of the rules intersect, i.e. rules $r_1 : E_1/a^{c_1} \to a$, and $r_2 : E_2/a^{c_2} \to a$ have $L(E_1) \cap L(E_2) \neq \emptyset$. When multiple rules are applicable in a neuron, the neuron non-deterministically selects one rule to activate. When a rule is activated, c spikes are consumed in the neuron.

When a spiking rule is activated in neuron σ_i, all neurons σ_j such that $(i, j) \in syn$ receive a spike from σ_i at the same step as rule activation.

When a plasticity rule $E/a^c \to \alpha k(i, N)$ is activated in σ_i, the neuron performs one of the following actions depending on α and k:

1. ($\alpha = +$) Add at most k synapses from σ_i to k neurons whose labels are specified in N.
2. ($\alpha = -$) Delete at most k synapses that connect σ_i to neurons whose labels are specified in N.
3. ($\alpha = \pm$) At time step t, perform the actions for $\alpha = +$, then in time step $t + 1$, perform the actions for $\alpha = -$.
4. ($\alpha = \mp$) Similar to $\alpha = \pm$ except that actions for $\alpha = -$ are performed before $\alpha = +$.

Let $P(i) = \{j \mid (i, j) \in syn\}$, be the set of neuron labels such that σ_i is connected to σ_j by a synapse. If a plasticity rule is activated and is specified to add k synapses, there are cases when the rule can only add less than k synapses: when most of the neurons in N already have connections from σ_i, i.e. $|N - P(i)| < k$. The rule connects σ_i to the remaining neurons specified in N that are not in $P(i)$. If $|N - P(i)| = 0$ then there are no more synapses to add. If $|N - P(i)| = k$ then there are exactly k synapses to add.

When $|N - P(i)| > k$ then the rule non-deterministically selects k neurons from $N - P(i)$ and connects σ_i to the selected neurons.

Additionally, we note the following subsumed action: when a synapse is created at time step t, connecting σ_i to σ_j, a spike is sent to σ_j at the same time step t.

Similar cases can occur when deleting synapses. If $|P(i)| < k$, then a rule only deletes less than k synapses that connect σ_i to neurons specified in N that are also in $P(i)$. If $|P(i)| = 0$, then there are no synapses to delete. If $|P(i) \cap N| = k$ then the rule deletes exactly k synapses that connect σ_i to neurons specified in N.

When $|P(i) \cap N| > k$ then the rule non-deterministically selects k synapses that connect σ_i to neurons in N and delete the selected synapses.

A plasticity rule with $\alpha \in \{\pm, \mp\}$ activated at step t is applied until time step $t + 1$: during these steps, no other rules can be activated but the neuron can still receive spikes.

The output neuron has a synapse to the *environment*. For generating languages, and as is commonly done in literature such as [4,5,29,8] when a spiking rule is activated at time step t by the output neuron, the system is said to generate the symbol '1'. When no spiking rules are activated in the output neuron, the system is said to generate the symbol '0'. The string generated by the system is the *concatenation* of the symbols generated until the entire system halts.

When considering only finite strings, which are the ones considered in this work, if the system does not halt on a given computation then no string is generated. Using this interpretation, only binary languages are directly generated by the system. In this way, we define the language generated by Π as $L(\Pi) = \{w \in \{0, 1\}^+ \mid w \text{ is generated from a halting computation of } \Pi\}$.

Only binary languages without λ are considered since the output neuron of Π only generates the symbols '0' or '1' when a spike is sent or not sent to the environment, respectively.

2.3 Arithmetic-Memory Modules

Arithmetic-Memory (AM) modules [3] are SNPSP systems that act as information storage (memory) and calculating units (arithmetic). An AM module stores a whole number. Two AM modules can be instructed to perform any of the four arithmetic operations (addition, 'subtraction' - absolute difference, multiplication, whole number division).

If there are multiple AM modules, any two modules can be instructed to perform an arithmetic operation by sending spikes to specified neurons of the modules. One modules stores the value of the first operand while other module stores the value of the second operand. Once activated (by receiving spikes to specified neurons) the modules will interact with each other. The module holding the first operand ('first' module) will produce the result of the arithmetic operation. The total number of spikes produced by the first module during the operation is the result of the arithmetic operation.

Figure 1 shows a 'black-box' view of an AM module. It labels all the neurons of the module that send or receive spikes to/from neurons outside the module itself. We note the following neurons: $I_1, ..., I_9, O, E, A_1, A'_1, ..., A_4, A'_4$. Neurons $I_1, ..., I_9$ are the 'instruction' neurons. Neuron O is the 'output' neuron. Neuron E is the 'end signal' neuron. Neurons $A_1, A'_1, ..., A_4, A'_4$ are the 'addressing' neurons.

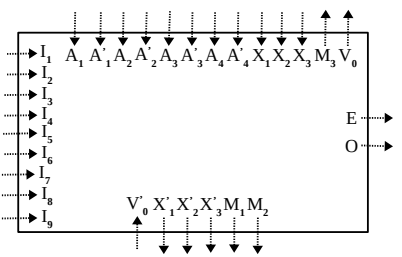


Fig. 1. A 'Black Box' View of an AM Module. All neurons shown send/receive spikes to/from neurons outside the AM module itself.

There are 9 addressing neurons each of which corresponds to 1 type of instruction. i.e. I_1 for Decrement, I_2 for Clear, I_3 for Add, I_4 for Subtract, I_5 for Multiplicand, I_6 for Multiplier, I_7 for Dividend, I_8 for Divisor, I_9 for Increment. Some example use-cases: (1) Sending a spike to I_1 of an AM module will decrement by one the number store in the module, (2) Sending a spike to I_9 will increment by one the value in the module, (3) Sending spikes to neuron I_3 of two modules will trigger the modules to perform the 'Add' operation, (4) Sending a spike to I_7 of module A will trigger the 'Dividend' operation and sending a spike to I_8 to module B will trigger 'Divisor' operation. If configured correctly, together module A and module B will perform the actual division operation.

Any two modules can perform arithmetic operation together. This means the modules need to interact (exchange spikes) and so they need to know the 'address' of each other. i.e. To perform division, module A ('dividend' module) should know the address of module B ('divisor' module) and module B should also know the address of the module A. The addressing neurons are used to specify these addresses. Another use for the addressing neurons is to specify the neuron where the output of the operation (spikes) will be directed. Before instructing two AM modules to perform an arithmetic operation together, they should first be set-up by specifying the necessary addresses needed by the two modules.

The output neuron produces the spikes representing the result of the operation. The end signal neuron sends a spike to signal that the module has done the instructed operation. Where to direct the output of the end signal neuron can also be specified using the addressing neurons.

AM modules will be used in the succeeding sections.

3 Generating Context-Free Languages using SNPSP Systems

This section contains the main result of this work, a procedure for constructing an SNPSP system Π for a given *context-free* language L. Formally, we will prove Theorem 1. Parts of the proof will be given in Subsections 3.1-3.6. The proof will show how the SNPSP system Π is constructed.

Theorem 1. *If L is a context-free language over $T = \{t_1, t_2, \cdots, t_y\}$, then there is an SNPSP system Π, a morphism $h_1 : (T \cup \{c\})^* \to \{0, 1\}^*$, and a projection $h_2 : (T \cup \{c\})^* \to T^*$ such that $L = h_2({h_1}^{-1}(L(\Pi)))$*

Subsection 3.1 discusses context-free grammars in *Greibach normal form* (GNF) and an algorithm that uses the said grammar to generate strings. The algorithm uses a stack data structure and operations (push, pop) on the stack. The subsection also discusses how the algorithm's stack can be encoded as number (stack encoding) and how the push and pop operations can be mapped to arithmetic operations on the said number (stack encoding).

Subsections 3.2-3.6 describe how different SNPSP subsystem can be constructed to in order to implement the string generating algorithm described in Subsection 3.1. Subsection 3.3 shows how to construct an SNPSP subsystem that implements the arithmetic version of the pop operation. Subsection 3.4 shows how to construct an SNPSP subsystem that implements a compare operation needed by the algorithm. Subsection 3.5 shows how to construct an SNPSP subsystem that produces the output spike train and Subsection 3.6 shows how to construct an SNPSP subsystem that implements the arithmetic version of the push operation.

3.1 Encoding and Stack Operations

A language L is *context-free* if it can be generated by a *context-free* grammar. In this paper, we will use context-free grammars in *Greibach normal form*(GNF) [6]. A grammar G in GNF is $G = (N, T, S, P)$ where $N = \{n_1, n_2, \cdots, n_x\}$ is the set of *non-terminal* symbols, $T = \{t_1, t_2, \cdots, t_y\}$ is the set of *terminal* symbols, $S \in N$ is the *start* symbol, and P is the set of *production rules*. The production rules R_j $(1 \leq j \leq |P|)$ in P have the following form: $R_j : n_{r_j} \rightarrow t_{r'_j} N_j$ where $n_{r_j} \in N$, $t_{r'_j} \in T$, $N_j \in N^*$, $1 \leq r_j \leq x$, $1 \leq r'_j \leq y$.

SNPSP system Π will simulate how grammar G can generate the words in L. SNPSP system Π will use a stack data structure and simulate the algorithm below:

0. When generating a word w, we start with the non-terminal initial symbol S. This symbol is placed on the stack. We use the notation '$\overline{N}$' to represent the stack. $\overline{N}$ is a string over the non-terminal symbols, $\overline{N} \in N^*$, that represents the content of the stack. The first (left-most) symbol of $\overline{N}$ represents the symbol at the bottom of the stack while the last (right-most) symbol represents the symbol at the top of the stack. *i.e.* If $\overline{N} = abcde$, then the stack contains the 5 symbols a, b, c, d, e; a is the symbol on bottom of the stack while e is the symbol at the top of the stack.
1. 'Pop' (get and remove) the top symbol of the stack.
2. Check which production rules can be applied to the top symbol and non-deterministically select and apply one of those rules. The main form of production rule is "$n \rightarrow tN'$" where $n \in N$, $t \in T$, and $N' \in N^*$. If the top symbol is n_i, then any rule with $n = n_i$ can be applied. Applying the rule means to output the terminal symbol t and 'pushing' (inserting a symbol or multiple symbols to the stack) the symbols in N' to the stack. Since the stack is the string $\overline{N}$, 'pushing' a symbol a to the stack means appending a to $\overline{N}$ so the new value of $\overline{N}$ is $\overline{N}a$ ($\overline{N}_{new} = \overline{N}_{old}a$). *i.e.* Applying "$A \rightarrow aAB$" means the top symbol popped is the non-terminal A, the terminal symbol a will be output, and non-terminal symbols B and A will be 'pushed' to the stack, symbol B being pushed first. If N' is the empty string λ, then no non-terminal symbols will be pushed to the stack.
3. Steps 1 and 2 are counted as one iteration. Iterations will continue as long as the stack still contains non-terminal symbols. The algorithm stops when there are no more non-terminal symbols in the stack. The non-deterministically generated word w is the string of terminal symbols 'outputted' per iteration.

An example run of the algorithm is in Appendix A.1.

SNPSP systems only have a single type of object, the spike a, there is no way to represent the non-terminal symbols directly. A string s is encoded as the number $val_k(s)$ where k is the size of the alphabet of used by string s. Numbers can be represented in SNP and SNPSP systems as the number of spikes in the neurons. Using this encoding, we can store the stack data structure (a string) as the number of spike in a neuron.

Let $\overline{N} \in N^*$ be the string representing the stack. Let $N = \{n_1, ..., n_x\}$ be set of non-terminal symbols of grammar G. $\overline{N}$ is encoded as the number $val_x(\overline{N})$ which can be computed recursively as:

$$val_x(\overline{N}) = \begin{cases} (x+1)val_x(\overline{N}') + val_x(n_i) & \text{if } \overline{N} = \overline{N}' n_i,\ \overline{N'} \in N^+,\ n_i \in N \\ i & \text{if } \overline{N} = n_i \in N,\ 1 \leq i \leq x \\ 0 & \text{if } \overline{N} = \lambda \end{cases} \quad (1)$$

Alternatively, if $\overline{N} = n_{s_1} n_{s_2} n_{s_3} ... n_{s_l}$ where $n_{s_i} \in N$ and $1 \leq s_i \leq x$, $val_x(\overline{N})$ can be computed as :

$$val_x(\overline{N}) = \begin{cases} \sum_{j=1}^{l} val_x(n_{s_j})(x+1)^{l-j} & \text{if } \overline{N} \neq \lambda \text{ where } val_x(n_i) = i,\ n_i \in N, 1 \leq i \leq \\ 0 & \text{if } \overline{N} = \lambda \end{cases} \quad (2)$$

An example of how this encoding works is available in Appendix A.2.

The algorithm above involves two operations on the stack: popping and pushing of elements. Since the stack is represented as string $\overline{N}$, the operations of popping and pushing elements are actually operations on strings. *i.e.* concatenation of strings for the push operation and removal of a symbol from a string for the pop operation.

We can define the push and pop operations formally using 3 as functions p_1, p_2, and, p_3.

Function $p_1 : N^* \times N^* \to N^*$, where $N = \{n_1, ..., n_x\}$ is the set of non-terminal symbols, is the push function (or operation). It is defined as:

$$p_1(\overline{N}, N') = \begin{cases} p_1(p_1(\overline{N}, n_i), N'') & \text{if } N' = n_i N'',\ N' \in N^+,\ N'' \in N^+,\ n_i \in N \\ \overline{N} n_i & \text{if } N' = n_i \in N,\ 1 \leq i \leq x \\ \overline{N} & \text{if } N' = \lambda \end{cases} \quad (3)$$

$\overline{N} \in N^*$ is the stack. The push function p_1 recursively appends the symbols of $N' \in N^*$ to string $\overline{N}$. Simply, $p_1(\overline{N}, N') = \overline{N} N'$.

Function $p_2 : N^* \to N^*$, is one of the pop functions. $p_2(\overline{N})$ gives the top symbol (right-most) of stack $\overline{N}$.

$$p_2(\overline{N}) = n, \text{where } \overline{N} = N' n, N' \in N^*, n \in N \cup \{\lambda\} \quad (4)$$

Function $p_3 : N^* \to N^*$ is also one of the pop functions. $p_3(\overline{N})$ gives the substring (prefix) N' which is the string $\overline{N}$ except for the last (right-most) symbol.

$$p_3(\overline{N}) = \begin{cases} N' \text{ if } \overline{N} = N' n_i\ ,\ n_i \in N,\ N' \in N^* \\ \lambda \quad \text{if } \overline{N} = \lambda \end{cases} \quad (5)$$

Since in SNPSP systems, we will be using the val_x encoding of strings like $\overline{N}$ and $N' \in N^*$, we will create push and pop functions p_1', p_2', and p_3' that deal

with numbers (non-negative integers, we denote this set as $\mathbb{W}$ for *whole numbers*) instead of string.

The push function $p'_1 : \mathbb{W} \times \mathbb{W} \to \mathbb{W}$ corresponds to the push function p_1. If $\overline{N}$ is the stack string with encoding $val_x(\overline{N})$ and $N' \in N^*$ is some string with encoding $val_x(N')$, then p'_1 is defined as:

$$p'_1(val_x(\overline{N}), val_x(N')) = val_x(p_1(\overline{N}, N)) = val_x(\overline{N}N') \tag{6}$$

Push function p'_1 takes the encodings of 2 strings and returns the encoding of the concatenation of those 2 strings. We can directly use the encodings and express p'_1 without using the function p_1.

$$p'_1(val_x(\overline{N}), val_x(N')) = val_x(\overline{N})(x+1)^{|N'|} + val_x(N') \tag{7}$$

The expression on the right-hand side of Equation 7 is derived from how val_x encoding is calculated in Equation 1. An example usage of p'_1 is avaiable in Appendix A.3.

The pop function $p'_2 : \mathbb{W} \to \mathbb{W}$ corresponds to p_2. It takes the encoding of the stack string $\overline{N}$ and returns the encoding the top (right-most) symbol of the stack $\overline{N}$.

$$p'_2(val_x(\overline{N})) = \begin{cases} val_x(n_i) & \text{if } \overline{N} = N'n_i,\ N' \in N^*,\ n_i \in N \\ \lambda & \text{if } \overline{N} = \lambda \end{cases} \tag{8}$$

When $val_x(\overline{N})$ is written in base $x+1$, the right-most symbol of $\overline{N}$ corresponds to the least significant digit of $val_x(\overline{N})$. i.e. If $\overline{N} = ABCAA$ and its right-most symbol being A, then $val_3(ABCAA) = 12311_4$ and its least significant digit is 1. To get the least significant digit of $val_x(\overline{N})$ get the remainder when we divide the encoding with base $x+1$. With this, p'_2 can be rewritten as:

$$p'_2(val_x(\overline{N})) = val_x(\overline{N}) \mod (x+1) \tag{9}$$

The pop function $p'_3 : \mathbb{W} \to \mathbb{W}$ corresponds to p_3. It takes the encoding of the stack string $\overline{N}$ and returns the encoding the prefix substring of $\overline{N}$ excluding the last symbol.

$$p'_3(val_x(\overline{N})) = \begin{cases} val_x(N') & \text{if } \overline{N} = N'n_i,\ N' \in N^*,\ n_i \in N \\ \lambda & \text{if } \overline{N} = \lambda \end{cases} \tag{10}$$

$p'_3(val_x(\overline{N}))$ can be calculated by dividing it with the base $x+1$ (getting only the integer quotient). e.g. If $\overline{N} = ABCAA$ and $val_3(ABCAA) = 12311_4$, then $\lfloor 12311_4 / 10_4 \rfloor = 1231_4$ which is exactly the encoding $val_3(ABCA)$ for the string $ABCA$ which is $\overline{N}$ excluding the last symbol A.

$$p'_3(val_x(\overline{N})) = val_x(\overline{N})/(x+1) \tag{11}$$

Now that we have the encoding function val_x and stack functions p'_1, p'_2, p'_3 we can know use these function in the algorithm describe above that simulates string generation of any context-free grammar G in GNF. A full example of the

usage of $val_k(\overline{N})$ encoding and arithemtic stack operations p'_1, p'_2, p'_3 is available in Appendix A.4.

3.2 Constructing SNPSP System for CFL

We can now recreate algorithm in Section 3.1 that deals with numbers instead of strings. We will use the encoding function val_x and push and pop functions p'_1, p'_2, p'_3. Functions p'_1, p'_2, p'_3 involve arithmetic operations. We will use AM modules to implement p'_1, p'_2, p'_3 in the SNPSP system.

Recall that we will be simulating the string generation process of a GNF grammar $G = (N = \{n_1, ..., n_x\}, T = \{t_1, ..., t_y\}, S \in N, P)$ with P containing rules of the form "$R_j : n_{r_j} \rightarrow t_{r'_j} N_j$" where $n_{r_j} \in N$, $t_{r'_j} \in T$, $N_j \in N^*$, $1 \leq r_j \leq x$, $1 \leq r'_j \leq y$. We can look at a production rule R_j as having the 3 components n_{r_j}, $t_{r'_j}$, N_j. Using val_x encoding, the components n_{r_j}, N_j can be represented in the SNPSP system as the numbers $val_x(n_{r_j}), val_x(\rho(N_j))$ where $\rho(N_j)$ is the reverse of string N_j. We explained in Section 3.1 why we reverse the N_j of production rule R_j before getting its encoding.These numbers are stored in the system using AM modules. In the algorithm in Section 3.1, when rule R_j is used, the terminal symbol $t_{r'_j}$ is 'outputted' immediately. $t_{r'_j}$ does not need to be stored in the system as number but it will be represented as a neuron.

For each non-terminal symbol $n_i \in N$, there will be an AM module, module N_i, that stores the encoding $val_x(n_i)$. We can see these AM modules in Figure 3. For each terminal symbol $t_i \in T$, there will be a neuron, neuron t_i, that corresponds to the terminal symbol. These neurons are in Figure 4. For each rule $R_j \in P$, two AM modules, module N'_j and module L_j, will be added to the SNPSP system. Module N'_j stores the encoding of the reverse of string N_j of rule R_j. We use the function $\rho : N^* \rightarrow N^*$ to reverse a string s making its left-most symbol the right-most symbol of a new string s' and vice versa. We define ρ below:

$$\rho(N') = \begin{cases} n\rho(N'') & \text{if } N' = N''n,\ N'' \in N^+,\ n \in N \\ n & \text{if } N' = n \in N \\ \lambda & \text{if } N' = \lambda \end{cases} \tag{12}$$

Module N'_j stores the number $val_x(\rho(N_j))$ while module L_j stores the number $(x+1)^{|N_j|}$. These modules are shown in Figure 5. In equation 7, we can see how $val_x(\rho(N_j))$ (which is $val_x(N')$ in the equation 7) and $(x+1)^{|N_j|}$ (which is $(x+1)^{|N'|}$ in equation 7) are used by the push function p'_1 to perform the arithmetic equivalent of the push operation.

3.3 Constructing SNPSP Subsystem for the Pop Operation

Figure 2 is the SNPSP subsystem that performs the pop operations. There are 5 AM modules: module A, module B, module C, module D, and module G. In general (most of the time), module A stores the current encoding of the stack, $val_x(\overline{N})$. We use the variable W to mean actual number stored in module A

since during the operations, module A may contain the value that is not (yet) equal to $val_x(\overline{N})$ since the operation is still ongoing. Module B is used as a temporary number storage during the operation. We use the variable S to mean the number currently store in the module. Module C stores the number $x+1$ and is setup to perform the division process with module A. Module D also stores the number $x+1$ and is setup to perform the multiplication process with module B. Module G contains the address of module C, as the number C. Module G will give this address to module A before the division process between module A and module C. Module A, being the module that holds the stack encoding, also performs arithmetic operations with a module other than module C. So the address of module C is stored in module G.

Details of the SNPSP subsystem that are not available in Figure 2 are described below:

Neurons in Figure 2 with the single rule "$a \to a$": e_A, e'_A, e''_A, o'_A, o''_A, o'_B, o''_B, e_D, e'_D, e''_D, e_C, e_G, $d_1, ..., d_8$, d_x, d_y, d_z, $m_1, ..., m_9$.

Static synapses in the SNPSP subsystem in Figure 2: (k_i, o_A) where $1 \leq i \leq 7$, (q_i, o_B) where $1 \leq i \leq 4$, (d_x, a_i) where $1 \leq i \leq 7$, (d_y, a_i) where $1 \leq i \leq 7$, (d_z, a_i) where $1 \leq i \leq 7$,(a_i, m_j) where $1 \leq i \leq 7$, $1 \leq j \leq i+1$, (m_i, p'_{23}) where $1 \leq i \leq 8$.

Rules for o_A: $r_0 : a^2/a \to \pm1(o_A, \{o'_A\})$, $r_1 : a^3 \to \lambda$, $r_2 : a^5/a \to \pm1(o_A, \{o''_A\})$, $r_3 : a^6 \to \lambda$, $r_4 : a^8/a \to \pm1(o_A, \{o'''_A\})$, $r_5 : a^9 \to \lambda$.

Rules for o_B: $r_0 : a^2/a \to \pm1(o_B, \{o'_B\})$, $r_1 : a^3 \to \lambda$, $r_2 : a^5/a \to \pm1(o_B, \{o''_B\})$, $r_3 : a^6 \to \lambda$.

Rules in neurons $a_1, a_4, ..., a_7$: $r_1 : a^1 \to \lambda$, $r_2 : a^2 \to \lambda$, $r_3 : a^3 \to \lambda$, $r_4 : a^5 \to a$.

Rules in neurons a_2, a_3: $r_1 : a^1 \to \lambda$, $r_2 : a^2 \to \lambda$, $r_3 : a^3 \to \lambda$, $r_4 : a^5/a \to \lambda$, $r_5 : a^7 \to a$.

Rules for p'_{23}:

- $r_0 : a^1 \to \pm5(p'_{23}, \{I_{G,3}, d_1, d_x, d_y, d_z\})$
- $r_1 : a^2 \to \pm10(p'_{23}, \{I_{A,7}, I_{C,8}, k_1, k_2, k_3, k_4, d_2, d_x, d_y, d_z\})$
- $r_2 : a^3 \to \pm8(p'_{23}, \{I_{B,5}, I_{D,6}, k_1, k_2, q_1, d_3, d_x, d_y, d_z\})$
- $r_3 : a^4 \to \pm8(p'_{23}, \{I_{A,3}, k_1, q_1, q_2, d_4, d_x, d_y, d_z\})$
- $r_4 : a^5 \to \pm7(p'_{23}, \{I_{A,2}, k_1, k_2, d_5, d_x, d_y, d_z\})$
- $r_5 : a^6 \to \pm9(p'_{23}, \{I_{B,3}, q_1, q_2, q_3, q_4, d_6, d_x, d_y, d_z\})$
- $r_6 : a^7 \to \pm8(p'_{23}, \{I_{B,2}, A'_{A,4}, q_1, q_2, d_7, d_x, d_y, d_z\})$
- $r_7 : a^8 \to \pm1(p'_{23}, \{f\})$

Neuron p'_{23} contains 8 rules that correspond to 8 steps performed during the pop operations. The neuron is labeled as 'p'_{23}' since it essential simulates calculations of $p'_2(val_x(\overline{N}))$ and $p'_3(val_x(\overline{N}))$. The 8 steps performed are as follow:

Step 0: Load the address of module C in module A . This step is triggered by sending one spike to neuron p'_{23} which will then activate rule r_0. Plasticity rule r_0 will create synapses and send spike to neuron $I_{G,3}$ ('Add' instruction neuron I_3 of module G), and neurons d_1, d_x, d_y, d_z. Send a spike to neuron $I_{G,3}$ will trigger the 'Add' operation in the module. Since no other modules are activated

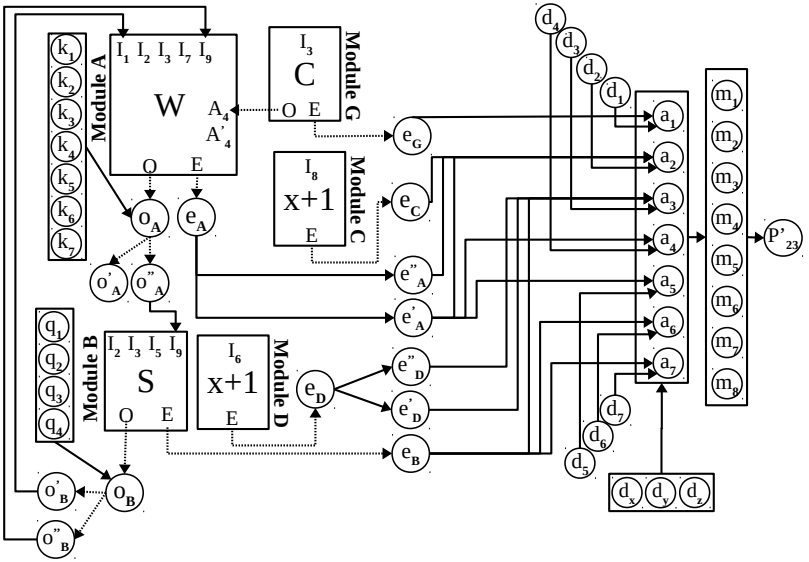


Fig. 2. SNPSP Subsystem for Performing 'Pop' Operations (p'_2, p'_3).

module G will simply send out C spikes via its output neuron O. Module G is setup to forward its output spikes to module A's neuron A'_4. After receive all C spikes from module G (C spike representing the address of module C), module A has now been setup and is ready to perform division with module C.

Spikes sent to neurons d_x, d_y, d_z are forwarded to neurons $a_1, ..., a_7$. The spike sent to neuron d_1 is forwarded to neuron a_1. Neurons $a_2, ..., a_8$ would receive 3 spikes each while neuron a_1 will receive 4 spikes. Neurons $a_2, ..., a_8$ would delete the 3 spikes they received using rule $r_3 : a^3 \rightarrow \lambda$. Neuron a_1 will keep its 4 spikes.

Neuron a_1 represents the next step, *step 1*. After performing the 'Add' operation, module G will send a spike to its end signal neuron E which will then be forwarded to neuron e_G. Neuron e_G will forward the spike to neuron a_1 which will change the neuron count from 4 spikes to 5 spikes and activating rule $r_4 : a^5 \rightarrow a$. Neuron a_1 will forward the spike to neurons m_1, m_2 and they will then forward the two spikes to neuron p'_{23}.

Step 1: Divide W **by** $x + 1$. After receiving 2 spikes from the previous step, neuron p'_{23} will apply the rule r_1, which will create synapses and send spikes to neurons $I_{A,7}, I_{C,8}, k_1, ...k_4, d_2, d_x, d_y, d_z$.

Module A has the value $W = val_x(\overline{N})$ while module C has the value $x + 1$. Sending spikes to neurons $I_{A,7}, I_{C,8}$ will trigger the 'Dividend' operation in module A and the 'Divisor' operation in module C which will start the division process between the two modules. The spikes sent to neurons $k_1, ..., k_4$ will be forwarded to neuron o_A which will now have 4 spikes. After this, a spike sent to neuron o_A will change the spike count from 4 to 5, activating rule

$r_2 : a^5 \to \pm 1(o_A, \{o''_A\})$. When neuron o_A has 4 spikes, a spike sent to the neuron is forwarded to neuron o''_A. Neuron o''_A is connected to the (increment) instruction neuron I_9 of module B. At this point, every spike sent out of module A (via output neuron O) increments the value S stored in module B. Initially, $S = 0$.

Spikes sent to neurons d_x, d_y, d_z are forwarded to neurons $a_1, ..., a_7$. The spike sent to neuron d_2 is forwarded to neuron a_2. Neurons $a_1, a_3, ..., a_7$ would receive 3 spikes each while neuron a_2 will receive 4 spikes. Neurons $a_1, a_3, ..., a_7$ would delete the 3 spikes they received using rule $r_3 : a^3 \to \lambda$. Neuron a_2 will keep its 4 spikes.

During the division process, module C will repeatedly perform the 'Divisor' operation which means it will send out spikes (possibly) multiple times via its end signal neuron E. The spike sent out by module C via neuron E is forwarded to neuron e_C which will then forward the spike to neuron a_2. This will change the spike count from 4 to 5 and will activate the rule $r_4 : a^5/a \to \lambda$ in neuron a_2. The spike count will return from 5 to 4. The division process ends when module A finishes the 'dividend' operation and send a spike to its neuron E. Neuron E will forward the spike to neuron e_A which will then forward the spike to neurons e'_A, e''_A. e'_A, e''_A will forward the two spikes to neuron a_2 changing the spike count from 4 to 6. Module C finishes its last 'divisor' operation and it will send a spike to its neuron E which will forward the spike to neuron e_C. e_C will forward the spike to neuron a_2 changing the spike count from 6 to 7 and activating the rule $r_5 : a^7 \to a$. Neuron a_2 will send spike to neurons m_1, m_2, m_3 which will then forward the 3 spikes to neuron p'_{23} starting *step 2.*

After division process, $W/(x+1)$ where $W = val_x(\overline{N}))$, between module A and module C, the value in module B should now be $S = W/(x+1)$ since all output spike sent out by module A is forwarded to module B's increment instruction neuron I_9. Module B now stores $S = val_x\overline{N}/(x+1)$ which is the value $p'_3(val_x(\overline{N}))$.

Step 2: Multiply S by $x+1$. After receiving 3 spikes from the previous step, neuron p'_{23} will apply the rule r_2, which will create synapses and send spikes to neurons $I_{B,5}, I_{D,6}, k_1, k_2, q_1, d_3, d_x, d_y, d_z$.

Module B has the value $S = val_x(\overline{N})/(x+1)$ while module D has the value $x+1$. Sending spikes to neurons $I_{B,5}, I_{D,6}$ will trigger the 'Multiplicand' operation in module B and the 'Multiplier' operation in module D which will start the multiplication process between the two modules. The spikes sent to neurons k_1, k_2 will be forwarded to neuron o_A which will change the spike count from 4 spikes (from step 1) to 6 spikes and will activate the rule $r_3 : a^6 \to \lambda$ (resetting the spike count to 0). The spike sent to neuron q_1 is forwarded to neuron o_B changing the spike count from 0 to 1. Any spike sent to neuron o_B from output neuron O of module B will be forwarded to neuron o'_B using the rule $r_0 : a^2/a \to \pm 1(o_B, \{o'_B\})$. Neuron o'_B will then forward the spike to module A's decrement instruction neuron I_1.

Spikes sent to neurons d_x, d_y, d_z are forwarded to neurons $a_1, ..., a_7$. The spike sent to neuron d_3 is forwarded to neuron a_3. Neurons $a_1, a_2, a_4..., a_7$ would receive 3 spikes each while neuron a_3 will receive 4 spikes. Neurons $a_1, a_2, a_4..., a_7$

would delete the 3 spikes they received using rule $r_3 : a^3 \to \lambda$. Neuron a_3 will keep its 4 spikes.

Spikes sent to neurons k_1, k_2 will be forwarded to neuron o_A changing the spike count from 4 spikes (from step 1) to 6 spikes. Rule $r_x : a^6 \to \lambda$ will be activated resetting the spike count in neuron o_A to 0.

During the division process, module B will repeatedly perform the 'Multiplicand' operation which means it will send out spikes (possibly) multiple times via its end signal neuron E. The spike sent out by module B via neuron E is forwarded to neuron e_B which will then forward the spike to neuron a_3. This will change the spike count from 4 to 5 and will activate the rule $r_4 : a^5/a \to \lambda$ in neuron a_3. The spike count will return from 5 to 4. The multiplication process ends when module D finishes the 'multiplier' operation and send a spike to its neuron E. Neuron E will forward the spike to neuron e_D which will then forward the spike to neurons e'_D, e''_D. e'_D, e''_D will forward the two spikes to neuron a_3 changing the spike count from 4 to 6. Module B finishes its last operation (which is actually the 'Add' operation instead the 'Multiplicand' operation) and it will send a spike to its neuron E which will forward the spike to neuron e_B. e_B will forward the spike to neuron a_3 changing the spike count from 6 to 7 and activating the rule $r_5 : a^7 \to a$. Neuron a_3 will send spike to neurons $m_1, ..., m_4$ which will then forward the 4 spikes to neuron p'_{23} starting *step 3*.

After multiplication process, $S \cdot (x+1)$ where $S = val_x(\overline{N})/(x+1)$, between module B and module D, the value in module A should now be $W = W-(S\cdot(x+1))$ where $W = val_x(\overline{N})$ since all output spike sent out by module B during the multiplication process is forwarded to module A's decrement instruction neuron I_1. Module A now stores $W = val_x(\overline{N}) - (val_x\overline{N}/(x+1)) \cdot (x+1) = val_x(\overline{N})$ mod $(x+1)$ which is the value $p'_2(val_x(\overline{N}))$.

Step 3: Copy the content of module A to other modules. After receiving 4 spikes from the previous step, neuron p'_{23} will apply the rule r_3, which will create synapses and send spikes to neurons $I_{A,3}, k_1, q_1, q_2, d_4, d_x, d_y, d_z$.

After step 2, module A now contains the value $W = p'_2(val_x(\overline{N}))$ (top symbol of the stack) while module B now contains the value $S = p'_3(val_x(\overline{N}))$ (the remaining stack when the top symbol is removed). Module A is the storage for the stack encoding so the current value S should be copied to module. But before we can copy S to module A, the top of the stack encoding W should be copied to other modules.

The spikes sent to neuron k_1 will be forwarded to neuron o_A changing the spike count from 0 to 1. Every spike sent to neuron o_A by output neuron O of module A will be forwarded to neuron o'_A using the plasticity rule $r_o : a^2/a \to \pm 1(o_A, \{o'_A\})$.

The spike sent to $I_{A,3}$ will trigger the 'Add' operation in module A. Since no other module is activated along with module A, module A will simply send out W spikes using its output neuron O. These spikes will be forwarded to neuron o_A which will forward it to neuron o'_A. In Figure 3, we can see neuron o'_A will forward the spikes to increment instruction neurons I_9 of modules $T_1, ..., T_x$.

Essentially, this step copies the value in module A which is the top symbol encoding $W = p'_2(val_x(\overline{N}))$ to x modules, modules $T_1, ..., T_x$.

Spikes sent to neurons q_1, q_2 will be forwarded to neuron o_B changing the spike count from 1 spike to 3 spikes. Neuron o_B will then activate the rule $r_1 : a^3 \to \lambda$ resetting the spike count of neuron o_B to 0.

Spikes sent to neurons d_x, d_y, d_z are forwarded to neurons $a_1, ..., a_7$. The spike sent to neuron d_4 is forwarded to neuron a_4. Neurons $a_1, a_2, a_3, a_5, a_6, a_7$ would receive 3 spikes each while neuron a_4 will receive 4 spikes. Neurons $a_1, a_2, a_3, a_5, a_6, a_7$ would delete the 3 spikes they received using rule $r_3 : a^3 \to \lambda$. Neuron a_4 will keep its 4 spikes.

After performing the 'Add' operation, module A will send a spike to neuron e_A which will forward the spike to neuron e'_A. Neuron e'_A will forward the spike to neuron a_4 changing the spike count from 4 to 5. Using the rule $r_4 : a^5 \to a$, neuron a_4 will send spikes to neurons $m_1, ..., m_5$ which then will forward them to neuron p'_{23}. Neuron p'_{23} having 5 spikes will activate rule $r_4 : a^5 \to \pm 7(p'_{23}, \{I_{A,2}, k_1, k_2, d_5, d_x, d_y, d_z\})$ which will lead to *step 4*.

Step 4: Clear the content of module A**.** After receiving 5 spikes from the previous step, neuron p'_{23} will apply the rule r_4, which will create synapses and send spikes to neurons $I_{A,2}, k_1, k_2, d_5, d_x, d_y, d_z$.

The spikes sent to neurons k_1, k_2 will be forwarded to neuron o_A changing the spike count from 1 spike (from the previous step) to 3 spikes. Neuron o_A will then activate rule $r_1 : a^3 \to \lambda$ resetting the spike count in the neuron to 0.

Spikes sent to neurons d_x, d_y, d_z are forwarded to neurons $a_1, ..., a_7$. The spike sent to neuron d_5 is forwarded to neuron a_5. Neurons $a_1, .., a_4, a_6, a_7$ would receive 3 spikes each while neuron a_5 will receive 4 spikes. Neurons $a_1, ..., a_4, a_6, a_7$ would delete the 3 spikes they received using rule $r_3 : a^3 \to \lambda$. Neuron a_5 will keep its 4 spikes.

The spike sent to neuron $I_{A,2}$ will activate the 'Clear' operation in module A which will reset the value W in module A to 0. At the end of the 'clear' operation in module A, the end signal neuron E of the module will send a spike to neuron e_A which will then forward it to neuron e'_A. Neuron e'_A will then send the spike neuron a_5 changing the spike count from 4 spikes to 5 spikes. Using rule $r_4 : a^5 \to a$, neuron a_5 will send spikes to neurons $m_1, ..., m_6$ which will then forward the spikes to neuron p'_{23}. Neuron p'_{23} having 6 spikes will activate rule $r_5 : a^6 \to \pm 9(p'_{23}, \{I_{B,3}, q_1, q_2, q_3, q_4, d_6, d_x, d_y, d_z\})$ which will lead to *step 5*.

Step 5: Copy the content $S = p'_3(val_x(\overline{N}))$ **of module** B **to module** A**.** After receiving 6 spikes from the previous step, neuron p'_{23} will apply the rule r_5, which will create synapses and send spikes to neurons $I_{B,3}, q_1, q_2, q_3, q_4, d_6, d_x, d_y, d_z$.

The spikes sent to neurons $q_1, ..., q_4$ will be forwarded to neuron o_B changing the spike count from 0 spike to 4 spikes. When neuron o_B has 4 spikes, a spike sent to the neuron by the output neuron O of module B will be forwarded by neuron o''_B using rule $r_2 : a^5/a \to \pm 1(o_B, \{o''_B\})$. Neuron o''_B will then forward the spikes it receive to the increment instruction neuron I_9 of module A.

Spikes sent to neurons d_x, d_y, d_z are forwarded to neurons $a_1, ..., a_7$. The spike sent to neuron d_6 is forwarded to neuron a_6. Neurons $a_1, .., a_5, a_7$ would receive 3 spikes each while neuron a_6 will receive 4 spikes. Neurons $a_1, ..., a_5, a_7$ would delete the 3 spikes they received using rule $r_3 : a^3 \to \lambda$. Neuron a_6 will keep its 4 spikes.

The spike sent to neuron $I_{B,3}$ will activate the 'Add' operation in module B. Since no other module is activated, module B will simply send out $S = p'_3(val_x(\overline{N}))$ spikes. The output neuron O of module B will send these S spikes to neuron o_B. After performing the 'Add' operation, the value in module A would have been incremented S times. In step 4, the value in module A was set to 0 and in this step the value was incremented S times. Also, at the end of the 'Add' operation, output neuron E of module B will send a spike to neuron e_B which will then forward the spike to neuron a_6 changing the spike count in neuron a_6 from 4 spikes to 5 spikes. Using rule $r_4 : a^5 \to a$, neuron a_6 will send spikes to neurons $m_1, ..., m_7$ which will then forward the spikes to neuron p'_{23}. Neuron p'_{23} having 7 spikes will activate rule $r_6 : a^7 \to \pm 7(p'_{23}, \{I_{B,2}, A'_{A,4}, q_1, q_2, d_7, d_x, d_y, d_z\})$ which will lead to *step 6*.

Step 6: Reset the content of module B ($S = 0$) and clear the address of module C set in module A. After receiving 7 spikes from the previous step, neuron p'_{23} will apply the rule r_6, which will create synapses and send spikes to neurons $I_{B,2}, A'_{A,4}, q_1, q_2, d_7, d_x, d_y, d_z$.

The spikes sent to neurons q_1, q_2 will be forwarded to neuron o_B changing the spike count from 4 spikes to 6 spikes. With 6 spikes, neuron o_B will activate the rule $r_3 : a^6 \to \lambda$ and reset the spike count to 0.

Spikes sent to neurons d_x, d_y, d_z are forwarded to neurons $a_1, ..., a_7$. The spike sent to neuron d_7 is forwarded to neuron a_7. Neurons $a_1, .., a_6$ would receive 3 spikes each while neuron a_7 will receive 4 spikes. Neurons $a_1, ..., a_6$ would delete the 3 spikes they received using rule $r_3 : a^3 \to \lambda$. Neuron a_7 will keep its 4 spikes.

The spike sent to addressing neuron $A'_{A,4}$ will clear the address of module C set in module A. The spike sent to clear instruction neuron $I_{B,2}$ will set the value S stored in the module to 0. After clearing the value in module A, it will send out a spike from its end signal neuron E to neuron e_B which will then forward the spike to neuron e'_B. Neuron e'_B will forward the spike to neuron a_7 changing the spike count from 4 spikes to 5 spikes. Neuron a_7 having 5 spikes will apply rule $r_4 : a^5 \to a$ sending spikes to neurons $m_1, ..., m_8$. Neurons $m_1, ..., m_8$ will forward the spikes to neuron p'_{23}. Neuron p'_{23} having 8 spikes will apply rule $r_7 : a^8 \to \pm 1(p'_{23}, \{f\})$ which will lead to *step 7*.

Step 7: Initiate the process of checking production rule can be applied. After receiving 8 spikes from the previous step, neuron p'_{23} will apply the rule r_7, which will create a synapse and a spike to neuron f. Sending spike to neuron f triggers the SNPSP subsystem in Figure 3. This subsystem is responsible for checking which production rule of the grammar is applicable given the encoding of the stack top symbol $p'_2(val_x(\overline{N}))$. In *step 3*, the calculate (encoding of) stack top symbol is stored in module A and is copied to modules $T_1, ..., T_x$ which are parts of the subsystem in Figure 3.

We summarize below the activities of the SNPSP subsystem in Figure 2. Given the following initial values in the modules: Module $A : W = val_x(\overline{N})$, Module B: $S = 0$, Module $C : x + 1$, Module $D : x + 1$, Module C: address of Module C, the subsystem will do the following steps/calculations:

0. Load the address of module C in module A.
1. $S \leftarrow W/(x + 1)$. This is $S = val_x(\overline{N})/(x + 1)$ which is $p'_3(val_x(\overline{N}))$. This is the encoding of the new stack with the top symbol removed.
2. $W \leftarrow W - (S \cdot (x + 1))$. This is $W = val_x(\overline{N}) - (val_x(\overline{N})/(x + 1))(x + 1) = val_x(\overline{N}) \mod (x + 1)$ which is $p'_2(val_x(\overline{N}))$. This is the encoding of the stack's top symbol.
3. $T \leftarrow W$. The top symbol encoding in module A is copied to modules $T_1, ..., T_x$ in Figure 3. Modules $T_1, ..., T_x$ always store the same value we denote as T.
4. $W \leftarrow 0$. Reset the value stored in module A to 0.
5. $W \leftarrow S$. The new stack encoding $S = p'_3(\overline{N})$ is copied to module A.
6. $S \leftarrow 0$. Reset the value stored in module B to 0.
7. Trigger the 'Compare' process in the subsystem in Figure 3.

Neuron p'_{23} contains all rules that trigger the steps listed above. The purpose of neurons $d_x, d_x, d_z, d_1, ..., d_7$, $a_1, ..., a_7, m_1, ..., m_8$ is to provide neuron p'_{23} the correct number of spikes at the proper time step in order for neuron p'_{23} to active rules $r_1, ..., r_7$ in order and at the correct time.

For each rule r_i $(1 \leq i \leq 7)$ in neuron p'_{23}, there is a corresponding neuron a_i such that when neuron a_i spikes, after two steps neuron p'_{23} would receive $i+1$ spikes and will activate rule r_i in the next time step. The purpose of neuron a_i is to receive spikes from the end signal neurons of the modules activated in the previous step. These 'end spikes' signal to neuron a_i that modules are done performing their operations and the subsystem can now proceed with step i. We say that neuron a_i is 'listening' to the end signals of the modules involve in step i-1. Rule r_i sends 4 spikes to neuron a_{i+1} via neuron d_x, d_y, d_z, d_{i+1} while the rest of neurons $a_{j\neq i+1}$ only receives 3 spikes via neurons d_x, d_y, d_z. For a neuron a_{i+1}, having 4 spikes means it is the neuron that is 'listening' for the end spikes from the active modules which signify the end of step i. The other neurons $a_{j\neq i+1}$ will delete any end spikes they receive from the active modules. Neuron $a_{j\neq i+1}$ having less than 4 spikes (i.e. 3 spikes received from d_x, d_y, d_z) means the neuron is not 'actively listening' and they are not task to trigger the next step (step i+1) of the process. Neurons $a_{j\neq i+1}$ will delete the end spikes they receive from the active modules.

Neuron o_A is used in the subsystem to direct the output spikes of module A. While neuron o_B is used to direct the output spikes of module B. This is done by sending the specified number of spikes to the neurons specifying the target neuron in which they will forward the next incoming spike. i.e. In step 1, 4 spikes are sent to neuron o_A to 'instruct' it to forwards the next spikes to neuron o''_A which is connected to neuron I_9 of module B. While in step 3, 1 spike is sent to neuron o_A to instruct it forwards the next spikes to neuron o'_A which is connected to neurons I_3 of modules $T_1, ..., T_x$ in Figure 3.

3.4 Constructing SNPSP Subsystem for the Compare Operation

During the pop operation performed by the subsystem in Figure 2, modules $T_1, ..., T_x$ received $p'_2(val_x(\overline{N}))$ spikes. Each module now contains the number encoding of the top symbol of the stack. The last rule, r_7, activated by neuron p'_{23} sends a spike to neuron f which activates the 'compare' operation performed by the subsystem in Figure 3. The 'compare' operation compares the encoding T of the top symbol stored in modules T_i to the encodings $val_x(n_i)$ of the non-terminal symbols (stored in modules N_i) to determine which $val_x(n_i)$ equals T or simply to determine what is the top symbol.

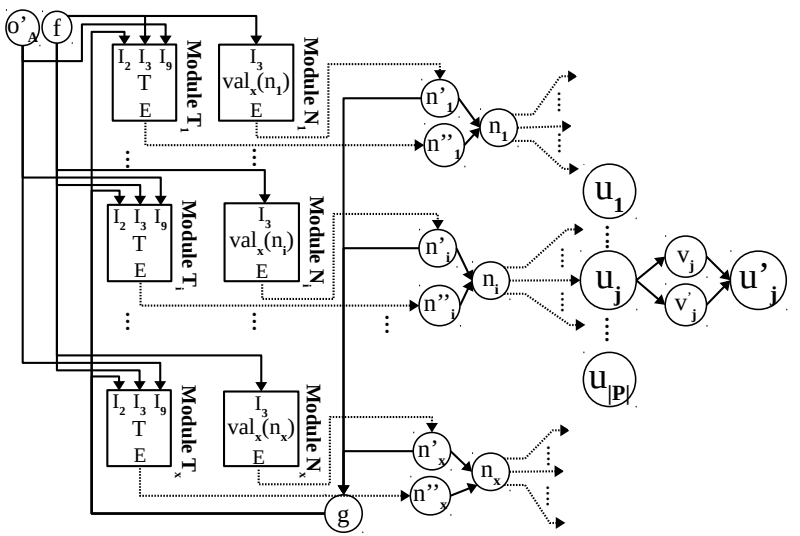


Fig. 3. SNPSP Subsystem for Determining Applicable Production Rules.

The comparison is done by sending a spike to the add instruction neuron I_3 of each module N_i and each module T_i. None of the modules are setup to actually perform binary addition. We can determine the top symbol by observing the end spikes of the modules.

Module T_i will be paired with module N_i. The end spike of module T_i will be received by neuron n''_i while the end spike of module N_i will be received by neuron n'_i. Since all modules will receive the spike through the add instruction neuron I_3 at the same time, then they will all perform the 'Add' operation at the same time. If module T_i and module N_i send out their end spikes at the same time, then we know that the encoding T of the top symbol is equal to $val_x(n_i)$ which means the popped top symbol is n_i. Otherwise, the popped top symbol is not n_i.

Neuron n_i', n_i'' will forward the spikes they receive, using rule $a \to a$, to neuron n_i. Neuron n_i receiving 2 spikes signifies that n_i is the popped top symbol of the stack. It means any production rule of the grammar that transforms the non-terminal symbol n_i can be activated. Any rule "$R_j : n_{r_j} \to t_{r_j'} N_j$" with $n_{r_j} = n_i$ (or when $r_j = i$) can be activated.

We note that the spikes sent to neurons n_i' are also forwarded to neuron g. Neuron g has the single rule $a^x \to a$ which is activated when neuron g receives x spikes. This will only occur when all modules N_i are done with the 'Add' operation. Neuron g will then forward the spike to all clear instruction neurons I_2 of modules T_i. This will clear the value of the modules setting $T = 0$ for all modules $T_1, ..., T_x$.

For each production rule R_j of the grammar, there is a corresponding neuron u_j. Neuron n_i has the following 2 rules of the "$r_1 : a \to \lambda$" and "$r_2 : a^2 \to \pm 1(n_i, W_i)$" where $W_i = \{u_j |$ neuron u_j corresponds to rule $R_j : n_{r_j} \to t_{r_j'} N_j$ and $n_{r_j} = n_i\}$. W_i is the set of neurons u_i corresponding to some rules R_j such that the non-terminal symbol n_{r_j} being transformed by the rule is n_i. Neuron n_i activates rule r_1 when it only receives one spike at a time. This is the case when $T \neq val_x(n_i)$. Neuron n_i activates rule r_2 when $T = val_x(n_i)$. When neuron n_i activates rule r_2, it will non-deterministically select one of the $u_j \in W_i$ neuron to connect and send a spike to. This means R_j of the grammar is selected and applied to non-terminal top symbol. This completes the 'compare' operation. The subsystem in Figure 3 was able to determine which of the non-terminal symbol n_i is the top symbol and non-deterministically select which of the applicable production rules R_j to apply by sending a spike to its corresponding neuron u_j.

3.5 Constructing SNPSP Subsystem for Output

The strings generated by grammar G are strings over the alphabet $T = \{t_1, ..., t_y\}$ of terminal symbols but the strings generated by SNPSP systems are strings over the binary alphabet $B = \{0, 1\}$. We use the morphism $h_1 : (T \cup \{c\})^* \to B^*$ and projection $h_2 : (T \cup \{c\})^* \to T^*$ to relate the strings generated by grammar G to the strings generated by the SNPSP system. We define h_1 as follows: $h_1(t_i) = 01^i0$ for $1 \leq i \leq y$, $h_1(c) = 0$. We define h_2 as follows: $h_2(t_i) = t_i$ for $1 \leq i \leq y$, $h_2(c) = \lambda$.

The SNPSP subsystem in Figure 3 determines which of the production rules can be applied and non-deterministically select one of the applicable production rule. When production rule is applied, the grammar being in GNF, one terminal symbol is generated. When production rule R_j is applied, its corresponding neuron u_j receives a spike. Neuron u_j has the single rule $a \to a$ and its only task is to forward the spikes it receive to connected neurons. In Figure 4, we can see that u_j is connected to a neuron t_i. Given rule $R_j : n_{r_j} \to t_{r_j'} N_j$, its corresponding neuron u_j is connected to t_i if $t_i = t_{r_j'}$ (or when $r_j' = i$).

All neurons in the SNPSP subsystem in Figure 4 has the single rule $a \to a$ except for neurons $u_1', ..., u_j', ..., u_{|P|}'$ and neuron z. Neuron z is the output neuron for the entire SNPSP system and has a synapse to the environment.

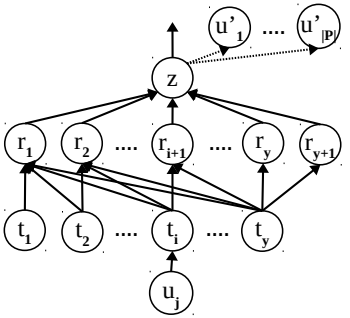


Fig. 4. SNPSP Subsystem for the String Output of the System.

When rule R_j is applied, neuron u_j will forward a spike to neuron $t_i = t_{r'_j}$ which will then forward the spikes to neurons r_k where $1 \leq k \leq i+1$. Neurons r_k $(1 \leq k \leq i+1)$ will forward their spikes to neuron z. Neuron z will receive $i+1$ spikes. Neuron z has the following rules: $r_1 : a(a^+)/a \rightarrow a$ and $r_2 : a \rightarrow \pm|P|(z, \{u'_1, ..., u'_{|P|}\})$. After receiving $i+1$ spike, neuron z will apply rule r_1 i times sending the spike train 1^i to the environment. After i steps, there will be a single spike in neuron z and the neuron will apply the rule r_2. In two steps, neuron z will connect and send spikes to neurons $u'_1, ..., u'_{|p|}$ then it will delete the created synapses. Neurons $u'_1, ..., u'_{|p|}$ would have received one spike each from neuron z.

Observing the spike train to the environment, when rule $R_j : n_{r_j} \rightarrow t_{r'_j} N_j$ is applied such that $t_{r'_j} = t_i$ (or $r'_j = i$) then substring "01^i0" is a generated. The full string generated by the system is composed of substrings of the form "01^i0" which are separated by substrings of the form "0^j". If the SNPSP system was able to apply n productions before halting, then there will be n substrings of the form "01^i0" generated. The form of the full string generated by the system is: $S = \ 0^{j_1} \mid 01^{i_1}0 \mid 0^{j_2} \mid 01^{i_2}0 \mid \cdots \mid 01^{i_n}0 \mid 0^{j_{n+1}}$.

The substrings "01^i0" are generated when rules are applied and the 'output' SNPSP subsystem in Figure 4 is activated. Substrings "0^j" are generated when the other SNPSP subsystems in Figures 2, 3, and 5 are active which means output neuron z in the 'output' subsystem is not sending spikes to the environment.

Applying inverse morphism h_1^{-1} to S will give the string $S' = h_1^{-1}(S)$: $S' = c^{j_1} \mid t_{i_1} \mid c^{j_2} \mid t_{i_2} \mid \cdots \mid t_{i_n} \mid c^{j_{n+1}}$. Then applying morphism h_2 to S' will give the string $S'' = h_2(S')$: $S'' = \ t_{i_1} \mid t_{i_2} \mid \cdots \mid t_{i_n}$. The string generated by grammar G is $S'' = h_2(h_1^{-1}(S))$ where S is the string generated by the SNPSP system.

3.6 Constructing SNPSP Subsystem for the Push Operation

When rule $R_j : n_{r_j} \rightarrow t_{r'_j} N_j$ is applied, aside from generating the substring "$01^{r'_j}0$" that corresponds to the terminal symbol $t_{r'_j}$, the string N_j should be 'pushed' to the stack. From Section 3.1, we explained that the reverse of N_j, denote as $\rho(N_j)$ in Section 3.2, is the one pushed to the stack. In equation 7, we show how the new stack encoding $p'_1(val_x(\overline{N}), val_x(\rho(N_j)))$ is calculated using the values $val_x(\overline{N})$, $val_x(\rho(N_j))$ and $(x+1)^{|N_j|}$. The process of calculating $p'_1(val_x(\overline{N}), val_x(\rho(N_j)))$ is the arithmetic version of string operation of pushing symbols to the stack.

The SNPSP subsystem in Figure 5 performs the arithmetic operations using AM modules in order to calculate $p'_1(val_x(\overline{N}), val_x(\rho(N_j)))$. Module A, the storage of the stack encoding $val_x(\overline{N})$, and module B, a module used for temporary storage, are reused in this 'push' SNPSP subsystem. Aside from the two main modules A and B the following modules are part of the 'push' SNPSP subsystem: Modules N'_j store the values $val_x(\rho(N_j))$, Modules L_j store the values $(x+1)^{|N_j|}$, Module E will store the value B which will be a copy of a specific value $val_x(\rho(N_j))$ stored in some Module N'_j, Module F will store the value D which will be a copy of a specific value $(x+1)^{|N_j|}$ stored in some module L_j, Module H stores the address of module E.

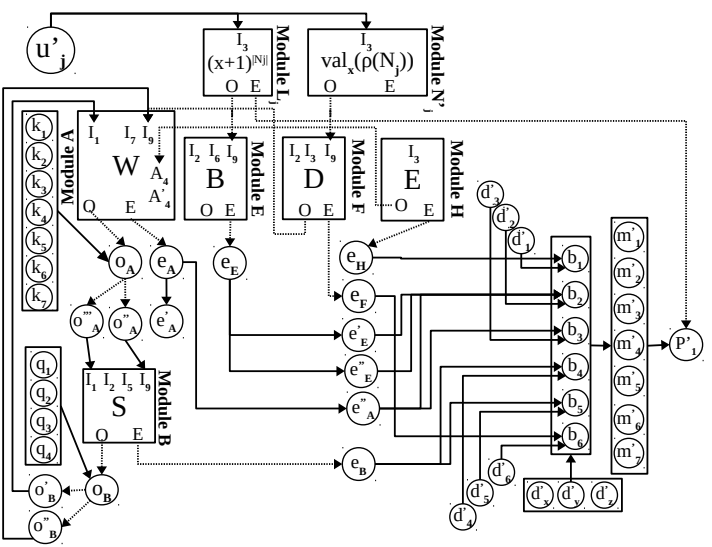


Fig. 5. SNPSP Subsystem for Performing the 'Push' Operation (p'_1).

The details about the subsystem that are not visible in Figure 5 and are not yet detailed Section 3.3 are shown below:

Neurons while the single rule $a \to a$: e_E, e'_E , e''_E, e_F, e_H.

Static synapses in the SNPSP subsystem in Figure 5: (d'_x, b_i) where $1 \leq i \leq 6$, (d'_y, b_i) where $1 \leq i \leq 6$, (d'_z, b_i) where $1 \leq i \leq 6$, (d'_i, b_i) where $1 \leq i \leq 6$, (b_i, m'_j) where $1 \leq i \leq 6$, $1 \leq j \leq i+1$, (m'_i, p'_1) where $1 \leq i \leq 6$.

Rules for p'_1: $r_0 : a^1 \to \pm 4(p'_1, \{I_{H,3}, d'_1, d'_x, d'_y, d'_z\})$, $r_1 : a^2 \to \pm 10(p'_1, \{I_{A,5}, I_{E,6}, k_1, k_2, k_3, k_4, d'_2, d'_x, d'_y, d'_z\})$, $r_2 : a^3 \to \pm 8(p'_1, \{I_{A,3}, k_5, k_6, k_7, d'_3, d'_x, d'_y, d'_z\})$, $r_3 : a^4 \to \pm 9(p'_1, \{I_{B,3}, q_1, q_2, q_3, q_4, d'_4, d'_x, d'_y, d'_z\})$, $r_4 : a^5 \to \pm 7(p'_1, \{I_{B,2}, I_{E,2}, I_{F,3}, d'_5, d'_x, d'_y, d'_z\})$, $r_5 : a^6 \to \pm 9(p'_1, \{A'_{A,4}, I_{F,2}, k_1, k_2, q_1, q_2, d'_6, d'_x, d'_y, d'_z\})$, $r_6 : a^7 \to \pm 1(p'_1, \{P'_{23}\})$

Similar to the 'pop' SNPSP subsystem in Figure 2, the 'push' subsystem will perform the computation of $p'_1(val_x(\overline{N}), val_x(\rho(N_j)))$ in steps, 7 steps specifically. Neuron p'_1 has the 7 rules $r_0, ..., r_6$ that trigger each of the step. Rule r_i triggers the i^{th} step. Similar to the rules in neuron p'_{23} in the 'pop' subsystem, the rules in p'_1 are used to (1) trigger the necessary arithmetic operations by sending spikes instruction neurons of the modules, (2) direct the output spikes of the modules (specifically modules A or B) by sending spikes to specific neurons in $\{k_1, ..., k_7, q_1, ..., q_4\}$, (3) prepare for the next step, e.g. step $i+1$ by sending spikes to neurons $d'_x, d'_y, d'_z, d'_{i+1}$.

Neurons $d'_1, ..., d'_6, d'_x, d'_y, d'_z, b_1, ..., b_6, m'_1, ..., m'_7$ in the 'push' subsystem have the same purpose as neurons $d_1, ..., d_7, d_x, d_y, d_z, a_1, ..., a_7, m_1, ..., m_8$ in the 'pop' subsystem. Neurons $d'_1, ..., d'_6, d'_x, d'_y, d'_z,\ b_1, ..., b_6, m'_1, ..., m'_7$ work together to provide the correct number of spikes to neuron p'_1 at the proper time step in order for neuron p'_1 to activate rules $r_1, ..., r_6$ in order and in the correct time.

We summarize below the activities of the SNPSP subsystem in Figure 5. Given the following initial values in the modules: Module $A : W = val_x(\overline{N})$, Module B: $S = 0$, Module $N'_j : val_x(\rho(N_j))$, Module $L_j : (x+1)^{|N_j|}$, Module E : $B = 0$, Module $F : D = 0$, Module H : address of module E, the subsystem will do the following steps or calculations:

-4. In Figure 3, when rule R_j is activated a spike is sent to its corresponding neuron u_j. Neuron u_j forwards the spike to neurons v_j, v'_j which then forward the spikes to neuron u'_j. Neuron u'_j has now 2 spikes.
-3. Neuron u_j also sends a spike to some neuron t_i to trigger the 'output' operation described in Section 3.5. At the end of the output operation neuron z in Figure 4 will use is rule r_2 and send spikes to all neurons u'_j. Neuron u'_j has only two rules $r_1 : a \to \lambda$ and $r_2 : a^3 \to a$. The specific neuron u'_j that previously received 2 spikes will not have 3 spikes.
-2. This neuron u'_j will send spikes to the add instruction neurons I_3 of modules L_j and N'_j. Module L_j will send out $(x+1)^{|N_j|}$ spikes to the increment instruction I_9 of module E so the variable $B = (x+1)^{|N_j|}$. Module N'_j will send out $val_x(\rho(N_j))$ spikes to the increment instruction I_9 of module F so the variable $D = val_x(\rho(N_j))$.
-1. The end spike of module L_j will be sent to neuron p'_1 triggering start of the 7 steps for computing $p'_1(val_x(\overline{N}), val_x(\rho(N_j)))$.
0. Load the address of module E to module A
1. $S \leftarrow W \cdot B$. This means $S = val_x(\overline{N}) \cdot (x+1)^{|N_j|}$.

2. $S \leftarrow S - W$. This means $S = val_x(\overline{N}) \cdot (x+1)^{|N_j|} - val_x(\overline{N})$.
3. $W \leftarrow W + S$. This means $W = val_x(\overline{N}) + val_x(\overline{N}) \cdot (x+1)^{|N_j|} - val_x(\overline{N})$. Or $W = val_x(\overline{N}) \cdot (x+1)^{|N_j|}$
4. $W \leftarrow W + D$, $S \leftarrow 0$, $B \leftarrow 0$. For W this means $W = val_x(\overline{N}) \cdot (x+1)^{|N_j|} + val_x(\rho(N_j))$.
5. $D \leftarrow 0$, clear address of module E set in module A.
6. Send a spike to neuron p'_{23} which will trigger the 'pop' operation after this 'push' operation.

Steps -3 to -1 are triggered by a spike sent to neuron u_j at the end of the 'compare' operation and a spike sent to neuron u'_j at the end of the 'output' operation. Steps 0 to 6 are triggered by the rules $r_0, ..., r_6$ in neuron p'_1. After the 'push' operation, module A now stores $W = val_x(\overline{N}) \cdot (x+1)^{|N_j|} + val_x(\rho(N_j))$ which is exactly the value of $p'_1(val_x(\overline{N}), val_x(\rho(N_j)))$. Module B now stores $S = 0$, module E now stores $B = 0$, module F now stores $D = 0$.

The entire system halts and stops generating symbols when the stack encoding reaches 0. This finishes the proof of Theorem 1.

4 Conclusions

To generate context-free languages, we created a procedure that will create an SNPSP system that simulates the way a context-free grammar in Greibach normal form generates strings using a stack algorithm. The stack algorithm involves symbols being pushed to or popped from the stack. Initially, we used a string to represent the stack and we define string operations to represent both push and pop stack operations. We then use an encoding to convert the stack string to a whole number so it can be stored in a single neuron (in an AM module). Corresponding functions on numbers were created to represent the string push and pop operations. The stack algorithm can now be implemented using AM modules since we have converted the stack to a number and the stack operations to operations on numbers.

By using a different encoding on the string and different numerical operations on the number (stack), one can create a variant of this procedure for creating the SNPSP system. It is possible to encode a string to a number using only multiplication as operation. This will result to the push and pop operation being converted to numerical operations that only involve multiplication and division. The resulting SNPSP system will probably have a different number of AM modules used.

Acknowledgements

F.G.C. Cabarle is supported in part by an RLC 2018–2019 grant from the OVCRD of UP Diliman, and the Dean Ruben A. Garcia PCA2018–2019.

R.T.A. de la Cruz is supported by Engineering Research and Development for Technology (ERDT) scholarship program of Department of Science and Technology - Science Education Institue (DOST-SEI Philippines).

H. Adorna is supported in part by ERDT-DOST, OVCRD - RLC and Semirara Mining Corp Professorial Chair for Computer Science.

References

1. Cabarle, F.G.C., de la Cruz, R.T.A., Zhang, X., Jiang, M., Liu, X., Zeng, X.: On string languages generated by spiking neural p systems with structural plasticity. IEEE Transactions on NanoBioscience **17**(4), 560–566 (Oct 2018). https://doi.org/10.1109/TNB.2018.2879345
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. Cabarle, F.G.C., de la Cruz, R.T.A., Zeng, X.: Arithmetic and memory modules using spiking neural p systems with structural plasticity. (Submitted)
4. Chen, H., Freund, R., Ionescu, M., Păun, Gh., Pérez-Jiménez, M.J.: On String Languages Generated by Spiking Neural P Systems. Fundam. Inform. **75**(1-4), 141–162 (2007)
5. Chen, H., Ionescu, M., Ishdorj, T.O., Păun, A., Păun, Gh., Pérez-Jiménez, M.J.: Spiking Neural P Systems with Extended Rules: Universality and Languages. Natural Computing **7**(2), 147–166 (2008)
6. Greibach, S.A.: A new normal-form theorem for context-free phrase structure grammars. J. ACM **12**(1), 42–52 (Jan 1965). https://doi.org/10.1145/321250.321254, `http://doi.acm.org/10.1145/321250.321254`
7. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking Neural P Systems. Fundam. Inf. **71**(2,3), 279–308 (Feb 2006)
8. Jiang, K., Chen, W., Zhang, Y., Pan, L.: On string languages generated by sequential spiking neural P systems based on the number of spikes. Natural Computing **15**(1), 87–96 (2016). https://doi.org/10.1007/s11047-015-9514-5
9. Kong, Y., Zhang, Z., Liu, Y.: On string languages generated by spiking neural P systems with astrocytes. Fundamenta Informaticae **136**(3), 231–240 (2015)
10. Kong, Y., Zhang, Z., Liu, Y.: On String Languages Generated by Spiking Neural P Systems with Astrocytes, pp. 225–229. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
11. Krithivasan, K., Metta, V.P., Garg, D.: On String Languages Generated by Spiking Neural P Systems with Anti-Spikes. International Journal of Foundations of Computer Science **22**(01), 15–27 (2011)
12. Pan, L., Păun, G.: Spiking neural P systems with anti-spikes. International Journal of Computers, Communications and Control **4**(3), 273–282 (2009)
13. Pan, L., Wang, J., Hoogeboom, H.: Spiking neural P systems with astrocytes. Neural Computation **24**(3), 805–825 (2012)
14. 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 (Sept 2017). https://doi.org/10.1109/TNB.2017.2722466
15. Pan, L., Zeng, X., Zhang, X., Jiang, Y.: Spiking neural P systems with weighted synapses. Neural Processing Letters **35**(1), 13–27 (2012). https://doi.org/10.1007/s11063-011-9201-1
16. Pan, L., Păun, G., Zhang, G., Neri, F.: Spiking neural p systems with communication on request. International Journal of Neural Systems **27**(08), 1750042 (dec

2017). https://doi.org/10.1142/s0129065717500423, https://doi.org/10.1142/s0129065717500423
17. Păun, G.: Spiking neural P systems with astrocyte-like control. Journal of Universal Computer Science **13**(11), 1707–1721 (2007)
18. Păun, G.: Computing with membranes. J. Comput. Syst. Sci. **61**(1), 108–143 (Aug 2000)
19. Păun, Gh., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)
20. Song, T., Pan, L.: Spiking neural P systems with rules on synapses working in maximum spikes consumption strategy. IEEE Transactions on Nanobioscience **14**(1), 37–43 (2015)
21. Song, T., Pan, L., Păun, G.: Spiking neural P systems with rules on synapses. Theoretical Computer Science **529**, 82–95 (2014). https://doi.org/10.1016/j.tcs.2014.01.001
22. Wang, J., Hoogeboom, H., Pan, L., Păun, G., Pérez-Jiménez, M.: Spiking neural P systems with weights. Neural Computation **22**(10), 2615–2646 (2010)
23. Wu, T., Pun, A., Zhang, Z., Pan, L.: Spiking neural p systems with polarizations. IEEE Transactions on Neural Networks and Learning Systems **29**(8), 3349–3360 (Aug 2018). https://doi.org/10.1109/TNNLS.2017.2726119
24. Wu, T., Zhang, Z., Pan, L.: On languages generated by cell-like spiking neural P systems. IEEE Transactions on Nanobioscience **15**(5), 455–467 (2016)
25. Wu, T., Bîlbîe, F.D., 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 (oct 2018). https://doi.org/10.1142/s0129065718500132, https://doi.org/10.1142/s0129065718500132
26. Wu, T., Zhang, Z., Pun, G., Pan, L.: Cell-like spiking neural p systems. Theor. Comput. Sci. **623**(C), 180–189 (Apr 2016). https://doi.org/10.1016/j.tcs.2015.12.038, https://doi.org/10.1016/j.tcs.2015.12.038
27. X. Zhang and L. Pan and A. Păun: On the universality of axon p systems. IEEE Transactions on Neural Networks and Learning Systems **26**(11), 2816–2829 (Nov 2015). https://doi.org/10.1109/TNNLS.2015.2396940
28. Zeng, X., Zhang, X., Song, T., Pan, L.: Spiking neural p systems with thresholds. Neural Computation **26**(7), 1340–1361 (July 2014)
29. Zhang, X., Zeng, X., Pan, L.: On string languages generated by spiking neural p systems with exhaustive use of rules. Natural Computing **7**(4), 535–549 (2008)
30. Zhang, X., Zeng, X., Pan, L.: On languages generated by asynchronous spiking neural P systems. Theoretical Computer Science **410**(26), 2478–2488 (2009). https://doi.org/10.1016/j.tcs.2008.12.055

A Appendix

A.1 String Generation - Example

Example: $G = (N = \{A, B\}, T = \{a, b\}, A, P)$ where P contains the following rules: $R_1 : A \rightarrow aAB$, $R_2 : B \rightarrow bAB$, $R_3 : A \rightarrow a$, $R_4 : B \rightarrow b$.

The table below shows how the word "*aabab*" is generated by the algorithm using grammar G.

Iteration	Stack ($\overline{N}$)	Applied Rule	Output Symbol	New Stack
1	A	$R_1 : A \rightarrow aAB$	a	BA
2	BA	$R_3 : A \rightarrow a$	a	B
3	B	$R_2 : B \rightarrow bAB$	b	BA
4	BA	$R_3 : A \rightarrow a$	a	B
5	B	$R_4 : B \rightarrow b$	b	λ

Table 1. String Production for *aabab*.

In iteration 1, $\overline{N} = A$, and the popped top symbol is A which means rules R_1, R_3 can be applied. After popping the top symbol, the stack is now $\overline{N} = \lambda$. R_1 is non-deterministically selected and applied. Applying R_1 will output the terminal symbol a and will push the symbol B and then symbol A to the stack which will result to $\overline{N} = BA$. The rest of the iterations are shown above. Each iteration performs steps 1 and 2 of the algorithm. For each iteration, there is an output terminal symbol. Looking at the 'Output Symbol' column in Table 1, you can see that the word "*aabab*" is generated by the algorithm.

A.2 Encoding of the Stack String - Example

$N = \{n_1 = A, n_2 = B, n_3 = C\}$, $x = |N| = 3$, $val_3(n_i) = i$ for $1 \leq i \leq 3$.

$\overline{N}$ (Stack)	val_3 Computation (Base 10)	$val_3(\overline{N})$ (Base 4)	$val_3(\overline{N})$ (Base 10)
A	$val_3(A) = 1$	$(1)_4$	$(1)_{10}$
B	$val_3(B) = 2$	$(2)_4$	$(2)_{10}$
C	$val_3(C) = 3$	$(3)_4$	$(3)_{10}$
ABC	$val_3(ABC) = 1(4)^2 + 2(4)^1 + 3(4)^0$	$(123)_4$	$(27)_{10}$
AA	$val_3(AB) = 1(4)^2 + 1(4)^0$	$(11)_4$	$(5)_{10}$
CAB	$val_3(CAB) = 3(4)^2 + 1(4)^1 + 2(4)^0$	$(312)_4$	$(54)_{10}$
CCBBA	$val_3(CAB) = 3(4)^4 + 3(4)^3 + 2(4)^2 + 2(4)^1 + 1(4)^0$	$(33221)_4$	$(1001)_{10}$

Table 2. String Encoding Examples

A.3 Arithmetic Version of Push Operation (or String Concatenation) - Example

We can use strings in Table 2. The size of the alphabet N is $x = 3$, string ABC is encoded as 123_4 (in base $x+1 = 4$) and string AA is encoded as 11_4. If we let $\overline{N} = ABC$ and $N' = AA$, then $\overline{N}N' = ABCAA$. If we encode $\overline{N}N' = ABCAA$ as described in Equation 2, then (note: $x + 1 = 4_{10}$ or $x + 1 = 10_4$)

$$val_3(ABCAA)_{10} = val_3(A)\cdot 4^4 + val_3(B)\cdot 4^3 + val_3(C)\cdot 4^2 + val_3(A)\cdot 4^1 + val_3(A)\cdot 4^0$$

$$val_3(ABCAA)_{10} = 1 \cdot 4^4 + 2 \cdot 4^3 + 3 \cdot 4^2 + 1 \cdot 4^1 + 1 \cdot 4^0 = 437_{10}$$

The encoding $val_3(ABCAA)_{10} = 437_{10}$ is calculated using base 10. In base $x+1 = 4$, it is written as 12311_4. It is easier to visualize how the encoding works when the result is written in base $x+1$ since for every symbol in the string there is a corresponding 'digit' in the encoding when written in base $x + 1$. In base $x+1 = 4$, the resulting encoding $val_3(ABCAA)$ is the concatenation of the digits of $val_3(ABC) = 123_4$ and digits of $val_3(AA) = 11_4$ which is 12311_4. But since concatenation is an operation on strings and the encodings are numbers, we use the Equation 7 for p'_1 to define arithmetically this digit concatenation process. Using p'_1, $val_3(ABCAA)$ can be calculated as $p'_1(val_3(ABC), val_3(AA))$:

$$p'_1(val_3(ABC), val_3(AA)) = val_3(ABC)(10_4)^{|AA|} + val_x(AA)$$

$$p'_1(val_3(ABC), val_3(AA)) = 123_4 \cdot (10_4)^{2_4} + 11_4 = 12300_4 + 11_4 = 12311_4$$

A.4 String Generation using Arithmetic Version of Stack Operations - Example

Let $G = (N = \{A, B, C\}, T = \{a, b, c\}, A, P)$ where P contains the following rules: $R_1 : A \to aABB$, $R_2 : B \to bBBC$, $R_3 : A \to a$, $R_4 : B \to b$, $R_5 : C \to c$.

The string $aabbbcb$ is generated by grammar G as follows:

$A \xrightarrow{R_1} aABB \xrightarrow{R_3} aaBB \xrightarrow{R_2} aabBBCB \xrightarrow{R_4} aabbBCB \xrightarrow{R_4} aabbbCB \xrightarrow{R_5}$ $aabbbcB \xrightarrow{R_4} aabbbcb$

Starting form the axiom symbol A, one applicable rule is non-deterministically selected and applied to the first (left-most) non-terminal symbol. One rule is applied at any given time. Since grammar G in Greibach normal form, when a rule is applied only one terminal symbol is generated at a time. The string is generated from left to right.

We note a difference in conventions used in the string generation above and the string generation using the stack algorithm. In string generation above, for example in the step: $aaBB \xrightarrow{R_2} aabBBCB$, the terminal symbols are generated from left to right, applying the rules first on the left-most non-terminal symbol which in this step is B. When R_2 is applied to B, terminal symbol b is generated and non-terminal symbols BBC are generated. In the stack algorithm, stack

string $\overline{N}$ stores the non-terminal symbols. In this step, $\overline{N} = BB$ but the top of the stack is the right-most symbol of $\overline{N}$. When R_2 is applied, $\overline{N} = B$ since the top symbol B was popped, and the string CBB is pushed to the stack making the stack string $\overline{N} = BCBB$. The stack algorithm also generates the string from left to right similar to the normal derivation, the only difference is on the convention of how the stack string is written. The left-most non-terminal symbol in the generated string corresponds to the top (right-most) symbol of the stack.

This convention in writing the stack string is used so when the stack string is encoded as a number the top symbol of the stack will be the least significant digit of the encoded number. i.e. stack string $\overline{N} = ABCABC$, $val_3(ABCABC) = 123123_4$, the top of symbol of the stack is C, and the top symbol encoding is the least significant digit 3_4. The least significant digit of $val_x(\overline{N})$ is easier to calculate compared to its most significant digit.

Table 3 shows the 7 iterations of the algorithm when generating the string *aabbbcb*. The third column shows the encoding $val_3(\overline{N})$ in base $x + 1 = 4$ of the stack string $\overline{N}$. The value of $p'_2(val_3(\overline{N}))$ is the least significant digit of $val_3(\overline{N})$ which corresponds to the top symbol of the stack. The value of $p'_3(val_3(\overline{N}))$ is the encoding of the new stack after removing the top symbol of the stack. N' is the reverse string of non-terminal symbols of a production rule. e.g. For $R_1 : A \rightarrow aABB$, $N' = BBA$ and for $R_2 : B \rightarrow bBBC$, $N' = CBB$. Noting again, that N' is the reverse of the actual non-terminal substring because of the convention used in the stack algorithm (right-most symbols being first to be processed, top of the stack). $val_3(N')$ is the encoding of N'. $val_3(N')$ is that is pushed, using p'_1 to the new stack $p'_3(val_3(\overline{N}))$. The resulting stack is $p'_1(p'_3(val_3(\overline{N})), val_3(N'))$. Column 6 shows the generated terminal symbol per iteration.

No.	$\overline{N}$	$val_3(\overline{N})$	$p'_2(val_3(\overline{N}))$	Rule	Output	$p'_3(val_3(\overline{N}))$	$val_3(N')$	$p'_1(p'_3(val_3(\overline{N})), val_3(N'))$
1	A	1_4	1_4	R_1	a	0_4	221_4	221_4
2	BBA	221_4	1_4	R_3	a	22_4	0_4	22_4
3	BB	22_4	2_4	R_2	b	2_4	322_4	2322_4
4	BCBB	2322_4	2_4	R_4	b	232_4	0_4	232_4
5	BCB	232_4	2_4	R_4	b	23_4	0_4	23_4
6	BC	23_4	3_4	R_5	c	2_4	0_4	2_4
7	B	2_4	2_4	R_4	b	0_4	0_4	0_4

Table 3. String Generation for *aabbbcb*

Improved Hybrid Particle Swarm Optimization for Image Segmentation

Shuo Liu[1], Kang Zhou[1]*, Shan Zeng[1], Huaqing Qi[2], and Jiangrong Liu[1]

1.Department of Math and Computer, Wuhan Polytechnic University, Wuhan 430023, Hubei, China 2.Department of Economics and Management, Wuhan Polytechnic University, Wuhan 430023, Hubei, China
{liushuo1979@hotmail.com,zhoukang_wh@163.com,
zengshan1981@whpu.edu.cn,qihuaqing@sohu.com,
liujiangrong@163.com}

Abstract. A method for image segmentation based on improved hybrid particle swarm optimization (PSO) is proposed. In view of the traditional PSO algorithm is easy to fall into local optimal solution, we update the particle velocity based on the combination of global optimization, region equilibrium and compression factor. By this way, the search ability of the particle and performance of optimization is improved. Compared three classic test function, the algorithm can greatly improve the search ability. And experiments prove that it performs well on image segmentation.

Keywords: Hybrid particle swarm optimization,region equilibrium,compression factor,image segmentation

1 Introduction

Image segmentation technology is very important research work in the field of computer vision and artificial intelligence. This technology not only involves the lower data processing of image information, but also involves the upper level knowledge expression, which belongs to the classical difficult problem of image information engineering. The current methods of image segmentation mainly include threshold method, edge test [2], region growing method[3], morphology watershed method[4,5], and etc. For real-time applications, image segmentation is usually used one-dimensional Otsu method[6]. Based on the maximum class square error method, Otsu proposes an algorithm to calculate the single threshold. Single threshold Otsu method only considers the gray information of the image, but does not take into account the spatial position of the pixels. In 1979, Otsu proposed a two-dimensional segmentation method, which became a hot issue in the study[7,8]. But the computation complexity increases exponentially. How to efficiently calculating the optimal threshold becomes the key of the algorithm.Swarm intelligence evolutionary algorithms such as genetic algorithm [9],

* Zhou Kang is corresponding author (phone: 0086-027-85504737; fax: 0086-027-85504742; e-mail: zhoukang_wh@163.com).

particle swarm optimization algorithm [10] are introduced to deal with them.In this paper, the regional equilibrium optimal solution and the compression factor of flight speed control are introduced into the particle velocity update formula.The search ability and convergence accuracy of the hybrid particle swarm optimization algorithm are improved by improving the particle velocity update formula.

2 Hybrid PSO algorithm with region equilibrium and compression factor

2.1 Standard particle swarm optimization algorithm

The standard particle swarm algorithm (PSO) was proposed by Eberhart et al in 1995[1], and simulated the foraging behavior of bird. Birds use simple rules to determine their flight speed and direction. Similar to other evolutionary algorithms, PSO uses the concepts of "population" and "evolution" to operate according to the fitness value of an individual. Assuming a S dimension search space, there is a population containing m particles, wherein the particles are represented as a vector, the position of each particle is representative a potential solution (or called for the initial position), for a vector, we can calculate the fitness value using an objective function. And we can select the optimal solution according to the fitness value. The initial position, the local best position, the globe best position and the velocity for particle are respectively denoted as:$X_i = (X_{i1}, X_{i2}, ..., X_{is})$, $P_i = (P_{i1}, P_{i2}, ..., P_{is})$, $P_g = (P_{g1}, P_{g2}, ..., P_{gs})$, and $V_i = (V_{i1}, V_{i2}, ..., V_{is})$, For a particle, the velocity and position are updated by the formula:

$$A_t = c_1 r_1 (P_i(t) - X_i(t) + c_2 r_2 (P_g(t) - X_i(t)) \tag{1}$$

$$V_i(t+1) = V_i(t) + A_t \tag{2}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{3}$$

The nonnegative constants are random numbers independent of each other. We generally chose the maximum number of iterations as the condition of the iteration termination.

2.2 Optimal solution of regional equilibrium

For the standard particle swarm optimization (PSO), the velocity updating depends on the local optimal solution of the particle itself and the globe optimal solution of the whole population. In the initial stage of the algorithm, it is very likely that the particle swarm search is too fast to miss the globe optimal solution. Thus, the search is fall into local optimum.

In order to improve the search accuracy of algorithm, Zhang [11] proposed a scheme of the current optimal solution instead of the global optimal solution to balance the local search ability of each particle's neighbors, but it still has some limitations, especially the computation and storage of distance. In this paper, the particle is regarded as the center of a local region, and a circular neighborhood is set. The geometric center of the particle's current optimum position is used as the global optimum position in the region. In this way, local information about each particle will be shared more efficiently and accurately. Even in the extreme cases, there may be no other particles in the given neighborhood, so that the neighborhood should be adjusted. The neighborhood expansion process is as follows:

If the particles are not present after the three adjustments, the local information of the particle is not enough and no balance is needed, so the formula updated velocity use the original directly. A velocity updating formula based on local region optimal solution is proposed:

$$A_t = c_1 r_1 (P_i(t) - X_i(t) + c_2 r_2 (P_n(t) - X_i(t)) \tag{4}$$

Previous studies show that the global PSO algorithm converges fast, but it is easy to fall into local optimum. The algorithm combined with region equilibrium can get better optimal and more accurate solution, but the search speed will be slower. Therefore, we adopt a hybrid algorithm that combines global optimization and regional equilibrium optimization to update particle velocity:

$$H_i(t+1) = u_1 M_i(t+1) + u_2 N_i(t+1) \tag{5}$$

$$Au_i(t) = V_i(t) + c_1 r_1 (P_i(t) - X_i(t)) \tag{6}$$

For local equilibrium optimization:

$$M_i(t+1) = Au_i(t) + c_2 r_2 (P_n(t) - X_i(t) \tag{7}$$

For global equilibrium optimization:

$$N_i(t+1) = Au_i(t) + c_2 r_2 (P_g(t) - X_i(t) \tag{8}$$

$u_1 = t/T, u_2 = 1 - u_1$ ensure that the algorithm taking into account local optimization and global optimization, the particle velocity is not so fast that the best solution is missed and the algorithm is more accurate in the later stage.

2.3 Compression factor

In the formula of velocity update, the values of those two non negative constants determine the particle's own experience and the influence of the empirical information of other particles on the trajectory, reflecting the exchange of information between particles[12]. Smaller values of configuration parameters cause particles to iteration more locally; On the other hand, will cause the premature

convergence. In order to achieve the balance between global and local search, Shi put forward the strategy of setting contraction factor. The velocity update formula is as follows:

$$A_i(t) = c_1 r_1 (P_i(t) - X_i(t) + c_2 r_2 (P_g(t) - X_i(t) \tag{9}$$

$$V_i(t+1) = \varphi(V_i(t) + Au1_i(t)) \tag{10}$$

$$\varphi = 2/\sqrt{2 - c - \sqrt{c^2 - 4c}}, c = c_1 + c_2 \tag{11}$$

Document [13] pointed out that, compared to the optimization of weights, the setting of compression factors not only effectively constrains the particle velocity, but also enhances the local search ability of the algorithm. Some improvements have been made to the compression factor. In the case of controlling the velocity of the particles, the local and the global optimum fitness information are preserved, and the updated velocity update formula is as follows:

$$V_i(t+1) = \varphi(V_i(t)) + Au1_i(t) \tag{12}$$

2.4 Hybrid PSO algorithm with region equilibrium and compression factor

Based on the two factors of regional equilibrium and compression factor, the updating formulas are improved, and a hybrid particle swarm optimization algorithm is proposed. The new speed and position update formulas are as follows:

$$X_i(t+1) = X_i(t) + H_i(t+1) \tag{13}$$

$$H_i(t+1) = u_1 M_i(t+1) + u_2 N_i(t+1) \tag{14}$$

$$Aw_i(t) = \varphi(V_i(t)) + c_1 r_1 (P_i(t) - X_i(t)) \tag{15}$$

$$M_i(t+1) = Aw_i(t) + c_2 r_2 (P_n(t) - X_i(t) \tag{16}$$

$$N_i(t+1) = Aw_i(t) + c_2 r_2 (P_g(t) - X_i(t) \tag{17}$$

2.5 Algorithm Test

Since the algorithm in this paper is improved on the basis of traditional algorithms, it is necessary to compare with other algorithms to verify the performance of the algorithm. In order to test the actual performance of the proposed algorithm (HRC-PSO), it is compared with the inertia weight linear decreasing algorithm (DW-PSO) and the adaptive weight algorithm (AW-PSO).

We chose three representative and widely used test functions to verify the performance.These functions include Ackley, Rastrigrin and Schaffer function, the range of the corresponding variable respectively $(-32, 32), (5.12, 5.12) and (-10, 10)$. The global optimum is all 0. The parameter configuration is as follows: population size = 40; $C1 = C2 = 1.5$; target value = 0; Iteration number = 100. The results as shown in table 1.

Table 1. Demographic Prediction performance comparison by three evaluation metrics.

Function	Average run time (s)			Best average fitness		
	DW-PSO	AW-PSO	HRC-PSO	DW-PSO	AW-PSO	HRC-PSO
Ackley	0.0760	0.0786	0.2323	-0.525	-0.539	-0.255
Rastrigrin	0.0650	0.0586	0.1883	-0.425	-0.434	-0.221
Schaffer	0.0692	0.0667	0.2011	-0.423	-0.512	-0.105

The optimal values can be obtained by the 3 algorithms within 100 generations. On the average time index, HRC-PSO uses more complicated speed update formula, so it takes more time. But on the Best average fitness index, the superiority of the algorithm is fully reflected. With the same number of iterations, the algorithm used in this paper has long search time. But the search efficiency is much higher. This fully demonstrates the effectiveness of the strategy and the advantages of the improved algorithm.

3 Simulation experiment

3.1 Principle of Otsu threshold images segmentation

For a digital image with size$M * N$, for each pixel, its pixel gray value and neighborhood gray value are L, define a binary array (i, j), of which i is the pixel gray value, j is the average gray value of corresponding $(3*3)$ neighborhood; n_{ij} is the number of the pixel which have the same binary array (i, j); The relative probability is $P_{ij} = n_{ij}/M * N$. s is the threshold of pixel gray value and t is the gray value threshold of neighborhood.

The relative probability is:

$$p_{i,j} = \frac{n_{i,j}}{M \cdot N}; \tag{18}$$

Assume that the object and background as I_0 and I_1, s is the threshold of pixel gray value and t is the gray value threshold of neighborhood.

Two classes of the probability are:

$$P_0(s,t) = \sum_{i=1}^{s} \sum_{j=1}^{t} P_{i,j} \tag{19}$$

$$P_1(s,t) = \sum_{i=s+1}^{L} \sum_{j=t+1}^{L} P_{i,j} \tag{20}$$

Two classes of the corresponding mean vector are:

$$\mu_0 = (\sum_{i=1}^{s} \sum_{j=1}^{t} \frac{i \cdot P_{i,j}}{P_0} \quad \sum_{i=1}^{s} \sum_{j=1}^{t} \frac{j \cdot P_{i,j}}{P_0})^T \tag{21}$$

$$\mu_1 = (\sum_{i=1}^{L} \sum_{j=1}^{L} \frac{i \cdot P_{i,j}}{P_1} \quad \sum_{i=1}^{L} \sum_{j=1}^{L} \frac{j \cdot P_{i,j}}{P_1})^T \tag{22}$$

The population mean vector is

$$mu_T = (\sum_{i=1}^{L} \sum_{j=1}^{L} i \cdot P_{i,j} \quad \sum_{i=1}^{L} \sum_{j=1}^{L} j \cdot P_{i,j})^T \tag{23}$$

Ignoring the influence of noise and other, we can do the assumptions as the following:

$$P_0 + P_1 \approx 1, \mu \approx P_0\mu_0 + P_1\mu_1 \tag{24}$$

Use the trace of matrix Sb(s,t) as the discrete degree measure, So the best optional threshold value (s, t) should meet the following equation:

$$(s^*, t^*) = arg \max_{1 \le s,t \le L} (R) \tag{25}$$

How to ascertain the optimal threshold is the key for the algorithm.

3.2 Experimental results and discussions

In order to verify the performance of the algorithm proposed in this paper, the algorithm is compared with the experimental results.

The simulation experiment based on image data set, a large number of simulation is performed on the data set by these 3 algorithms. The following are discussed from two aspects

1. Does not limit the running time and compares the average time between the two algorithms to achieve the optimal threshold. Through the program implementation, we can find that the two algorithms can find the optimal threshold. However, there is a clear gap that the time when the optimal threshold is obtained. Statistically speaking, the efficiency of the improved algorithm is obviously higher than that of the traditional algorithm. This shows that the compression factor and region equalization strategy indeed improve the convergence rate of the algorithm. the experimental results are good, because of the limited space, the segmentation results of 3 original images in this paper are only shown in Table 2.

(a) Camera

(b) Sheep

(c) Man

Fig. 1. Experimental images

Table 2. Performance comparison

Image	AW-PSO		HRC-PSO	
	The threshold	Time	The threshold	Time
(a)	(90,147)	0.423	(90,147)	0.071
(b)	(66,140)	0.386	(66,140)	0.062
(c)	(82,138)	0.365	(82,138)	0.058

According to the implementation principle and running time of the algorithm, the calculation time of the algorithm is related to the size of the image and the resolution of the gray scale, but it is independent of the specific image. It has strong robustness. The algorithm has certain practical value.

2. Taking into account the online requirements of image processing, we limit the time. The segmentation results of two algorithms are compared.

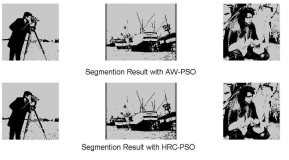


Fig. 2. Segmentation results for images

Obviously, the algorithm can accurately segment the main target in the image, and the segmentation results are clearer than the contrast algorithm, contain more details, and are closer to the original image visually. In the segmentation result of Fig. 2 (b), the image processed by the algorithm contains more details,

while other algorithms lack the corresponding details. Fig.2 (c) shows the algorithm can eliminate some noise interference, and the segmentation stability is better.Therefore, from the practical point of view, the improved algorithm has a strong ability to process images online and is more suitable for the strict requirements of time.

4 Conclusion

Based on region equilibrium and compression factor, the PSO algorithm is improved, and a threshold image segmentation algorithm based on OTSU method is proposed. In view of the traditional PSO algorithm is easy to fall into local optimal solution, we update the particle velocity based on the combination of global optimization, region equilibrium and compression factor. It is obvious that the optimal value of convergence is not very different, but the algorithm presented in this paper has obvious advantages in convergence speed. The segmentation results in image data sets show that the proposed algorithm can get the segmentation result is stable and clear, and has strong ability to overcome the curse of dimensionality.

Acknowledgments. Zhou Kang is corresponding author. The work was supported by National Natural Science Foundation of China (Grant No. 61179032 and 61303116), the Special Scientific Research Fund of Food Public Welfare Profession of China(Grant No. 2015130043), the Research and Practice Project of Graduate Education Teaching Reform of Polytechnic University (YZ2015002), the Scientific research project of Wuhan Polytechnic University (2016Y01), the science and technology research project of the Hubei province (B201601).

References

1. J. Kennedy, and R Eberhart, Particle Swarm Optimization, IEEE Conference on Neural Networks, pp. 1942-1948, (Perth, Australia), Piscataway, NJ, IV, 1995.
2. Huang F Q, Chen C X, et al. Application of PSO Algorithm to Brain Tumor Image Division [J]. Journal of Henan University of Science and Technology, 2007.
3. Ru A, Peng G, Wang H L, et al. A modified PSO algorithm for remote sensing image template matching. [J]. Photogrammetric Engineering & Remote Sensing, 2010, 76(4):379-389.
4. Chen P, Zou T, Chen J Y, et al. The Application of Improved PSO Algorithm in PMMW Image OSTU Threshold Segmentation [J]. Applied Mechanics & Materials, 2015, 721:779-782.
5. Xie X, Wang J B, Feng-Ping H U. An improved floc segmentation algorithm based on PSO and OSTU[J]. Transducer & Microsystem Technologies, 2015.
6. PÍęrez-JimÍęnez and Peng H. A novel image thresholding method based on membrane computing and fuzzy entropy. Journal of Intelligent and Fuzzy Systems, 24(2), 2013: 229-237.
7. Peng H,Wang J.Optimal multi-level thresholding with membrane computing. Digital Signal Processing, 2015, 37: 53-64.

8. Raja N S M, Sukanya S A, Nikita Y. Improved PSO Based Multi-level Thresholding for Cancer Infected Breast Thermal Images Using Otsu [J]. Procedia Computer Science, 2015, 48:524-529
9. Chhikara R R, Sharma P, Singh L. A hybrid feature selection approach based on improved PSO and filter approaches for image steganalysis[J]. International Journal of Machine Learning & Cybernetics, 2015:1-12.
10. Priya S S, Menon P R. Improved PSO algorithm approach in Gray scale image multi-level thresholding[J]. International Journal of Advanced Trends in Computer Science & Engineering, 2016.
11. Zhang P, Zhong W B.Hybrid particle swarm optimization with improved learning factor and constraint factor [J]. Computer Applied Research, 2015, 32:3626-3628
12. Liu S, Zhou K, Zeng S, et al. An Image Threshold Segmentation Algorithm with Hybrid Evolutionary Mechanisms Based on Membrane Computing[M]// Bio-inspired Computing ÍC Theories and Applications. Springer Singapore, 2016:85-94.
13. Wen P, Zhou D, Wu M, et al. Hybrid methods of particle swarm optimization and spatial credibilistic clustering with a clustering factor for image segmentation[C]// IEEE International Conference on Industrial Engineering and Engineering Management. IEEE, 2016:1443-1447.
14. Hamdaoui F, Ladgham A, Sakly A, et al. Multi-level fractional order PSO new paradigm algorithm for image segmentation[J]. 2016, 9(4/5):218.
15. Alva A, Akash R S, Manikantan K. Optimal multilevel thresholding based on Tsallis entropy and half-life constant PSO for improved image segmentation[C]// Electrical Computer and Electronics. IEEE, 2016.
16. Khairuzzaman A K M, Chaudhury S. Multilevel thresholding using grey wolf optimizer for image segmentation[J]. Expert Systems with Applications, 2017.
17. Zhu H, Zhuang Z, Zhou J, et al. Segmentation of liver cyst in ultrasound image based on adaptive threshold algorithm and particle swarm optimization[J]. Multimedia Tools & Applications, 2017, 76(6):1-18.

Matrix Representation and Simulation Algorithm of Spiking Neural P Systems with Structural Plasticity

Zechariah B. Jimenez[1], Francis George C. Cabarle[1,2], Ren Tristan A. de la Cruz[1], Kelvin C. Buño[1], Henry N. Adorna[1], Nestine Hope S. Hernandez[1], and Xiangxiang Zeng[2,3]

Dept. of Computer Science, University of the Philippines
Diliman, 1101, Quezon City, Philippines;
[2]School of Information Science and Technology, Xiamen University
Xiamen 361005, Fujian, China.
[3]Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid,
Campus de Montegancedo, Boadilla del Monte 28660, Madrid, Spain.
fccabarle@up.edu.ph

Abstract. In this paper we create a matrix representation for Spiking Neural P Systems with Structural Plasticity (SNPSP, for short), taking inspiration from existing algorithms and representations for related models. Using our matrix representation, we provide a simulation algorithm for SNPSP systems. We analyse the time and space complexity of our algorithm. Such a simulation algorithm will be useful for implementing SNPSP systems, including related models with a dynamic topology, in software or hardware.

Keywords: Spiking Neural P Systems, Structural Plasticity, Matrix Representation, Membrane Computing

1 Introduction

In the realm of computer science, models are used to describe the workings of various systems and how they are said to be "computing." These models possess various features that depict how values are changed and manipulated to achieve a desired output. Present-day computers are based off of some specific variants of the Turing machine and the like and therefore carry their characteristic advantages (and disadvantages) over other, more "primitive" models; speed, Turing-completeness, and space capacity are just some of the important properties that describe the abilities of these models. Small changes in any of these could spell the difference between being able to compute or solve a problem, and otherwise. It is thus imperative to come up with models that overcome the obstacles that impede other models, and thus solve problems better and faster. While many such models still have no commercially available physical realization, simulation on modern computers will be enough to prove the abilities of these models until such feasible prototypes are created.

A good example of some powerful models that are still in the simulation stage are those discovered in the field of natural computing, specifically in membrane computing. These are based off of natural phenomena, like the transfer of chemicals within cells and throughout cell systems. The advantage of these models over standard ones (e.g. Turing machines) are their characteristic parallelism, even over small space constraints. The parallelism could then be used to solve NP-Complete and other hard problems, and possibly in a more efficient manner than the sequential TM's. This paper focuses on a computational model based on the Spiking Neural P Systems (SNP Systems) as defined in [8, 10]. The matrix representation and simulation algorithm also draws inspiration from those mentioned in [3–6, 11]. A preliminary version of the matrix representation in this paper is in [7].

As elaborated in [1,5,11], the benefits of a matrix representation compared to other representations is due to the increased parallelism when performing linear algebra operations. This increased parallelism when simulating computations can benefit sequential (e.g. CPU) simulators but more so using parallel (e.g. GPU) simulators. More benefits using a matrix representation and other parallel computing techniques are recently given in [4, 6, 9]. In [9], variants of SN P systems that have a dynamic topology, i.e. adding or removing neurons, synapses, or both, are compared with respect to the recent technologies of GPUs. It is then noted in [9] that for such GPUs, the more efficient way to perform dynamism in the topology is the plasticity found in SNPSP systems. Also in [9] it is noted that simulator performance in GPUs can be further improved if a "compact" representation is used, e.g. removing all or most zeroes from *sparse* matrices representing the system. A zero in the matrix representation can refer to absence of a parameter, such as having no edge between two nodes.

This paper is structured as follows: in Section 2 the preliminaries for this work are introduced; in Section 3 and Section 4 our matrix representation and notations are provided, respectively; the representation and notations are used in our simulation algorithm in Section 5; An example of a "simulation run" of our algorithms is in Section 6; lastly, Section 7 provides closing remarks and research directions. Detailed proofs of our theorems are given in Appendix A.

2 Preliminaries

For this work, a specific variant of the SNP system would be in focus, namely the Spiking Neural P System with Structural Plasticity (SNPSP). Here, forgetting rules are replaced by plasticity rules, thus marking the characteristic difference between the two models. Plasticity rules allow for the creation, deletion, and rewiring of synapses by their respective source neurons. More formally, as also given in [3]:

Definition 1 (SNPSP System)**.** *A spiking neural P system with structural plasticity (SNPSP system, for short) of degree $m \geq 1$ is a construct of the form*

$$\Pi = (O, \sigma_1, \cdots, \sigma_m, syn, out)$$

where:

1. $O = \{a\}$ *is the singleton alphabet (a is called spike)*
2. $\sigma_1, \cdots, \sigma_m$ *are pairs* $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, *called neurons, where* $n_i \geq 0$ *and* $n_i \in \mathbb{N} \cup \{0\}$ *represents the initial spikes in* σ_i, R_i *is a finite set of rules of* σ_i *with the following forms:*
   (a) *Spiking rule:* $E/a^c \to a$, *where* E *is a regular expression over* O, *with* $c \geq 1$;
   (b) *Plasticity rule:* $E/a^c \to \alpha k(i, N_j)$, *where* $c \geq 1$, $\alpha \in \{+, -, \pm, \mp\}$, $k \geq 1$, $1 \leq j \leq |R_i|$, *and* $N_j \subseteq \{1, \cdots, m\}$.
3. $syn \subseteq \{1, \cdots, m\} \times \{1, \cdots, m\}$, *with* $(i, i) \notin syn$ *for* $1 \leq i \leq m$, *are synapses between neurons;*
4. $out \in \{1, \cdots, m\}$ *indicates the output neuron.*

Given neuron σ_i (we can also say neuron i or simply σ_i if there is no confusion), we denote the set of neuron labels which has σ_i as their presynaptic neuron as $pres(i)$, i.e., $pres(i) = \{j | (i, j) \in syn\}$. Similarly, we denote the set of neuron labels which has σ_i as their postsynaptic neuron as $pos(i) = \{j | (j, i) \in syn\}$.

Plasticity rules are applied as follows. If at time t we have that σ_i has $b \geq c$ spikes and $a^b \in L(E)$, a rule $E/a^c \to \alpha k(i, N) \in R_i$ can be applied. The set N is a collection of neurons to which σ_i can connect to (synapse creation) or disconnect from (synapse deletion) using the applied plasticity rule. The rule consumes c spikes and performs one of the following, depending on α:

If $\alpha = +$ and $N - pres(i) = \emptyset$, or if $\alpha = -$ and $pres(i) = \emptyset$, then there is nothing more to do, i.e., c spikes are consumed but no synapse is created or removed. For $\alpha = +$: If $|N - pres(i)| \leq k$, deterministically create a synapse to every σ_l, $l \in N_j - pres(i)$. If, however, $|N - pres(i)| > k$, then nondeterministically select k neurons in $N - pres(i)$ and create one synapse to each selected neuron.

For $\alpha = -$: If $|pres(i)| \leq k$, deterministically delete all synapses in $pres(i)$. If, however, $|pres(i)| > k$, then non- deterministically select k neurons in $pres(i)$ and delete each synapse to the selected neurons.

If $\alpha \in \{\pm, \mp\}$, create (respectively, delete) synapses at time t and then delete (respectively, create) synapses at time $t + 1$. Only the priority of application of synapse creation or deletion is changed, but the application is similar to $\alpha \in \{+, -\}$. The neuron is always open from time t until $t + 1$, i.e., the neuron can continue receiving spikes. However, the neuron can only apply another rule at time $t + 2$.

An important note is that for σ_i applying a rule with $\alpha \in \{+, \pm, \mp\}$, creating a synapse always involves an embedded sending of one spike when σ_i connects to a neuron. This single spike is sent at the time the synapse creation is applied. Whenever σ_i attaches to σ_j using a synapse during synapse creation, we have σ_i immediately transferring one spike to σ_j.

If two rules with regular expressions E_1 and E_2 can be applied at the same time, that is, $L(E_1) \cap L(E_2) \neq \emptyset$, then only one of them is nondeterministically chosen and applied. All neurons therefore apply at most one rule in one time step

(locally sequential), but all neurons that can apply a rule must do so (globally parallel). Note that the application of rules in neurons are synchronized, that is, a global clock is assumed.

A system state or configuration of an SNPSP system is based on (a) distribution of spikes in neurons and (b) neuron connections based on the synapse graph syn. We can represent (a) as $\langle s_1, \cdots, s_m \rangle$ where $s_i, 1 \leq i \leq m$, is the number of spikes contained in σ_i. For (b) we can derive $pres(i)$ and $pos(i)$ from syn, for a given σ_i. The initial configuration therefore is represented as $\langle n_1, \cdots, n_m \rangle$, with the possibility of a disconnected graph, i.e., $syn = \emptyset$. A computation is defined as a sequence of configuration transitions from an initial configuration. A computation halts if the system reaches a halting configuration, that is, a configuration where no rules can be applied and all neurons are open. Whether a computation is halting or not, we associate natural numbers $1 \leq t_1 < t_2 < \cdots$ corresponding to the time instances when the neuron out sends a spike out to (or when in receives a spike from) the system.

A result of a computation can be defined in several ways in SNP systems literature, but in this work we use the following as in [8]: We only consider the first two time instances t_1 and t_2 that σ_{out} spikes. Their difference, i.e., the number $t_2 - t_1$, is said to be computed by Π.

As an illustration, consider an SNPSP system Π_{ex} shown in Figure 1 from [3]. The labels of each rule in Figure 1 are used later in our matrix representation and algorithm so we ignore them for now. Neurons 2, $out = 3$, 4, and 5 contain only the rule $a \to a$, and we omit this from writing. In the initial configuration, at time $t_0 = 0$, is where only σ_1 has two spikes and σ_3 has only one spike. Neuron 1 is the only neuron with plasticity rules, where we have $syn = \{(2,4),(2,5),(4,1),(5,1)\}$.

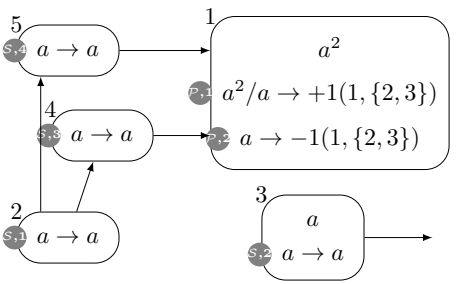


Fig. 1. An SNPSP system Π_{ex}.

As detailed in [3], we have Π_{ex} computing the set $\{1, 4, 7, 10, \ldots\} = \{3m + 1 | m \geq 0\}$. In Table 1 the output of Π_{ex} is $t_2 - t_1 = 1$ if neuron σ_1 creates synapse $(1, 3)$, where (!) means that the output neuron σ_3 fires a spike to the environment, and t_2 and t_1 are the second and first time σ_3 fires, respectively. In Table 2 the output of Π_{ex} is 4 if σ_1 creates synapse $(1, 2)$ instead of $(1, 3)$.

Time	σ_1	σ_2	σ_3		σ_{A_1}	σ_{A_2}	syn
0	2	0	1		0	0	syn
$t_1 = 1$	1	0	1	(!)	0	0	$syn \cup \{(1,3)\}$
$t_2 = 2$	0	0	0	(!)	0	0	syn

Table 1. Computation of Π_{ex} for $\{1\}$

Time	σ_1	σ_2	σ_3		σ_{A_1}	σ_{A_2}	syn
0	2	0	1		0	0	syn
$t_1 = 1$	1	1	0	(!)	0	0	$syn \cup \{(1,2)\}$
2	0	0	0		1	1	syn
3	2	0	0		0	0	syn
4	1	0	1		0	0	$syn \cup \{(1,3)\}$
$t_2 = 5$	0	0	0	(!)	0	0	syn

Table 2. Computation of Π_{ex} for $\{4\}$

3 Matrix Representation of SNPSP

To illustrate how SNPSP systems can be represented as specified below, we use Π_{ex} in Figure 1. Using the formal definition, the system can thus be expressed as $\Pi_{ex} = (\{a\}, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, syn, 3)$. The neurons are (from σ_1 to σ_5) $(2, \{r_{\mathcal{P},1}, r_{\mathcal{P},2}\})$, $(0, \{r_{\mathcal{S},1}\})$, $(1, \{r_{\mathcal{S},2}\})$, $(0, \{r_{\mathcal{S},3}\})$, and $(0, \{r_{\mathcal{S},4}\})$; the synapses are defined as $syn = \{(2,4), (2,5), (5,1), (4,1)\}, 3)$; and finally the rules are $r_{\mathcal{P},1} = a^2/a \to +1(1, \{2,3\})$, $r_{\mathcal{P},2} = a \to -1(1, \{2,3\})$, and $r_{\mathcal{S},i} = a \to a, \forall i \in \{1,2,3,4\}$.

For SNPSP systems, a neuron is said to be defined by its spike count and the set of rules associated with it. With this, we define the spike count vector and the rule source matrix.

Definition 2 (Spike Count Vector)**.** *Let Π be an SNPSP system with m neurons. In a computation, for any $k \in \mathbb{N}$, the vector $C^{(k)} = [c_1^{(k)}, c_2^{(k)}, \cdots, c_m^{(k)}]$ is called the spike count vector of the system at time k, where $c_i^{(k)}$ is the amount of spikes in neuron $\sigma_i, i = 1, 2, \cdots, m$ at time k.*

Definition 3 (Rule Source Matrix)**.** *Let Π be an SNPSP system with m neurons. Let $r_{\mathcal{R}}$ be the number of rules of type $\mathcal{R} \in \{\mathcal{P}, \mathcal{S}\}$, where $\mathcal{P}$ and $\mathcal{S}$ correspond to plasticity and spiking rules, respectively. Let $d_{\mathcal{R}} : (\mathcal{R}, 1), \cdots, (\mathcal{R}, r_{\mathcal{R}})$ be a total ordering of rules of type $\mathcal{R}$. The rule source matrices of the system Π, $Sr_{\mathcal{R}}$, are defined as follows:*

$$Sr_{\mathcal{R}} = \begin{bmatrix} sr_{\mathcal{R},1,1} & \cdots & sr_{\mathcal{R},1,m} \\ \vdots & \ddots & \vdots \\ sr_{\mathcal{R},r_{\mathcal{R}},1} & \cdots & sr_{\mathcal{R},r_{\mathcal{R}},m} \end{bmatrix}$$

where:

$$sr_{\mathcal{R},i,j} = \begin{cases} 1, & \text{if rule } r_{\mathcal{R},i} \text{ is in neuron } \sigma_j; \\ 0, & \text{otherwise.} \end{cases}$$

We also define an aggregate rule source matrix Sr to denote the combination of $Sr_{\mathcal{P}}$ and $Sr_{\mathcal{S}}$, with the rows (rules) arbitrarily ordered.

Next in the definition of SNPSP systems is the set of synapses. Here, since these connections are not constant, the synapse matrix is defined to change with

time. In addition to that, we define matrices that record the newly created (and deleted) synapses.

Definition 4 (Synapse Matrix)**.** *In an SNPSP system Π with m neurons, the synapse matrix, $Sy^{(k)}$, at time k, is defined as follows:*

$$Sy^{(k)} = \left[sy_{i,j}^{(k)}\right]_{m\times m} = \begin{bmatrix} sy_{1,1}^{(k)} & \cdots & sy_{1,m}^{(k)} \\ \vdots & \ddots & \vdots \\ sy_{m,1}^{(k)} & \cdots & sy_{m,m}^{(k)} \end{bmatrix}$$

where:

$$sy_{i,j}^{(k)} = \begin{cases} 1, & \text{if there exists a synapse from neuron } \sigma_i \\ & \text{to neuron } \sigma_j \text{ at time } k; \\ 0, & \text{otherwise.} \end{cases}$$

Definition 5 (Synapse Creation [Deletion] Matrix)**.** *In an SNPSP system Π with m neurons, the synapse creation [deletion] matrix, $Sy_{+}^{(k)}$ [$Sy_{-}^{(k)}$], at time k, is defined as follows ($o \in \{+,-\}$):*

$$Sy_{o}^{(k)} = \left[sy_{o,i,j}^{(k)}\right]_{m\times m} = \begin{bmatrix} sy_{o,1,1}^{(k)} & \cdots & sy_{o,1,m}^{(k)} \\ \vdots & \ddots & \vdots \\ sy_{o,m,1}^{(k)} & \cdots & sy_{o,m,m}^{(k)} \end{bmatrix}$$

where:

$$sy_{o,i,j}^{(k)} = \begin{cases} 1, & \text{if a synapse from neuron } \sigma_i \text{ to neuron } \sigma_j \\ & \text{was operated on at time } k; \\ 0, & \text{otherwise.} \end{cases}$$

and the indicated operation is deletion if $o = -$ or creation if $o = +$.

We also define a synapse change matrix $Sy_{\Delta}^{(k)} = Sy_{+}^{(k)} - Sy_{-}^{(k)}$ to be the net change in the synapse matrix at time k.

Given Definitions 4 and 5, we can obtain the next synapse matrix with

$$Sy^{(k)} = Sy^{(k-1)} + Sy_{\Delta}^{(k)} = Sy^{(k-1)} + Sy_{+}^{(k)} - Sy_{-}^{(k)} \tag{1}$$

Definition 2 records the spikes stored in the neurons, but here we would also need to know about the spikes sent out to the environment. For this, we have:

Definition 6 (Output Spike Count and Output Spike Indicator)**.** *In an SNPSP system Π, the output spike count at time k is denoted by $os^{(k)}$, which is the number of spikes already sent out by the output neuron to the environment from time 0 to time k. The output spike indicator at time k is defined as*

$$sp^{(k)} = \begin{cases} 1, & \text{if a spike was sent out to the environment} \\ & \text{at time } k; \\ 0, & \text{otherwise.} \end{cases}$$

In the computation as in Table 2, the output neuron spiked to the environment at times $t_1 = 1$ and $t_2 = 5$. Table 3 shows the values of $os^{(k)}$ and $sp^{(k)}$.

Time	$os^{(k)}$	$sp^{(k)}$
0	0	0
$t_1 = 1$	1	1
2	1	0
3	1	0
4	1	0
$t_2 = 5$	2	1

Table 3. Output Spike Counts and Indicators for Π_{ex} Computing {4}

form	pattern	p	q
a^*	$a^0(a^1)^*$	0	1
a^+	$a^1(a^1)^*$	1	1
a^k	$a^k(a^0)^*$	k	0
$a^k(a^j)^*$	$a^k(a^j)^*$	k	j
$a^k(a^j)^+$	$a^{k+j}(a^j)^*$	$k+j$	j

Table 4. Allowed Forms of Regular Expressions

Next, we define vectors and matrices that describe the rules associated with the neurons of the system Π. First, we need to describe the regular expressions used by the rules to determine the number of spikes required for firing. For this work we shall be limiting these regular expressions to be of the forms a^k, a^+, a^*, $a^k(a^j)^*$, and $a^k(a^j)^+$, for some positive integers j and k. In general, we describe these regular expressions to be of *linear* form – i.e., they can be described by the pattern $a^{p+qn} = a^p(a^q)^*$ for integers $p, q, n \geq 0$, $p+q \geq 1$. Table 4 illustrates this.

Thus, we can describe the regular expressions by their corresponding p and q values. To wit:

Definition 7 (Regular Expression P and Q Vectors)**.** *In an SNPSP system Π, $P_{\mathcal{S}}$ and $Q_{\mathcal{S}}$ [$P_{\mathcal{P}}$ and $Q_{\mathcal{P}}$] are the (regular expression) P and Q vectors of the spiking [plasticity] rules, defined as $P_{\mathcal{R}} = [p_{\mathcal{R},1}, \cdots, p_{\mathcal{R},r_{\mathcal{R}}}]$ and $Q_{\mathcal{R}} = [q_{\mathcal{R},1}, \cdots, p_{\mathcal{R},r_{\mathcal{R}}}]$ for $\mathcal{R} \in \{\mathcal{P}, \mathcal{S}\}$, which describe the p and q values for the regular expressions of each rule, such that:*

$$E_{\mathcal{R},i} = a^{\bar{p}+\bar{q}n} = a^{\bar{p}}(a^{\bar{q}})^*$$

where $\bar{p} = p_{\mathcal{R},i}$, $\bar{q} = q_{\mathcal{R},i}$, and $E_{\mathcal{R},i}$ is the regular expression of the rule $r_{\mathcal{R},i}$.

We also define aggregate P and Q vectors to denote the combination of $P_{\mathcal{P}}$ with $P_{\mathcal{S}}$, and $Q_{\mathcal{P}}$ with $Q_{\mathcal{S}}$, respectively. The elements are arbitrarily ordered.

Once we can decide if a rule can fire, we can then check which rules would fire and which would not. Note that for this work, if a rule is applicable, it must fire immediately. Rules have also been restricted to determinism per neuron, and to sequentiality. Thus, for rules r_a and r_b both in the same neuron, $r_a \neq r_b$, $L(E_a) \cap L(E_b) = \emptyset$. We then have the following definition:

Definition 8 (Rule Firing Vector)**.** *In an SNPSP system Π, the rule firing vectors at time k are defined as the vector*

$$Fi_{\mathcal{R}}^{(k)} = \left[fi_{\mathcal{R},1}^{(k)}, \cdots, fi_{\mathcal{R},r_{\mathcal{R}}}^{(k)} \right]$$

for $\mathcal{R} \in \{\mathcal{P}, \mathcal{S}\}$ ($\mathcal{P}$ for plasticity rules, $\mathcal{S}$ for spiking rules). The vectors describe which rules will be fired, as follows:

$$fi^{(k)}_{\mathcal{R},i} = \begin{cases} 1, & \text{if rule } r_{\mathcal{R},i} \text{ is fired at time } k; \\ 0, & \text{otherwise.} \end{cases}$$

We also define an aggregate rule firing vector Fi to denote the combination of $Fi_{\mathcal{P}}$ and $Fi_{\mathcal{S}}$, with the elements arbitrarily ordered.

Once a rule is fired, it consumes a specified number of spikes from its source neuron. Thus, we have:

Definition 9 (Spike Consumption Vector)**.** *In an SNPSP system Π, the spike consumption vectors at time k are defined as the vector $Co^{(k)}_{\mathcal{R}} = \left[co^{(k)}_{\mathcal{R},1}, \cdots, co^{(k)}_{\mathcal{R},r_{\mathcal{R}}}\right]$, for $\mathcal{R} \in \{\mathcal{P}, \mathcal{S}\}$ ($\mathcal{P}$ for plasticity rules, $\mathcal{S}$ for spiking rules). Here, $co^{(k)}_{\mathcal{R},i} = c$ is the number of spikes consumed by rule $r_{\mathcal{S},i} = E/a^c \to a^p$ if $\mathcal{R} = \mathcal{S}$, or by rule $r_{\mathcal{P},i} = E/a^c \to \alpha k(i, N)$ if $\mathcal{R} = \mathcal{P}$.*

We also define an aggregate spike consumption vector Co to denote the combination of $Co_{\mathcal{P}}$ and $Co_{\mathcal{S}}$, with the elements arbitrarily ordered.

The plasticity rules have four types of operations, namely $+$ for synapse creation, $-$ for synapse deletion, $\pm$ for successive creation and deletion in two time steps, and $\mp$ for successive deletion and creation. Just as in [7], this is further illustrated as timers of the form $(creation, deletion)$ follows: starting at an idle state, the timer is initialized at $(0, 0)$. A $+$ or a $-$ operation will set it to $(1, 0)$ and $(0, 1)$, respectively. Lastly, the $\pm$ and $\mp$ operations, having their component operations done in two consecutive time steps, set the timer to $(1, 2)$ and $(2, 1)$, respectively. For all of these, the timers count down at every time step up to 0.

Definition 10 (Timer Matrix)**.** *In an SNPSP system Π, the timer matrix at time k is defined as the matrix*

$$Ti^{(k)} = \begin{bmatrix} ti^{(k)}_{1,1} & ti^{(k)}_{1,2} \\ \vdots & \vdots \\ ti^{(k)}_{r_{\mathcal{P}},1} & ti^{(k)}_{r_{\mathcal{P}},2} \end{bmatrix}$$

where, for $o = [+, -]$:

$$ti^{(k)}_{i,j} = \begin{cases} t, & \text{if rule } r_{\mathcal{P},i} \text{ is to execute } o_j \text{ at time } k + t - 1; \\ 0, & \text{otherwise.} \end{cases}$$

We also define a primed timer matrix, $Ti'^{(k)}$, which is the timer after ticking (counting down) at time k, but before rules are fired at time k. Thus $Ti^{(k)}$ is also called the unprimed timer matrix.

Once we find out a rule should fire at time k, we then start the timer using the following matrix:

Definition 11 (Timer Start Matrix). *In an SNPSP system Π, the timer start matrix is defined as the matrix*

$$St = \begin{bmatrix} st_{1,1} & st_{1,2} \\ st_{2,1} & st_{2,2} \\ \vdots & \vdots \\ st_{r_{\mathcal{P}},1} & st_{r_{\mathcal{P}},2} \end{bmatrix}$$

where each $st_{i,j}$ would be the value that $ti_{i,j}^{(k)}$ should be set to once rule $r_{\mathcal{P},i}$ is to fire at time k.

For the remaining parts of the plasticity rules as defined, we have the following:

Definition 12 (Destination Candidate Matrix). *In an SNPSP system Π, the destination candidate matrix is defined as the matrix*

$$NM = \begin{bmatrix} nm_{1,1} & \cdots & nm_{1,m} \\ \vdots & \ddots & \vdots \\ nm_{r_{\mathcal{P}},1} & \cdots & nm_{r_{\mathcal{P}},m} \end{bmatrix}$$

where

$$nm_{i,j} = \begin{cases} 1, & \text{if } i \in N_j, \text{ for rule } r_{\mathcal{P},j} = E/a^c \rightarrow \alpha k(i, N_j); \\ 0, & \text{otherwise.} \end{cases}$$

Definition 13 (Synapse Count Vector). *In an SNPSP system Π, the synapse count vector is defined as the vector*

$$KV = [kv_1, \cdots, kv_{r_{\mathcal{P}}}]$$

where each $kv_i = k$, for rule $r_i = E/a^c \rightarrow \alpha k(i, N_j)$.

Finally, for the simulations, we need to keep track of each computation step and system configuration. Thus, we have the following definitions.

Definition 14 (Spike Gain Vector). *In an SNPSP system Π with m neurons, the spike gain vector at time k is defined as the vector $G^{(k)} = \left[g_1^{(k)}, \cdots, g_m^{(k)}\right]$ where each $g_i^{(k)}$ is the number of spikes gained by the neuron σ_i in time k, from other neurons. These gains can also be segregated according to the type of the rule that caused that gain, as with $G_{\mathcal{R}}^{(k)} = \left[g_{\mathcal{R},1}^{(k)}, \cdots, g_{\mathcal{R},m}^{(k)}\right]$ where $\mathcal{R} = \mathcal{P}$ for plasticity rules and $\mathcal{R} = \mathcal{S}$ for spiking rules.*

Definition 15 (Spike Loss Vector). *In an SNPSP system Π with m neurons, the spike loss vector at time k is defined as the vector $L^{(k)} = \left[l_1^{(k)}, \cdots, l_m^{(k)}\right]$ where each $l_i^{(k)}$ is the number of spikes lost by the neuron σ_i in time k from spike consumption by rule firing. These losses can also be segregated according to the type of the rule that caused that loss, as with $L_{\mathcal{R}}^{(k)} = \left[l_{\mathcal{R},1}^{(k)}, \cdots, l_{\mathcal{R},m}^{(k)}\right]$ where $\mathcal{R} = \mathcal{P}$ for plasticity rules and $\mathcal{R} = \mathcal{S}$ for spiking rules.*

Definition 16 (System State). *In the computations of an SNPSP system Π, the overall system state at time k is defined as*

$$\begin{aligned} Cf^{(k)} &= \langle Rule^{(k)} | Syn^{(k)} | Conf^{(k)} \rangle \\ &= \langle Fi^{(k)}, Ti^{(k)}, os^{(k)}, sp^{(k)} | Sy_{\Delta}^{(k)} | C^{(k)}, Sy^{(k)}, Ti'^{(k)} \rangle \end{aligned}$$

where $Rule^{(k)}$ is the rule change node, $Syn^{(k)}$ is the synapse change node, and $Conf^{(k)}$ is the system configuration node, all for time k.

The initial state $Cf^{(0)}$ marks the start of a computation. A computation is only to be terminated by a halting state $Cf^{(t)}$, where either (1) $os^{(t)}$ has been set to 2, or (2) t has reached a certain desired maximum time step.

The next few definitions would be for representing and generating computations and would be very important in the simulation algorithms.

Definition 17 (Computation Trace). *Given an SNPSP system Π, a computation trace of Π is a sequence of nodes $\{Conf^{(0)}, Rule^{(1)}, Syn^{(1)}, Conf^{(1)}, \ldots, Conf^{(t)}\}$ starting with an initial configuration node $Conf^{(0)}$ followed by triples of nodes of $(Rule^{(k)}, Syn^{(k)}, Conf^{(k)})$ representing system states. A computation trace is said to be valid iff the following conditions are satisfied:*

- *each system state $Cf^{(k)}$ (after the initial configuration) can be correctly generated or computed from the previous system state $Cf^{(k-1)}$;*
- *the initial system state is represented by $Rule^{(0)}$ (not in the sequence but defined to be filled with 0-values), $Syn^{(0)}$ (also not in the sequence but defined to be filled with 0-values), and $Conf^{(0)}$;*
- *the terminating (halting) system step is represented by the last rule change node $Rule^{(t)}$ either holds $os^{(t)} = 2$ or t has reached a maximum time step.*

Definition 18 (Computation Tree). *Given an SNPSP system Π, a computation tree/graph for Π is a rooted graph where each path from the root (the initial configuration node $Conf^{(0)}$) to a leaf (halting configuration node $Conf^{(t)}$) is a computation trace for Π. A computation tree is said to be correct if the set of all paths from the root to the leaves is equal to the set of valid computation traces.*

Note that we would allow loops in generating a computation tree, thus making it more appropriate to call them computation *graphs*.

4 Notations and Conventions

Here we would describe the conventions and notations in writing matrices. In this work, given a matrix Mat, we would refer to the rth row and the cth column as Mat_r and $Mat_{(c)}$, respectively. Note that these are both row vectors. For a matrix with subscripts and superscripts, as with $Mat_x^{(k)}$, we would then have $Mat_{x,r}^{(k)}$ and $Mat_{x,(c)}^{(k)}$. Since scalars here would usually be written in lowercase, a particular element of the matrix (say, the (i,j)th) would be denoted by $mat_{i,j}$. For example, for a matrix

$$Mat = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

$Mat_1 = [1,2,3]$ is the first row, $Mat_{(2)} = [2,2,2]$ is the second column (as a row vector), and $mat_{2,3} = 3$ is the value at the intersection of the second row and third column.

5 Simulation Algorithm

The algorithm is centered on forming the computation tree from a given configuration, by first branching out into *rule nodes* for the rule changes at the current time. Then, the rule nodes propagate into *synapse nodes* for the synapse changes (now for the next time step). Lastly, the synapse nodes branch out into their own *configuration nodes* for the system configurations. Note that since we are only considering nondeterminism in the synapse level, the configuration nodes will only ever branch out into just one single rule node each.

The simulation main algorithm will go as in Algorithm 1, creating a computation "tree" (strictly speaking, since identical configuration nodes will be joined, it is more of a computation graph) up to a specified depth. It creates the graph by forming the configuration nodes in a breadth-first manner using a queue, then the subtree of each configuration node (up to two levels) if created in a depth-first manner. The history of the past configurations (for checking uniqueness) is created using some arbitrary data structure. The specific methods of the graph and queue (connect, dequeue, enqueue, pop, push, empty, tooDeep) would not be specified in detail. Details of the proofs of the following Theorems are given in Appendix A.

Algorithm 2 will then check the applicability of the rules. This is done using the P and Q vectors of the given system. The `for` loop in Line 3 would check if a^{n_i}, where n_i is the number of spikes in neuron σ_i, would satisfy the regular expression for each of the rules. Note that the output spikes are monitored by Line 9. `newFi` (in Line 5 returns an all-zero firing vector for rules of type $\mathcal{R}$. Line 13 would just check if the rule was already fired and is still applying a plasticity operation, where the timer would be at 0, since it didn't just start firing then. The primed timer would just be copied over to the unprimed timer without changes. Otherwise, if the rule would only start to be applied, then Line

Algorithm 1: Main Algorithm

```
1  initializeValues()
   /* gets input, initializes matrices & vectors, and generally
      initializes system                                          */
2  confs ← [Cf^(0)]
3  hist ← [Cf^(0) → node (Cf^(0))]
   /* mapping of all prev configurations to their nodes in the
      computation tree                                            */
4  while not empty(confs) :
5      conf ← dequeue(confs)
6      if not tooDeep(conf) :
7          continue
8      rules ← getRules(conf)
9      while not empty(rules) :
10         rule ← pop(rules)
11         connect(conf, rule)
12         syns ← getSyns(conf, rule)
13         while not empty(syns) :
14             syn ← pop(syns)
15             connect(rule, syn)
16             cur    getConf(conf, rule, syn)
17             if cur in hist :
18                 connect(syn, hist [ cur ] )
19             else:
20                 connect(syn,cur)
21                 if os < 2 : enqueue(confs,conf)
```

16 would start the timer. Lastly, Line 18 would return the appropriate rule node. Details of `newRule`() would not be given, except for it being the constructor of rule nodes.

Theorem 1. *For an SNPSP system Π, the* `getRules`*() function (as described in Algorithm 2) generates a list of all the applicable rule nodes $Rule^{(k)}$ given $Conf^{(k-1)}$.*

Algorithm 3 would generate each configuration based on the possible combinations of candidate neurons nondeterministically selected by plasticity rules. `getCandidates`() would generate all permutations for this given these candidates (based from NM and KV and whichever of the synapses are existent on $Sy^{(k)}$). It would return a vector of (Sy_+, Sy_-) pairs.

Theorem 2. *For an SNPSP system Π, the* `getSyns`*() function (as described in Algorithm 3) returns $Syn^{(k)}$ given $Conf^{(k-1)}$, and $Rule^{(k)}$.*

Lastly, Algorithm 4 is focused on creating the current configuration given the previous one. First, we note that the kth configuration can be calculated from

Algorithm 2: Get Rule Nodes

function *getRules(* conf *)* **:**
- $k \leftarrow k + 1$
- $os^{(k)}, sp^{(k)} \leftarrow os^{(k-1)}, 0$
- **for each** $\mathcal{R}$ **in** $\{\mathcal{S}, \mathcal{P}\}$ **:**
  - $Sp \leftarrow (C^{(k-1)} \times Sr_{\mathcal{R}}^{T}) - P_{\mathcal{R}}$
  - $Fi_{\mathcal{R}}^{(k)} \leftarrow$ `newFi`$(\mathcal{R})$
  - **for** i **from** 1 **to** $r_{\mathcal{R}}$ **:**
    - **if** $(q_{\mathcal{R},i}, Sp_i > 0$ **and** Sp_i **mod** $q_{\mathcal{R},i} = 0)$ **or** $q_{\mathcal{R},i}, Sp_i = 0$ **:**
      - $fi_{\mathcal{R},i}^{(k)} \leftarrow 1$
      - **if** $\mathcal{R} = \mathcal{S}$ **and** $Sr_{\mathcal{S},out,i} = 1$ **:** $os^{(k)}, sp^{(k)} \leftarrow os^{(k)} + 1, 1$
    - **else :**
      - $fi_{\mathcal{R},i}^{(k)} \leftarrow 0$
- **for** i **from** 1 **to** $r_{\mathcal{P}}$ **:**
  - **if** 1 **in** $Ti_i^{\prime(k-1)}$ **:**
    - $fi_{\mathcal{P},i}^{(k)} \leftarrow 0$
    - $Ti_i^{(k)} \leftarrow Ti_i^{\prime(k-1)}$
  - **else if** $fi_{\mathcal{P},i}^{(k)} = 1$ **:**
    - $Ti_i^{(k)} \leftarrow St_i$
- **return** [`newRule`$(Fi^{(k)}, Ti^{(k)}, os^{(k)}, sp^{(k)})$]

Algorithm 3: Get Synapse Nodes

function *getSyns(* conf, rule *)* **:**
- syns $\leftarrow$ [ ]
- **for each** (Sy_+, Sy_-) **in** `getCandidates`$(Sr_{\mathcal{P}}, NM, KV, Sy^{(k)})$ **:**
  - `/* gets a list of all possible combinations of candidate synapses based on the N of the rule and the previous synapse connections */`
  - $Sy_{\Delta}^{(k)} \leftarrow Sy_{+}^{(k)} - Sy_{-}^{(k)}$
  - `push`(syns, `newSyn`$(Sy_{\Delta}^{(k)})$)
- **return** syns

the total gain and the total loss as such:

$$C^{(k)} = C^{(k-1)} + G^{(k)} - L^{(k)}$$

We are classifying the gains/losses according to the type of the causing rule, and thus $L^{(k)} = L_{\mathcal{P}}^{(k)} + L_{\mathcal{S}}^{(k)}$ and $G^{(k)} = G_{\mathcal{P}}^{(k)} + G_{\mathcal{S}}^{(k)}$.

Spike gains from spiking rules can be computed by checking the rules that fired, then tracing the source neurons of those rules and the destination of their corresponding out-synapses, therefore:

Theorem 3. *For an SNPSP system Π with m neurons and $r_{\mathcal{S}}$ spiking rules, where $d : 1, \cdots, r_{\mathcal{S}}$ is a total order for the spiking rules, the total spike gain from spiking rules at time k can be computed using*

$$G_{\mathcal{S}}^{(k)} = Fi_{\mathcal{S}}^{(k)} \times Sr_{\mathcal{S}} \times Sy^{(k)}$$

On the other hand, plasticity rules can only cause spike gains during synapse creation. Thus, gains from plasticity rules can be computed by checking the destination of the newly-created synapses, if any. In symbol form:

Theorem 4. *For an SNPSP system Π with m neurons and $r_{\mathcal{P}}$ plasticity rules, where $d : 1, \cdots, r_{\mathcal{P}}$ is a total order for the plasticity rules, the total spike gain from plasticity rules at time k can be computed by summing all of the rows of $Sy_{+}^{(k)}$ using $G_{\mathcal{P}}^{(k)} = \sum_{i=1}^{r_{\mathcal{P}}} Sy_{+,i}^{(k)}$.*

Spikes are only lost on consumption during rule firing. So for both rule types, this is computed from checking how many spikes are consumed according to the rules and then checking the source neurons of these rules.

Theorem 5. *For an SNPSP system Π with m neurons, $r_{\mathcal{S}}$ spiking rules, $r_{\mathcal{P}}$ plasticity rules, where $d_{\mathcal{R}} : 1, \cdots, r_{\mathcal{R}}$ is a total order for the spiking [plasticity] rules and $\mathcal{R} = \mathcal{S}$ [$\mathcal{R} = \mathcal{P}$], the total spike loss from spiking [plasticity] rules at time k can be computed using*

$$L_{\mathcal{R}}^{(k)} = (Fi_{\mathcal{R}}^{(k)} \odot Co_{\mathcal{R}}) \times Sr_{\mathcal{R}}$$

where $\mathcal{R} \in \{\mathcal{S}, \mathcal{P}\}$, and $\odot$ is element-wise multiplication

Theorem 6. *For an SNPSP system Π, the `getConf()` function (as described in Algorithm 4) returns $Conf^{(k)}$ given $Conf^{(k-1)}$, $Rule^{(k)}$, and $Syn^{(k)}$.*

Theorem 7. *For an SNPSP system Π (that follows the restrictions assumed in this paper), Algorithm 1 can correctly simulate the computation of Π and generate a correct computation tree (graph).*

Further algorithm analysis and proof of correctness are detailed in [7], as summarized in Table 5. In the next section, we give an example of a "run" of our algorithms in this section to simulate a simple RSSN P system.

6 Example Simulation

In this section we demonstrate the matrix representation and algorithms from the previous section using Π_{ex} from Figure 1. Note that for the illustrations to follow, the matrices and vectors that define an aggregate version that combines those

Algorithm 4: Get Configuration Nodes

function *getConf(* conf, rule, syn *)* **:**

$Sy^{(k)} \leftarrow Sy^{(k-1)} + Sy_{\Delta}^{(k)}$

$G_{\mathcal{S}}^{(k)} \leftarrow Fi_{\mathcal{S}}^{(k)} \times Sr_{\mathcal{S}} \times Sy^{(k)}$

$G_{\mathcal{P}}^{(k)} \leftarrow \texttt{sumRows}(Sy_{+}^{(k)})$

$L_{\mathcal{S}}^{(k)} \leftarrow (Fi_{\mathcal{S}}^{(k)} \odot Co_{\mathcal{S}}) \times Sr_{\mathcal{S}}$

$L_{\mathcal{P}}^{(k)} \leftarrow (Fi_{\mathcal{P}}^{(k)} \odot Co_{\mathcal{P}}) \times Sr_{\mathcal{P}}$

$G^{(k)} \leftarrow G_{\mathcal{P}}^{(k)} + G_{\mathcal{S}}^{(k)}$

$L^{(k)} \leftarrow L_{\mathcal{P}}^{(k)} + L_{\mathcal{S}}^{(k)}$

$C^{(k)} \leftarrow C^{(k-1)} + G^{(k)} - L^{(k)}$

for i **from** 1 **to** $r_{\mathcal{P}}$ **do**

  for j **from** 1 **to** 2 **do**

    $ti_{i,j}^{\prime(k)} \leftarrow \texttt{max}(ti_{i,j}^{(k)} - 1, 0)$

return $\texttt{newConf}((C^{(k)}, Sy^{(k)}, Ti^{\prime(k)})$

Algorithm	Time Complexity	Space Complexity	Notes
Tree (node count)	-	$O((2e)^{m^2 t/2})$	
Algorithm 1	$O(F(F+mr))$	$O(F + F^{r/(m^2)})$	$F = (2e)^{m^2 t/2} m^2$
Algorithm 2	-	$O(mr)$	
Algorithm 3	$O((2e)^{m^2 t/2} m^2)$	-	
Algorithm 4	$O(mr)$	-	

Table 5. Space and Time Complexities of Algorithms Presented

for plasticity and for spiking rules (i.e. Sr, P, Q, Fi, Co), the *arbitrary ordering* as specified in their respective definitions would simply be the concatenation of those for the spiking rules and for the plasticity rules. In other words, as with the rule firing vector, the resulting vector would be

$$Fi^{(k)} = \left[fi_{\mathcal{S},0}^{(k)} \cdots fi_{\mathcal{S},r_{\mathcal{S}}}^{(k)} \middle| fi_{\mathcal{P},0}^{(k)} \cdots fi_{\mathcal{P},r_{\mathcal{P}}}^{(k)} \right]$$

Given the initial values as computed above, the initial configuration of the system is

$$\begin{aligned} Cf^{(0)} &= \langle Rule^{(0)} | Syn^{(0)} | Conf^{(0)} \rangle \\ &= \left\langle \left[0\,0\,0\,0 \middle| 0\,0 \right], \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, 0, 0 \middle| \begin{bmatrix} 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \end{bmatrix} \middle| \left[2\,0\,1\,0\,0 \right], \begin{bmatrix} 0\,0\,0\,0\,0 \\ 0\,0\,0\,1\,1 \\ 0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \right\rangle \end{aligned} \tag{2}$$

We have already computed for the rule firing vectors and the timer matrix at time 1, which are

$$Fi^{(1)} = \left[\begin{array}{ccccc|cc} 0 & 1 & 0 & 0 & & 1 & 0 \end{array}\right] \qquad (3)$$

$$Ti^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad (4)$$

Since we have already decided that rules $r_{\mathcal{S},2}$ and $r_{\mathcal{P},1}$ are to fire, we could proceed to selecting which synapses are to be operated on using $r_{\mathcal{P},1}$. Since in the rule source matrix, $sr_{\mathcal{P},1,1} = 1$ (for *plasticity* rule number *one*, for the *first* neuron), then the source neuron of rule $r_{\mathcal{P},1}$ is neuron σ_1. The candidate destination neurons for the same rule are neurons σ_2 and σ_3, since $nm_{1,2} = nm_{1,3} = 1$ (for plasticity rule number *one*, for the *second* and *third* neurons). The operation is $o_1 = +$, given that $ti_{1,1} = 1$ (for plasticity rule number *one*, for the *first* neuron). Thus, we are to select $kv_1 = 1$ neuron from these two candidates to which we would *create* a synapse to (since the chosen operation is $op_1 = +$ for *synapse creation*). In the example where Π_{ex} computed 4, the first selected neuron was σ_2.

Since rule $r_{\mathcal{S},2}$ is in neuron $\sigma_3 = \sigma_{out}$ and has fired, we know that it has caused a spike to be sent to the environment at time 1. Therefore, the output spike count and indicator are $os^{(1)} = 1$ and $sp^{(1)} = 1$.

Afterwards, we could now create the next configuration. We have

$$Sy_{\Delta}^{(1)} = Sy_{+}^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (5)$$

$$\begin{aligned} Sy^{(1)} &= Sy^{(0)} + Sy_{\Delta}^{(1)} = Sy^{(0)} + Sy_{+}^{(1)} - Sy_{-}^{(1)} = Sy^{(0)} + Sy_{+}^{(1)} \\ &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \qquad (6)$$

Now we can use Algorithm 4 to create the matrices for the next configuration.

$$G_{\mathcal{S}}^{(1)} = Fi_{\mathcal{S}}^{(0)} \times Sr_{\mathcal{S}} \times Sy^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(7)

$$G_{\mathcal{P}}^{(1)} = sumRows(Sy_{+}^{(1)}) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times Sy_{+}^{(1)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

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

$$L_{\mathcal{P}}^{(1)} = (Fi_{\mathcal{P}}^{(0)} \odot Co_{\mathcal{P}}) \times Sr_{\mathcal{P}} = \left(\begin{bmatrix}1 & 0\end{bmatrix} \odot \begin{bmatrix}1 & 1\end{bmatrix}\right) \times \begin{bmatrix}1 & 0 & 0 & 0 & 0\\ 1 & 0 & 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 0 & 0\end{bmatrix} \tag{10}$$

$$L^{(1)} = L_{\mathcal{S}}^{(1)} + L_{\mathcal{P}}^{(1)} = \begin{bmatrix}0 & 0 & 1 & 0 & 0\end{bmatrix} + \begin{bmatrix}1 & 0 & 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}1 & 0 & 1 & 0 & 0\end{bmatrix} \tag{11}$$

$$G^{(1)} = G_{\mathcal{S}}^{(1)} + G_{\mathcal{P}}^{(1)} = \begin{bmatrix}0 & 1 & 0 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0 & 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0 & 0\end{bmatrix} \tag{12}$$

$$C^{(1)} = C^{(0)} + G^{(1)} - L^{(1)} = \begin{bmatrix}2 & 0 & 1 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 1 & 0 & 0 & 0\end{bmatrix} - \begin{bmatrix}1 & 0 & 1 & 0 & 0\end{bmatrix} = \begin{bmatrix}1 & 1 & 0 & 0 & 0\end{bmatrix} \tag{13}$$

Finally, the timer matrix would count down, and we would have

$$Ti'^{(1)} = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} \tag{14}$$

Given the computation as illustrated so far, the current state of the computation tree is shown in Figure 2.

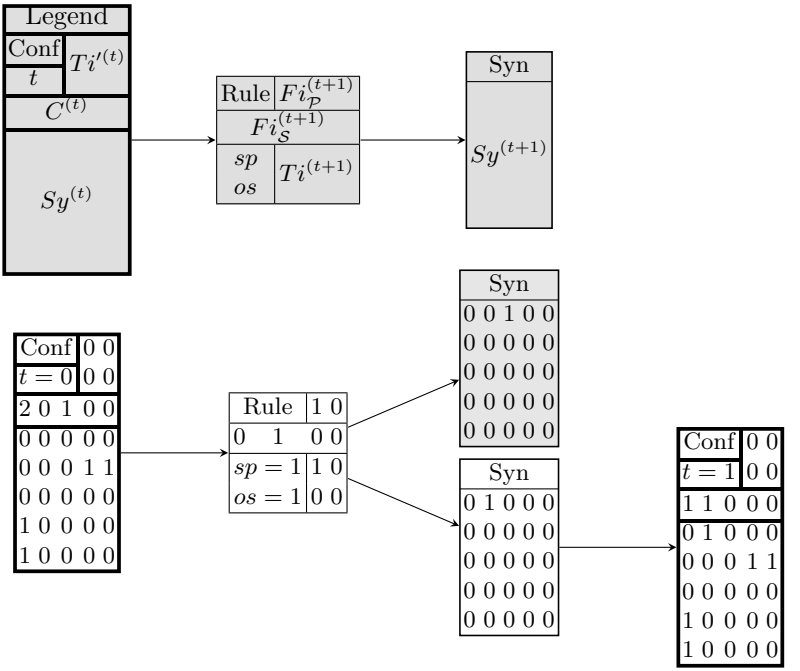


Fig. 2. Incomplete computation tree for Π_{ex} computing {4}

7 Closing Remarks

The algorithm can be sped up by GPU parallelization on the matrix operations. If an implementer decides not to check for uniqueness, then the simulation can be sped up further by performing parallel configuration generations. Otherwise, the configurations would have to be checked sequentially. If the implementer is sure that there would be no loops in the computation tree, then they can opt not to do uniqueness checks.

Future work for this paper includes a software implementation of the algorithm on GPU and CPU using the matrix representation as proof of concept; and simulation results using the software. As previously mentioned, for this work we are only dealing with SNPSP systems with determinism on the rule-level. Lastly, an algorithm is also to be provided for the conversion of such SNPSP systems without this deterministic restriction into another system that has the restriction. We also note that at present, the matrix representation seems to be applicable to *asynchronous* version of SNPSP systems, see e.g. [2]. In asynchronous mode of rule application, as opposed to the synchronous mode in this work, at each step a neuron can nondeterministically choose not to apply a rule even if a rule can be applied. However, the algorithms given in this work must be modified in order to include this additional level of nondeterminism.

References

1. Cabarle, F.G.C., Adorna, H.N., Martínez-del-Amor, M.Á., Pérez-Jiménez, M.J.: Improving GPU simulations of spiking neural P systems. ROMJIST 15(1), 5–20 (2012)
2. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J.: Asynchronous spiking neural P systems with structural plasticity. In: International Conference on Unconventional Computation and Natural Computation. pp. 132–143. Springer (2015)
3. 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)
4. Carandang, J.P., Cabarle, F.G.C., Adorna, H.N., Hernandez, N.H.S., Martinez-del Amor, M.A.: Handling Non-determinism in Spiking Neural P Systems: Algorithms and Simulations. Fundamenta Informaticae 164, 139–155 (2019)
5. Carandang, J.P.A., 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 for Information Science and Technology (ROMJIST) 20(1), 57–70 (2017)
6. Dela Cruz, R.T., Cailipan, D., Cabarle, F.G.C., Hernandez, N., Buño, K., Adorna, H., Carandang, J.: Matrix Representation and Simulation Algorithm for Spiking Neural P Systems with Rules on Synapses. Proc. 18th Philippine Computing Science Congress (PCSC2018), 15–17 March, 2018, Cagayan de Oro City, Misamis Oriental, Philippines pp. 104–112 (2018), `https://sites.google.com/site/2018pcsc/proceedings`
7. Dela Cruz, R.T., Jimenez, Z.B., Cabarle, F.G.C., Hernandez, N., Buño, K., Adorna, H., Carandang, J.: Matrix Representation of Spiking Neural P Systems with Structural Plasticity. Proc. 18th Philippine Computing Science Congress (PCSC2018),

15–17 March, 2018, Cagayan de Oro City, Misamis Oriental, Philippines pp. 104–112 (2018), https://sites.google.com/site/2018pcsc/proceedings
8. Ionescu, M., Păun, G., Yokomori, T.: Spiking neural p systems. Fundamenta informaticae 71(2, 3), 279–308 (2006)
9. Martínez-del-Amor, M.Á., Orellana-Martín, D., Cabarle, F.G.C., Pérez-Jiménez, M.J., Adorna, H.N.: Sparse-matrix Representation of Spiking Neural P Systems for GPU. In: Proc. of 15th Brainstorming Week on Membrane Computing. pp. 161–170. Fénix Editora, Seville, Spain (2017)
10. Păun, G.: Spiking neural p systems. a tutorial. Bulletin of the European Association for Theoretial Computer Science 91, 145–159 (02/2007 2007), http://cs.ioc.ee/yik/schools/win2007/paun/snppalmse.pdf
11. Zeng, X., Adorna, H.N., 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)

A Theorem proofs

Proof for Theorem 1.

Proof. By definition, $Rule^{(k)} = (Fi^{(k)}, Ti^{(k)}, os^{(k)}, sp^{(k)})$. First, given that the input $Conf^{(k-1)}$ is of the previous time step (fed into the function as $Conf^{(k)}$), we first increment k at Line 1 for appropriate usage in the resulting rule node. Thus we know that the `newRules()` constructor at Line 18 is of the right time step. Line 4 evaluates a formula and assigns it to a temporary variable Sp, for spikes. The formula consists of two parts, the multiplication and the subtraction. It goes as follows:

$$\begin{aligned} Sp &= \left(C^{(k-1)} \times Sr_{\mathcal{R}}^{T}\right) - P_{\mathcal{R}} \\ &= \left(\left[c_i^{(k-1)}\right]_m \times \left[sr_{\mathcal{R},i,j}\right]_{m\times r_{\mathcal{R}}}\right) - \left[p_{\mathcal{R},i}\right]_{r_{\mathcal{R}}} = \left(\left[\sum_i c_i^{(k-1)} sr_{\mathcal{R},i,j}\right]_{r_{\mathcal{R}}}\right) - \left[p_{\mathcal{R},i}\right]_{r_{\mathcal{R}}} \end{aligned}$$

Since $sr_{\mathcal{R},i,j} = 1$ if rule $r_{\mathcal{R},i}$ belongs to neuron σ_j (0 otherwise), and $c_i^{(k)}$ is the number of spikes in neuron σ_i at time k, we have

$$c_i^{(k-1)} sr_{\mathcal{R},i,j} = \begin{cases} c_i^{(k-1)}, & r_{\mathcal{R},j} \in R_i; \\ 0, & \text{otherwise.} \end{cases}$$

Also noting that rules can only be associated with one neuron, we can then conclude that $\sum_i c_i^{(k-1)} sr_{\mathcal{R},i,j}$ is the number of spikes in the source neuron of rule $r_{\mathcal{R},j}$. We let $rsp_{\mathcal{R},i}$ be this number. Now that we know each rule's respective source neuron spike count, we can now use the P and Q vectors to check compatibility with the rule's respective regular expression. Thus

$$\begin{aligned} Sp &= \left(\left[\sum_i c_i^{(k-1)} sr_{\mathcal{R},i,j}\right]_{r_{\mathcal{R}}}\right) - \left[p_{\mathcal{R},i}\right]_{r_{\mathcal{R}}} \\ &= \left[rsp_{\mathcal{R},i}\right]_{r_{\mathcal{R}}} - \left[p_{\mathcal{R},i}\right]_{r_{\mathcal{R}}} = \left[rsp_{\mathcal{R},i} - p_{\mathcal{R},i}\right]_{r_{\mathcal{R}}} \end{aligned}$$

With s being the current spike count of a certain neuron, we need to match a^s with $a^p(a^q)* = a^{p+qn}$, and thus we need to make sure $s = p + qn$ for some nonnegative integers p,q, and n. So we first subtract p in Line 4, and check for qn in the if clause of Line 7. There are two cases for a^s to match the regular expression. First, if there is a non-zero q for the rule. If there is, there should be no problem using Sp_i **mod** $q_{\mathcal{R},i} = 0$ (so long as Sp_i isn't negative, in which case $rsp_{\mathcal{R},i} - p_{\mathcal{R},i} = s - p < 0$). The other case would be if $q = 0$, in which case the regular expression is of the form a^p. Thus $p + qn = p$, a constant, and so $s = p + qn$ can only be satisfied if $s - p = rsp_{\mathcal{R},i} - p_{\mathcal{R},i} = 0$. If the regular expression is matched, $fi_{\mathcal{R}}, i^{(k)} = 1$; otherwise, $= 0$. Since the loop of Line 6 iterates over all the rules of type $\mathcal{R}$, and that $\mathcal{R}$ goes through both $\mathcal{P}$ and $\mathcal{S}$ (Line 3), these two loops go over all of the rules. Thus $Fi^{(k)}$ now tells us which rules have matched their regular expressions and can fire.

$os^{(k)}$ would by default copy the value from the previous time step, $os^{(k)}$, while $sp^{(k)}$ would stay at 0. The former would only increase and the latter be set to 1 if an output spike was discovered to be sent to the environment at time k. This condition is checked by the `if` clause at Line 9, which would only be reached if rule $r_{\mathcal{R},i}$ were to fire at time k for the given values of $\mathcal{R}$ and i. Thus, we only need to check if this rule sent an output spike. Since only spiking rules can send spikes to the environment, the condition at Line 9 should check if the given rule was a spiking rule ($\mathcal{R} = \mathcal{S}$) and if the current rule belonged to the output neuron ($Sr_{\mathcal{S},out,i} = 1$). Thus, $os^{(k)}$ and $sp^{(k)}$ are now computed correctly.

Lastly, the timer matrix $Ti^{(k)}$ would only be touched in the `for` loop of Line 12. For each plasticity rule, we first check if the rule already fired at the previous time step (Line 13) and is still executing at the current time step (as with the $\pm$ and $\mp$ rules). This could be checked by looking for a 1 in the primed timers of the said rule from the previous time step ($Ti_i'^{(k-1)}$), since the timers have already counted down after initial rule firing. $fi_{\mathcal{P},i}^{(k)}$ is simply marked as 0 since the rule isn't allowed to fire anew if it is still executing, and just copies the previous primed timers onto the current unprimed timers. Otherwise, if the rule isn't to execute a second operation at the current time step, we check if it fired anew at the current time step (Line 9). Since $Fi_{\mathcal{R}}^{(k)}$ now shows which rules are applicable (unless ongoing execution), we can now be sure that the rules *will be* applied at time step k and thus we start the timer (Line 17). Given that, we are now sure that $Fi^{(k)}$ and $Ti^{(k)}$ are now computed correctly.

Therefore, we are now sure that $os^{(k)}$, $sp^{(k)}$, $Fi^{(k)}$, and $Ti^{(k)}$ are computed correctly. `newRule`() is thus sure to be fed the correct arguments, and will return the correct rule node. □

Proof for Theorem 2.

Proof. Here, we return a list of all possible synapse nodes $Syn^{(k)}$. `getCandidates`() has not been specified in this paper, and is assumed to return a list of all possible combinations of created/deleted synapses based on permutations of destination neurons and synapse counts of applicable plasticity rules. Given this, we are ensured that $Sy_+^{(k)}$ and $Sy_-^{(k)}$ are the appropriate synapse creation and deletion matrices of each synapse node to be created. Thus, $Sy_\Delta^{(k)}$ would then hold the appropriate synapse change matrix for the same synapse node and would thus be appropriately passed onto the constructor for $Syn^{(k)}$ and be included in the return list. Therefore, `getSyns`() returns the correctly computed synapse nodes appropriate for the given rule node. □

Proof for Theorem 3.

Proof. Given the definitions of $Fi_{\mathcal{S}}$, $Sr_{\mathcal{S}}$, and Sy, we have

$$G_{\mathcal{S}}^{(k)} \stackrel{?}{=} Fi_{\mathcal{S}}^{(k)} \times Sr_{\mathcal{S}} \times Sy^{(k)}$$
$$= \left[fi_{\mathcal{S},i}^{(k)}\right]_{r_{\mathcal{R}}} \times \left[sr_{\mathcal{S},i,j}\right]_{r_{\mathcal{R}} \times m} \times \left[sy_{i,j}^{(k)}\right]_{m \times m} = \left[\sum_i fi_{\mathcal{S},i}^{(k)} sr_{\mathcal{S},i,j}\right]_m \times \left[sy_{i,j}^{(k)}\right]_{m \times m}$$

Since $fi_{\mathcal{S},i}^{(k)} = 1$ if rule $r_{\mathcal{S},i}$ has spiked at time k (0 otherwise), and $sr_{\mathcal{S},i,j} = 1$ if $r_{\mathcal{S},i}$ belongs to neuron σ_j (0 otherwise), we have

$$fi_{\mathcal{S},i}^{(k)} sr_{\mathcal{S},i,j} = \begin{cases} 1, & r_{\mathcal{S},i}{}^{\not{t}(k)} \in R_j; \\ 0, & \text{otherwise.} \end{cases}$$

Thus $\sum_i fi_{\mathcal{S},i}^{(k)} sr_{\mathcal{S},i,j}$ is the number of spiking rules that have spiked at time k from neuron σ_j. However, given that we have restricted neurons to only fire a maximum of one rule each, the value of this summation will only ever be 0 or 1, only indicating whether the neuron had a spiking rule fire or not. Continuing further, $sy_{i,j}^{(k)} = 1$ if neuron σ_i is connected to σ_j at time k (0 otherwise), so

$$\begin{aligned} G_{\mathcal{S}}^{(k)} &\stackrel{?}{=} \Big[\sum_i fi_{\mathcal{S},i}^{(k)} sr_{\mathcal{S},i,j}\Big]_m \times \Big[sy_{i,j}^{(k)}\Big]_{m\times m} \\ &= \Big[\sigma_i \overset{\Sigma s}{\underset{(k)}{\rightarrow}} \sigma\Big]_m \times \Big[sy_{i,j}^{(k)}\Big]_{m\times m} = \Big[\sigma_i \overset{s}{\underset{(k)}{\rightarrow}} \sigma\Big]_m \times \Big[\sigma_i \underset{(k)}{\rightarrow} \sigma_j\Big]_{m\times m} \\ &= \Big[\sum_i \Big(\Big(\sigma_i \overset{s}{\underset{(k)}{\rightarrow}} \sigma\Big)\Big(\sigma_i \underset{(k)}{\rightarrow} \sigma_j\Big)\Big)\Big]_m = \Big[\sum_i \sigma_i \overset{s}{\underset{(k)}{\rightarrow}} \sigma_j\Big]_m = \Big[\sigma \overset{\Sigma s}{\underset{(k)}{\rightarrow}} \sigma_j\Big]_m \end{aligned}$$

Spiking rules can only cause spike gains in a destination neuron if some other source neuron fires a spiking rule to the said destination, and so we finally have

$$G_{\mathcal{S}}^{(k)} \stackrel{?}{=} \Big[\sigma \overset{\Sigma s}{\underset{(k)}{\rightarrow}} \sigma_j\Big]_m = \Big[g_{\mathcal{S},j}^{(k)}\Big]_m \stackrel{\checkmark}{=} G_{\mathcal{S}}^{(k)}$$

□

Proof for Theorem 4.

Proof. Since plasticity rules can only cause spike gains by creating synapses (because creating synapses would inherently send one spike to the destination neuron), we only need to check $Sy_{+}^{(k)}$. Given the definition of Sy_{+} we have

$$G_{\mathcal{P}}^{(k)} \stackrel{?}{=} \sum_{i=1}^{r_{\mathcal{P}}} Sy_{+,i}^{(k)} = \Big[\sum_i sy_{+,i,j}^{(k)}\Big]_m = \Big[\sum_i \sigma_i \overset{+}{\underset{(k)}{\rightarrow}} \sigma_j\Big]_m = \Big[\sigma \overset{\Sigma +}{\underset{(k)}{\rightarrow}} \sigma_i\Big]_m = \Big[g_{\mathcal{P},i}^{(k)}\Big]_m \stackrel{\checkmark}{=} G_{\mathcal{P}}^{(k)}$$

□

Proof for Theorem 5.

Proof. Both spiking and plasticity rules can only cause spike loss through spike consumption upon rule firing. Thus,

$$\begin{aligned} L_{\mathcal{R}}^{(k)} &\stackrel{?}{=} (Fi_{\mathcal{R}}^{(k)} \odot Co_{\mathcal{R}}) \times Sr_{\mathcal{R}} \\ &= \Big(\big[fi_{\mathcal{R},i}^{(k)}\big]_{r_{\mathcal{R}}} \odot \big[co_{\mathcal{R},i}\big]_{r_{\mathcal{R}}}\Big) \times \big[sr_{\mathcal{R},i,j}\big]_{r_{\mathcal{R}}\times m} = \Big(\big[r_{\mathcal{R},i}{}^{\not{t}(k)}\big]_{r_{\mathcal{R}}} \odot \big[co_{\mathcal{R},i}\big]_{r_{\mathcal{R}}}\Big) \times \big[sr_{\mathcal{R},i,j}\big]_{r_{\mathcal{R}}\times m} \\ &= \big[r_{\mathcal{R},i}{}^{\not{t}(k)} co_{\mathcal{R},i}\big]_{r_{\mathcal{R}}} \times \big[sr_{\mathcal{R},i,j}\big]_{r_{\mathcal{R}}\times m} = \big[r_{\mathcal{R},i}{}^{\not{t}_c^{(k)}}\big]_{r_{\mathcal{R}}} \times \big[sr_{\mathcal{R},i,j}\big]_{r_{\mathcal{R}}\times m} \\ &= \big[\sum_i r_{\mathcal{R},i}{}^{\not{t}_c^{(k)}} sr_{\mathcal{R},i,j}\big]_m \end{aligned}$$

Since $sr_{\mathcal{R},i,j}$ will only have a nonzero value if rule $r_{\mathcal{R},i}$ is in neuron σ_j, we have

$$r_{\mathcal{R},i} \not\downarrow_c^{(k)} sr_{\mathcal{R},i,j} = \begin{cases} r_{\mathcal{R},i} \not\downarrow_c^{(k)}, & r_{\mathcal{R},i} \in R_j; \\ 0, & \text{otherwise} \end{cases}$$

Spike losses will only ever be caused by spike consumption from rule firing in a given neuron. Thus, also given the definition of $\sigma_j \not\downarrow_{c,\mathcal{R},i}^{(k)}$

$$L_{\mathcal{R}}^{(k)} \stackrel{?}{=} \left[\sum_i r_{\mathcal{R},i} \not\downarrow_c^{(k)} sr_{\mathcal{R},i,j}\right]_m = \left[\sum_i \sigma_j \not\downarrow_{c,R,i}^{(k)}\right]_m = \left[\sigma_i \not\downarrow_{c,R}^{(k)}\right]_m = \left[l_{\mathcal{R},i}^{(k)}\right]_m \stackrel{\checkmark}{=} L_{\mathcal{R}}^{(k)}$$

Lines 9–11 would tick the timer to get $Ti'^{(k)}$, by manually decreasing each element of the matrix by 1 unless the value is 0. □

Proof for Theorem 6.

Proof. By definition, $Conf^{(k)} = (C^{(k)}, Sy^{(k)}, Ti'^{(k)})$. Lines 1 to 8 have been proven to correctly compute for $C^{(k)}$ and $Sy^{(k)}$. The loop in Line 9 iterates over all plasticity rules, while the inner loop of Line 10 goes over the two plasticity operations creation (1) and deletion (2). Line 11 would then either count down the current unprimed timer ($ti_{i,j}^{(k)} - 1$), or keep it at zero (`max`). Thus, the loops correctly compute for $Ti'^{(k)}$. Line 12 thus returns the correct configuration node via the constructor for $Conf^{(k)}$, being passed the correct arguments for $C^{(k)}$, $Sy^{(k)}$, and $Ti'^{(k)}$. □

Proof for Theorem 7.

Proof. The first three lines are just for initialization. The loop in Line 4 iterates over the configuration nodes in a breadth-first manner (seen by the use of `dequeue` and `enqueue`). Line 6 would cut off the computation graph once it reaches a given depth. The loop in Line 9 would go through the rule nodes, connecting them to configuration nodes first before heading to the loop in Line 13. This loop would go through the synapse nodes and connect them to the rule nodes, and then generates a new configuration node in Line 16. These two inner loops, from the rule nodes down to the immediate next configuration nodes, would generate these three levels in a depth-first manner (as seen with `pop` and `push`). Essentially, what happens is (1) given a configuration node, generate the subtree of these configuration nodes up to three levels in depth-first manner, (2) go through these configuration nodes in breadth-first manner.

□

MCFDS: Membrane Computing Fault Diagnosis System for Power Systems

Kang Yi[1], Haina Rong[1], Jianping Dong[1], Gexiang Zhang[1] ⋆, Prithwineel Paul[1], and Zhiwei Huang[2]

[1] School of Electrical Engineering, Southwest Jiaotong University, Chengdu, 610031, China
[2] Beijing National Railway Research & Design Institute of Signal & Communication Group Co., Ltd, Chengdu Branch, Chengdu, 610000, China
zhgxdylan@126.com

Abstract. As an important variant of membrane computing models, fuzzy reasoning spiking neural P systems (FRSN P system) was introduced to build a link between P systems and fault diagnosis applications. FRSN P system offers an intuitive illustration based on a strictly mathematical expression, a good fault-tolerant capacity, a good description for the relationships between protective devices and faults, and an understandable diagnosis model-building process. However, the implementation of FRSN P system is a manual process, which is a time-consuming and very hard task, especially impossible to perform for large scale networks. In this work we developed a software system for automatically fulfilling the task, named as Membrane computing fault diagnosis system (MCFDS). The system consists of input, output and four subsystems containing network topology analysis, suspicious fault component analysis, construction of FRSN P system for suspicious fault components and fuzzy inference. Also, the feasibility of the FRSN P system is checked on the IEEE14 node system.

Keywords: Membrane computing; P system; Fuzzy reasoning spiking neural P system; Fault diagnosis; Power system

1 Introduction

As a branch of natural computing, membrane computing was introduced by Păun in 1998 [1]. The distributed parallel computational model is called a membrane system or a P system. Membrane computing aims to investigate the computational models and their applications abstracted from the structure and functioning of cells [3]. A large number of studies show that many variants of P systems are Turing complete [2,4,6,7,29]. Moreover, distributive, maximally parallel and expansibility [8] make P systems suitable for solving a variety of practical problems [5,10,13,26,37].

With the development of membrane computing, many types of membrane systems are proposed [11,12,14,15,16,17]. The spiking neural P systems is a hot research topic of neural P systems [19,20,21,22,24,25], which was introduced by Ionescu et al in 2006 [18]. Fuzzy reasoning spiking neural P system (FRSN P system) was introduced to build

⋆ Corresponding author.

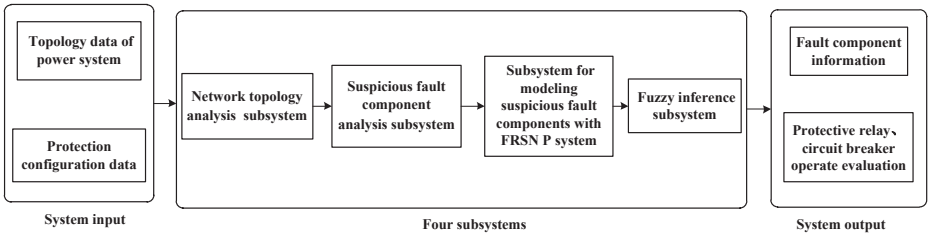


Fig. 1. System frame structure

a bridge between the P systems and fault diagnosis for electric power systems [8,28]. FRSN P system offers an intuitive illustration based on a strictly mathematical expression, a good fault-tolerant capacity, a good description for the relationships between protective devices and faults, and an understandable diagnosis model-building process [8,23,40]. According to the investigations reported in literature, FRSN P system have been successfully used to diagnose the faults occurring in transformers [28,44], power transmission networks [40], traction power supply systems of high-speed railways [38], metro traction power systems [41] and fault classification of power transmission lines [43].

In [39], several questions on FRSN P system were mentioned. Is FRSN P system suitable for large-scale power transmission networks? How is the complexity of the FRSN P system? Is the FRSN P system performance superior to other diagnosis methods like Petri Nets, with respect to diagnosis time, fault section misinformation rate, fault section missing rate and computational complexity? Until now, FRSN P system for fault diagnosis is implemented in a manual way and therefore it is impossible to provide the answers to these questions. This is the motivation for this work.

In this paper, a software system has been developed for automatically fulfilling the task and it is called Membrane Computing Fault Diagnosis System (MCFDS). The system consists of input, output and four subsystems: network topology analysis, suspicious fault component analysis, construction of FRSN P system for suspicious fault components and fuzzy inference. Furthermore, the feasibility of MCFDS is checked on the IEEE14 node system.

2 MCFDS

The Membrane Computing Fault Diagnosis System (MCFDS) consists of mainly three components, i.e., input, output and subsystems. The main components of the subsystems are network topology analysis, suspicious fault component analysis, construction of FRSN P system for suspicious fault components and fuzzy inference. The input data is composed of topology data of power systems and protection configuration data. The outputs include fault component information, protective relays information and circuit breakers operation evaluation. The schematic structure of MCFDS is shown in Fig. 1.

2.1 Input Data

The source of the information of the fault diagnosis program is the grid static data and switch state data based on fault information system. In this paper we use access database to store network topology information and protection configuration information. The static topology information and protection configuration information of the power network give as input into the access database to form the topology table and protection configuration table. Therefore, the input data of FRSN P system for fault diagnosis consists of two parts: topology data of a power system and protection configuration data.

2.1.1 Topology Data of Power Systems

In this paper we mainly discuss the fault diagnosis methods in power transmission network. Moreover, this paper improves the traditional line analysis [9] and redefines the components, i.e., transmission lines, busbars, transformers and generators. The circuit breakers work as switches.

The "Component" and "Switche" that appear in the following sections are defined in the following manner. A Power system is made up of components and switch devices which connect a variety of other equipments. So, the whole electric power system grid network can be represented by the power transmission network topology as shown in Fig. 2. The components shown in the figure refer to the transmission lines, busbars, transformers and generators. The switches refer to the circuit breakers with two states: open and closed.

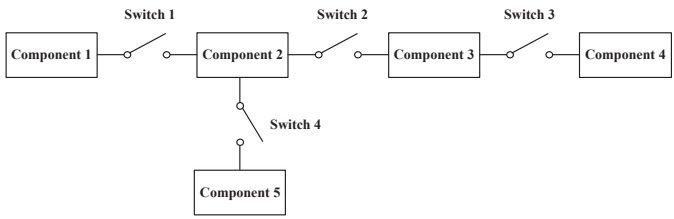


Fig. 2. Schematic diagram of power transmission network topology

After there is a failure, because of the grid power system components and complex wiring, it is very difficult to find the faults in huge systems. But whenever fault occurs in a power system, the protective relays and circuit breakers will operate to isolate the fault. We investigate the actions of protective relays and tripped circuit breakers in the network and the connection relationship between them. Fig. 2 shows a simple and clear power transmission network topology of the connection relation between components and circuit breakers. Then the fault component is searched according to the tripped circuit breakers.

The following topology table is constructed from the components and switches of the entire power transmission network topology. The table stores the data of the main

components and switches along with their connection relationships and the protection number associated with each component.

ID number	Switches operation	Component /switch	Type	Type number	Associated components/switches	Associated protective relays
10108	--	B08	Bus	101	CB0807.CB08G2	10110108. 21310214

Fig. 3. The topology table

2.1.2 Protection Configuration Data

The SCADA system can provide the tripped circuit breakers and operated protective relays information whenever there is a fault in the grid. Moreover, the data of the components, protective relays and circuit breakers are used to construct the correlation database.

The correlation relationships are introduced in [30,33,34] in the following manner:

Component - Protective relay means that the protective relays can be divided into the main protective relays of the component and one of the first backup protective relays;

Protective relay - Switch means that the circuit breaker can trip in principle once the protective relays operation is performed;

Component1 - Component2 relates to the scope of protection of the second backup protective relay of component1 which can protect component2.

With these correlations, the protection configuration table can be described as follows:

ID number	Protective relay	Relays operation	Operated switches	Protected equipment
10110113	Main_ protective relay	0	10426.10438.10439	B13

Fig. 4. The protection configuration table

2.2 Network Topology Analysis Subsystem

After the failure in the power system transmission network, the fault component is eventually isolated by tripped circuit breakers. Moreover, the fault component will be isolated in the passive network. We have elaborated in section 2.1, the entire topology database and protective relay database. Also, have established the corresponding topology table which represents the correlation between the whole transmission network topology structure and protective relays. Whenever a failure occurs in the power

system transmission network, at first the information is received from SCADA system by circuit breaker opening and closing state, and then the suspicious fault component is found by using the network topology analysis method. The specific network topology analysis method is as follows:

(1) Set up M, and store all component IDs into M;
(2) Set up the subset N. Take a component from the set M and put it in the subset N. Find all closed circuit breakers connected to it. If there is no closed circuit breaker, then turn to step (5);
(3) Identify the components connected to the closed circuit breaker and add the found components to the subset N;
(4) Continue to search for closed circuit breakers that are connected to the components in step (3) (except for circuit breakers used in step (3)). If there is a closed circuit breaker, go to step (3);
(5) Remove all components in the set M that appear in the subset collection N. If the set M is not empty, then transfer to step (2);
(6) List all the subsets N.

In fault diagnosis, the network topology analysis method is used to search all subsets, and then the passive networks are found from these subsets. This passive networks are the outage areas. In this way, the diagnosis scope can be reduced and then the suspicious fault component is diagnosed. It also reduces the amount of operations and improves the efficiency of fault diagnosis. The process of the searching of the passive networks is shown in Fig. 5.

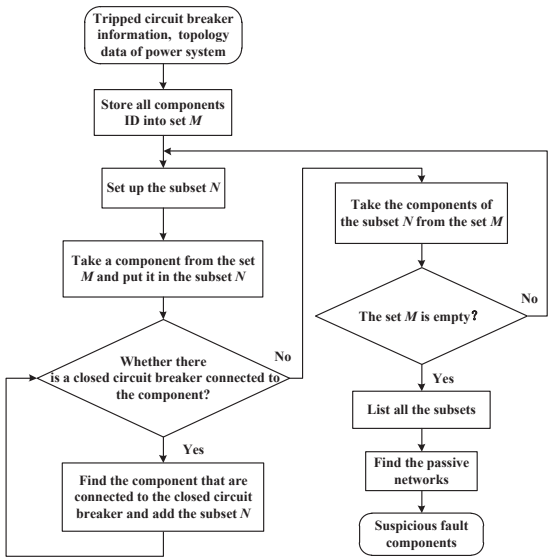


Fig. 5. Search the outage area flow chart

2.3 Suspicious Fault Component Analysis Subsystem

The network topology analysis method is used to find a passive network and the diagnosis of the suspicious fault components in the passive network. The modelling of FRSN P systems is very complex because of the existence of many components in the complex grid network. In order to improve the diagnosis efficiency and accuracy of the algorithm, in this paper, we introduce the concept of suspected fault component logic analysis. At first a logic diagram is constructed in such a manner that the suspected fault component is considered as the starting point. Then it searches and builds towards each connection to protect the component within the scope of the protection of all components and switches. The FRSN P system model is constructed according to the fault production rules of the suspected fault components. Hence the fault area is reduced and the fault component is identified.

In the logic diagram the suspicious component along with other system components and switches in the passive network are represented by a node and the edge between the two nodes represents the connection between the components and switches. The directional of the protective relay is used as the direction information. Moreover, the condition of the path search termination is:

(1) The search is complete when all the protective relays and switches that can protect the suspicious fault components are searched on different paths.
(2) If the search path is disconnected from the peripheral device due to normal operations (such as the operation of the blade, etc.), then the search will be terminated.
(3) If the search direction is opposite to the rule, the search path will terminate.
(4) Search for a loop structure or parallel edge structure on the search path, and if it exists then terminate this direction search.

The logic diagram of the suspected fault component describes the topological association of the suspected fault components and its associated protection in the power grid. The following example illustrates the method of forming the logic diagram of the suspected component. In Fig. 6, it is assumed that the suspicious fault component is B_3 by the method of network topology analysis. The logic diagram of the suspected fault component is established by bus B_3 in three paths: $B_3 \longrightarrow CB_5 \longrightarrow L_3 \longrightarrow CB_2$; $B_3 \longrightarrow CB_6 \longrightarrow L_4 \longrightarrow CB_7$; $B_3 \longrightarrow CB_9 \longrightarrow L_5 \longrightarrow CB_{10}$, respectively. Moreover, the mutual cooperation between the protective relay and circuit breaker will cut off the connection with the whole grid.

2.4 Subsystem For Modeling Suspicious Fault Components with FRSN P system

Before performing the reasoning algorithm, we need to build a FRSN P system diagnosis model. A local grid is shown in Fig 6 and the network topology analysis subsystem obtains the bus B_3 as the suspicious component. Also, the bus B_3 and line L_4 are used to build the FRSN P system fault diagnosis model.

At first, the bus B_3 in Fig. 6 is used to describe the model where the fault confidence level of bus B_3 is the value of output of the FRSN P system. Moreover, we discussed the mutual cooperation between the protective relay and circuit breaker by the suspicious

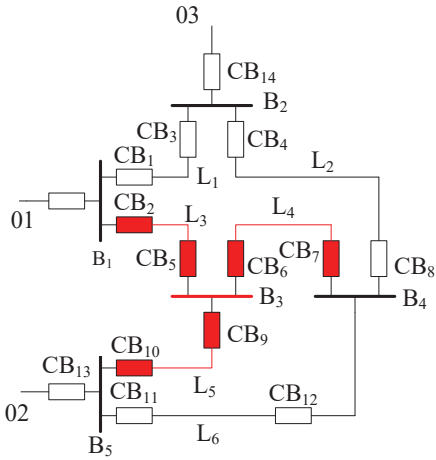


Fig. 6. A local grid

fault component logic diagram in Section 2.3. The fault production rules of bus B_3 are described as follows:

R_1: *IF* (B_{3m} *operates and* CB_5 *trips*) *OR* (L_{3Ss} *operates and* CB_2 *trips*) *THEN* B_3 *faults* (*CF* $= c_i$),

R_2: *IF* (B_{3m} *operates and* CB_6 *trips*) *OR* (L_{4Rs} *operates and* CB_7 *trips*) *THEN* B_3 *faults* (*CF* $= c_i$),

R_3: *IF* (B_{3m} *operates and* CB_9 *trips*) *OR* (L_{5Ss} *operates and* CB_{10} *trips*) *THEN* B_3 *faults* (*CF* $= c_i$).

Following fault diagnosis model based on FRSN P system for bus B_3 are built according to these fault production rules shown in Fig. 7(a). The FRSN P system Π is a construct of the form:

$$\Pi = (O, \sigma_{p1}, \ldots, \sigma_{p22}, \sigma_{r1}, \ldots, \sigma_{r10}, syn, in, out)$$

where

(1) $O = \{a\}$ is the singleton alphabet (a is called spike);

(2) $\sigma_{p1}, \ldots, \sigma_{p22}$ are proposition neurons corresponding to the propositions with fuzzy truth values $\theta_1, \theta_2, \ldots, \theta_{22}$;

(3) $\sigma_{r1}, \ldots, \sigma_{r10}$ are rule neurons, where $\sigma_{r1}, \ldots, \sigma_{r6}$ and σ_{r10} are and rule neurons and $\sigma_{r7}, \sigma_{r8}, \sigma_{r9}$ are or rule neurons. A real number $c_i \in [0, 1]$ is used to represent the certainty factor (*CF*) of the fuzzy production rule associated with σ_{ri} $(1 \leq i \leq 10)$;

(4) $syn \subseteq \{1, 2, \ldots, 22\} \times \{1, 2, \ldots, 10\}$ with $i \neq j$ for all $(i, j) \in syn, 1 \leq i, j \leq 22$, is a directed graph of synapses between the linked neurons;

(5) $in = \{\sigma_{p1}, \ldots, \sigma_{p12}\}$, $out = \{\sigma_{r10}\}$.

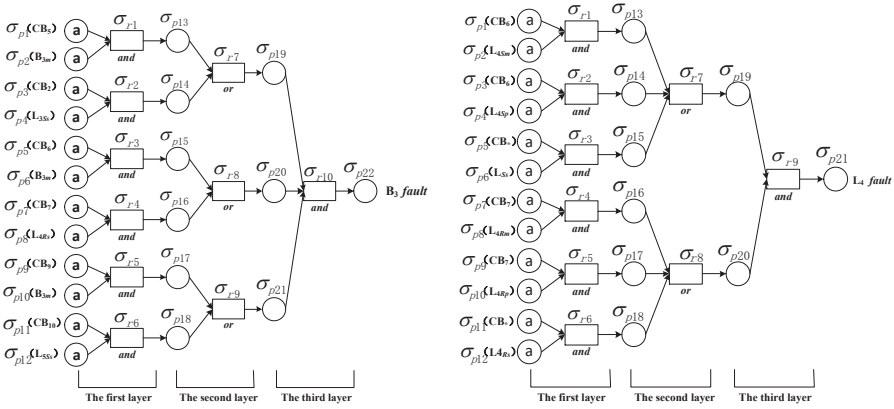
The transmission line L_4 in Fig. 6 is used to describe model building of transmission line, where the fault confidence level of transmission line L_4 is the value of output of

the FRSN P system. The fault production rules of transmission line L_4 are described as follows:

R_1: *IF* (L_{4Sm} *operates and* CB_6 *trips*) *OR* (L_{4Sp} *operates and* CB_6 *trips*) *OR* (*Second backup protection operates and* CB *trips*) *THEN* L_4 *faults* ($CF = c_i$),

R_2: *IF* (L_{4Rm} *operates and* CB_7 *trips*) *OR* (L_{4Rp} *operates and* CB_7 *trips*) *OR* (*Second backup protection operates and* CB *trips*) *THEN* L_4 *faults* ($CF = c_i$).

Therefore, fault diagnosis model based on FRSN P system for transmission line L_4 are built according to the fault production rules shown in Fig. 7(b).



(a) B_3 FRSN P system Model (b) L_4 FRSN P system Model

Fig. 7. Suspect fault components FRSN P system model

In order to make the reasoning reflect the operation of the actual power grid more accurately, the uncertain factors in the protection and the circuit breaker action informations are obtained from the power system dispatching center. In this paper, the initial confidence level setting of operate and non-operate protective relays and circuit breakers are given in [45], as shown in Table 1 and Table 2 respectively. At the same time, considering the uncertainty of the rule credibility, the certainty factor c_i of each fuzzy production rule is considered to be 0.95.

Table 1. Confidence levels of operate protective devices

Components	Main		First backup		Second backup	
Name	Relays	CBs	Relays	CBs	Relays	CBs
Bus	0.8564	0.9833	-	-	0.7	0.75
Line	0.9913	0.9833	0.8	0.85	0.7	0.75

Table 2. Confidence levels of non-operate protective devices

Components	Main		First backup		Second backup	
Name	Relays	CBs	Relays	CBs	Relays	CBs
Bus	0.2	0.2	-	-	0.2	0.2
Line	0.2	0.2	0.2	0.2	0.2	0.2

In this paper, a proposition neuron is used for all second backup protective relays and circuit breakers at both ends of the line. If there are multiple second backup protective relays, a factor μ is applied before the confidence level of the proposition neuron, and the two ends of the line (*S* end and *R* end) respectively.

$$\mu_1 = \frac{Number\ of\ protective\ relays\ (circuit\ breaker)\ for\ S\ end\ operation}{Number\ of\ all\ protective\ relays\ (circuit\ breakers)\ on\ the\ S\ end} \quad (1)$$

$$\mu_2 = \frac{Number\ of\ protective\ relays\ (circuit\ breaker)\ for\ R\ end\ operation}{Number\ of\ all\ protective\ relays\ (circuit\ breakers)\ on\ the\ R\ end} \quad (2)$$

The automatic generation of the FRSN P system model is shown below:

Step 1: According to the suspected fault component logic diagram, take one path of the suspected fault component logic diagram, and the components and switches involved in the path can be expressed as a fault production rule.
Step 2: Set up two set θ and δ, where θ is the proposition neurons corresponding to the propositions with fuzzy truth values. The initial value setting of the corresponding protective relay and circuit breaker of the first layer of proposition neurons in the FRSN P system model is set by the information of the input from SCADA system. δ is the certainty factor which is added to describe the credibility of the fuzzy generated rules of the neuron.
Step 3: Store the values of θ and δ according to the confidence levels of operate and non-operate protective relays and circuit breakers in the first path.
Step 4: Repeat the first three steps until all the values of θ and δ represented by the paths are added corresponding to one branch direction.

2.5 Fuzzy Inference Subsystem

After obtaining the confidence level of the proposition expressed by the proposition neuron and the certainty factor value of the rule neuron, the next step is to carry out the reasoning operation. By executing the following reasoning algorithm, the fuzzy values of the propositions are represented by the output proposition neurons which can be obtained quickly and simply. Specific algorithm steps [40] are as follows (where s represents the number of proposition neurons, t represents the number of rule neurons and $s + t = m$):

Step 1: Set the initial state to $g = 0$. Set the termination condition to $\mathbf{0} = (0, \ldots, 0)^T_{1\times t}$. The initial values of $\boldsymbol{\theta}$ and $\boldsymbol{\delta}$ are set to $\boldsymbol{\theta}_g = (\theta_{1g}, \theta_{2g}, \ldots, \theta_{sg})^T$ and $\boldsymbol{\delta}_g = (\delta_{1g}, \delta_{2g}, \ldots, \delta_{tg})^T$, respectively;

Step 2: g is increased by one;

Step 3: The firing condition of each input neuron $(g = 1)$ or each proposition neuron $(g > 1)$ is evaluated. If the condition is satisfied and there is a presynaptic rule neuron, the neuron fires and transmits a spike to the next rule neuron. Compute the fuzzy truth value vector $\boldsymbol{\delta}_g$ according to (3):

$$\boldsymbol{\delta}_g = (\boldsymbol{D}_1^T \otimes \boldsymbol{\theta}_{g-1}) + (\boldsymbol{D}_2^T \oplus \boldsymbol{\theta}_{g-1}) + (\boldsymbol{D}_3^T \odot \boldsymbol{\theta}_{g-1}) \tag{3}$$

Step 4: If $\boldsymbol{\delta}_g = \mathbf{0}_1$, then the algorithm stops and output the reasoning results. Otherwise, the algorithm continues.

Step 5: Evaluate the firing condition of each rule neuron. If the condition is satisfied, then the rule neuron fires and transmits a spike to the next proposition neuron. Further, compute the fuzzy truth value vector $\boldsymbol{\theta}_g$ according to (4). Then, the algorithm goes to Step 2:

$$\boldsymbol{\theta}_g = \boldsymbol{E}^T \odot (\boldsymbol{C} \otimes \boldsymbol{\delta}_g) \tag{4}$$

Parameter vectors and matrices are described in the following manner:

(1) $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_s)^T$ is a real truth value vector of *s* proposition neurons. $\theta_i (1 \leq i \leq s)$ is a real number in $[0, 1]$ and represents the potential value contained in the ith proposition neuron. If there is no spike in a proposition neuron, its potential value is 0.

(2) $\boldsymbol{\delta} = (\delta_1, \delta_2, \ldots, \delta_t)^T$ is a real truth value vector of *t* rule neurons. $\delta_j (1 \leq j \leq t)$ is a real number in $[0, 1]$ and represents the potential value contained in the jth rule neuron. If there is no spike contained in a rule neuron, its potential value is 0.

(3) $\boldsymbol{C} = diag(c_1, c_2, \ldots, c_t)$ is a diagonal matrix, where $c_j (1 \leq j \leq t)$ is a real number in [0, 1] representing the certainty factor of the jth fuzzy production rule.

(4) $\boldsymbol{D}_1 = (d_{ij})_{s\times t}$ is a synaptic matrix representing the directed connection from proposition neurons to general rule neurons. If there is a directed arc (synapse) from the proposition neuron σ_i to the *general* rule neuron σ_j, $d_{ij} = 1$, otherwise, $d_{ij} = 0$.

(5) $\boldsymbol{D}_2 = (d_{ij})_{s\times t}$ is a synaptic matrix representing the directed connection from proposition neurons to *and* rule neurons. If there is a directed arc (synapse)from the proposition neuron σ_i to the *and* rule neuron σ_j, $d_{ij} = 1$, otherwise, $d_{ij} = 0$.

(6) $\boldsymbol{D}_3 = (d_{ij})_{s\times t}$ is a synaptic matrix representing the directed connection between proposition neurons and *or* rule neurons. If there is a directed arc (synapse) from the proposition neuron σ_i to the *or* rule neuron σ_j, $d_{ij} = 1$, otherwise, $d_{ij} = 0$.

(7) $\boldsymbol{E} = (e_{ji})_{t\times s}$ is a synaptic matrix representing the directed connection between rule neurons and proposition rule neurons. If there is a directed arc (synapse) from the rule neuron σ_j to the proposition neuron σ_i, $e_{ji} = 1$, otherwise, $e_{ji} = 0$.

Subsequently, we introduce the following three multiplication operations:

(1) $\boldsymbol{D}^T \otimes \boldsymbol{\theta} = (\bar{d_1}, \bar{d_2}, \ldots, \bar{d_t})^T$, where $\bar{d_i} = d_{1j} * \theta_1 + d_{2j} * \theta_2 + \ldots + d_{sj} * \theta_s$, $j = 1, 2, \ldots, t$.
(2) $\boldsymbol{D}^T \oplus \boldsymbol{\theta} = (\bar{d_1}, \bar{d_2}, \ldots, \bar{d_t})^T$, where $\bar{d_i} = min\{ d_{1j} * \theta_1, \ldots, d_{sj} * \theta_s\}$, $j = 1, 2, \ldots, t$.
(3) $\boldsymbol{E}^T \odot \boldsymbol{\theta} = (\bar{e_1}, \bar{e_2}, \ldots, \bar{e_s})^T$, where $\bar{e_i} = max\{ e_{1i} * \delta_1, \ldots, e_{ti} * \delta_t\}$, $i = 1, 2, \ldots, s$.

Through the network topology analysis algorithm to find the passive region, and to diagnose the suspicious fault components in the passive region one by one. At first, the suspicious fault component analysis subsystem is called to form the logic diagram of the suspicious fault components, and then the FRSN P system diagnosis model is formed by the query protection configuration data. Finally, by calling FRSN P system inference algorithm, the fault confidence levels of the suspected fault components is obtained and the fault of the components is determined to realize the whole diagnosis process.

3 Experiments

To verify the effectiveness of the programmed implementation, the IEEE14 node power system network model shown in Fig. 8 is tested. It is a 14-bus system containing $B_{01} \sim B_{14}$ 14 buses and $L_{0102} \sim L_{1314}$ 20 lines. The protective device consists of 134 protective relays, 40 circuit breakers, and $B_{01m} \ldots B_{14m}$ are the main protective relays of the buses. Moreover, L_{XSm} and L_{XRm} are the main protective relays on both ends of the line, L_{XSp} and L_{XRp} are the first backup protective relays on both ends of the line, L_{XSs} and L_{XRs} are the second backup protective relays on both ends of the line (where *X* represents the ID of the line). Before diagnosing the suspected fault component, the components and switches of the entire power transmission network must adopt the method described in the section 2.1 to construct their respective topology table.The detailed protection configuration of various components, as well as the relationship between the protective relays and the circuit breakers have been represented by the corresponding database in Fig. 9 and Fig. 10. Furthermore, the whole diagnosis process is carried out in the MATLAB environment. We give an example to illustrate the steps of MCFDS.

$Case$: $bus\ B_{13}\ has\ a\ fault$

(I). Operated relays: B_{13m}; Tripped circuit breakers: $CB_{1306}, CB_{1312}, CB_{1314}$.

(II). Call the network topology analysis algorithm to get the passive zone $\{13\}$.

(III). To query the topology database of power system, the corresponding logic diagram of suspected fault components is formed, as shown in Fig. 11.

(IV). According to logic diagram of suspected fault components, query protection configuration database is used to form a FRSN P system model, as shown in Fig. 12.

(V). The protective relays and circuit breakers operation information are obtained from the dispatching center. Then according to Tables 1 and 2, the corresponding proposition neurons are constructed in the FRSN P system model with initial values, and a pulse is inserted into each initial proposition neuron to start the ignition.

(VI). The FRSN P system algorithm is invoked to perform inference operations on suspicious fault components.

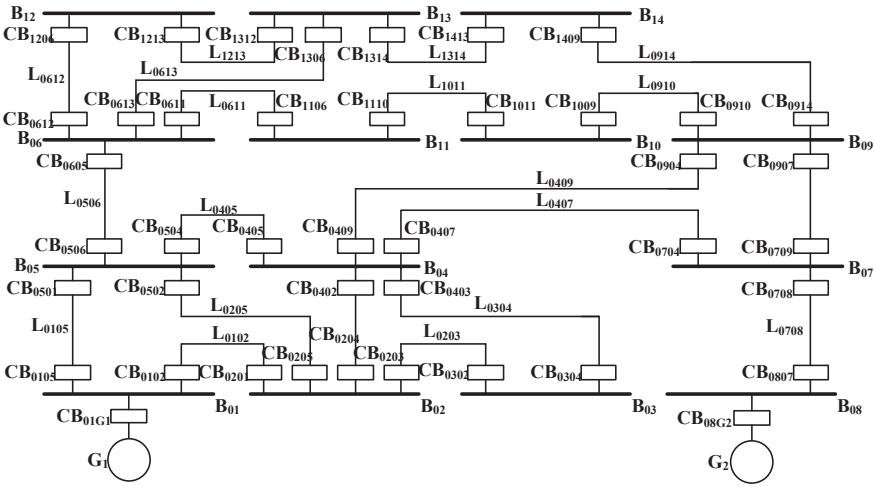


Fig. 8. IEEE14 node power system network

Topology data

ID number	Switch operation	Component	Type	Type	Associated components/switches	Associated protective relays
10101	—	B01	Bus	101	CB0101. CB0102. CB0105	10110101. 22310201. 22310202
10102	—	B02	Bus	101	CB0201. CB0203. CB0204. CB0205	10110102. 21310201. 22310203. 22310204
10103	—	B03	Bus	101	CB0302. CB0304	10110103. 21310203. 22310206
10104	—	B04	Bus	101	CB0402. CB0403. CB0405. CB0407. CB0409	10110104. 21310204. 21310206. 22310207
10105	—	B05	Bus	101	CB0501. CB0502. CB0504. CB0506	10110105. 21310202. 21310205. 21310207
10106	—	B06	Bus	101	CB0605. CB0611. CB0612. CB0613	10110106. 21310210. 22310211. 22310212
10107	—	B07	Bus	101	CB0704. CB0708. CB0709	10110107. 21310208. 22310214. 22310215
10108	—	B08	Bus	101	CB0807. CB08G2	10110108. 21310214
10109	—	B09	Bus	101	CB0904. CB0907. CB0910. CB0914	10110109. 21310209. 21310215. 22310216
10110	—	B10	Bus	101	CB1009. CB1011	10110110. 21310216. 22310218
10111	—	B11	Bus	101	CB1106. CB1110	10110111. 21310211. 21310218
10112	—	B12	Bus	101	CB1206. CB1213	10110112. 21310212. 22310219
10113	—	B13	Bus	101	CB1306. CB1312. CB1314	10110113. 21310213. 21310219. 22310220
10114	—	B14	Bus	101	CB1409. CB1413	10110114. 21310217. 21310220
10201	—	L0102	Line	102	B01. B02	21110201. 22110201. 21210201. 22210201
10202	—	L0105	Line	102	B01. B05	21110202. 22110202. 21210202. 22210202
10203	—	L0203	Line	102	B02. B03	21110203. 22110203. 21210203. 22210203
10204	—	L0204	Line	102	B02. B04	21110204. 22110204. 21210204. 22210204
10205	—	L0205	Line	102	B02. B05	21110205. 22110205. 21210205. 22210205
10206	—	L0304	Line	102	B03. B04	21110206. 22110206. 21210206. 22210206
10207	—	L0405	Line	102	B04. B05	21110207. 22110207. 21210207. 22210207
10208	—	L0407	Line	102	B04. B07	21110208. 22110208. 21210208. 22210208
10209	—	L0409	Line	102	B04. B09	21110209. 22110209. 21210209. 22210209
10210	—	L0506	Line	102	B05. B06	21110210. 22110210. 21210210. 22210210
10211	—	L0611	Line	102	B06. B11	21110211. 22110211. 21210211. 22210211

Fig. 9. IEEE14 node power system network topology database

Protection configuration data

ID number	Protective relay	Relays operation	Operated switches	Protected equipment
10110101	Main_protective relay	0	10401. 10403. 10501	B01
10110102	Main_protective relay	0	10402. 10405. 10407. 10409	B02
10110103	Main_protective relay	0	10406. 10411.	B03
10110104	Main_protective relay	0	10408. 10412. 10413. 10415. 10417	B04
10110105	Main_protective relay	0	10404. 10410. 10414. 10419	B05
10110106	Main_protective relay	0	10420. 10421. 10423. 10425	B06
10110107	Main_protective relay	0	10416. 10427. 10429	B07
10110108	Main_protective relay	0	10428. 10442	B08
10110109	Main_protective relay	0	10418. 10430. 10431. 10433	B09
10110110	Main_protective relay	0	10432. 10435	B10
10110111	Main_protective relay	0	10422. 10436	B11
10110112	Main_protective relay	0	10424. 10437	B12
10110113	Main_protective relay	0	10426. 10438. 10439	B13
10110114	Main_protective relay	0	10434. 10440	B14
21110201	Main_protective relay	0	10401	L0102
21110202	Main_protective relay	0	10403	L0105
21110203	Main_protective relay	0	10405	L0203
21110204	Main_protective relay	0	10407	L0204
21110205	Main_protective relay	0	10409	L0205
21110206	Main_protective relay	0	10411	L0304
21110207	Main_protective relay	0	10413	L0405
21110208	Main_protective relay	0	10415	L0407
21110209	Main_protective relay	0	10417	L0409
21110210	Main_protective relay	0	10419	L0506
21110211	Main_protective relay	0	10421	L0611

Fig. 10. IEEE14 node power system network protection configuration database

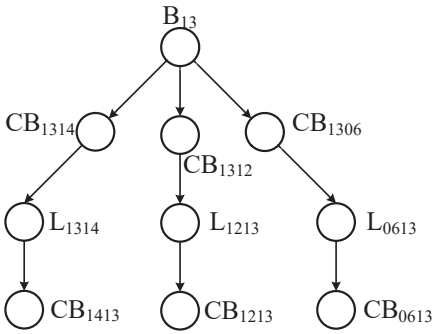


Fig. 11. B_{13} Logic diagram

The reasoning process is as follows:

(1) Parameter initialization: $g = 0$, $\boldsymbol{\theta}_0$ and $\boldsymbol{\delta}_0$ are set according to the pulse values contained in each neuron, i.e., $\boldsymbol{\theta}_0 = (0.9833, 0.8564, 0.2, 0.2, 0.9833, 0.8564, 0.2, 0.2, 0.9833, 0.8564, 0.20.2, 0, \ldots, 0)^T$, $\boldsymbol{\delta}_0 = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T$.

(2) $g = 1$,
$\boldsymbol{\theta}_1 = (0, \ldots, 0, 0.8136, 0.19, 0.8136, 0.19, 0.8136, 0.19, 0, 0, 0, 0)^T$,
$\boldsymbol{\delta}_1 = (0.8564, 0.2, 0.8564, 0.2, 0.8564, 0.2, 0, 0, 0, 0)^T$.

(3) $g = 2$,
$\boldsymbol{\theta}_2 = (0, \ldots, 0, 0.7729, 0.7729, 0.7729)^T$,
$\boldsymbol{\delta}_2 = (0, 0, 0, 0, 0, 0, 0.8136, 0.8136, 0.8136, 0)^T$.

(4) $g = 3$,
$\boldsymbol{\theta}_3 = (0, \ldots, 0, 0.7343)^T$,

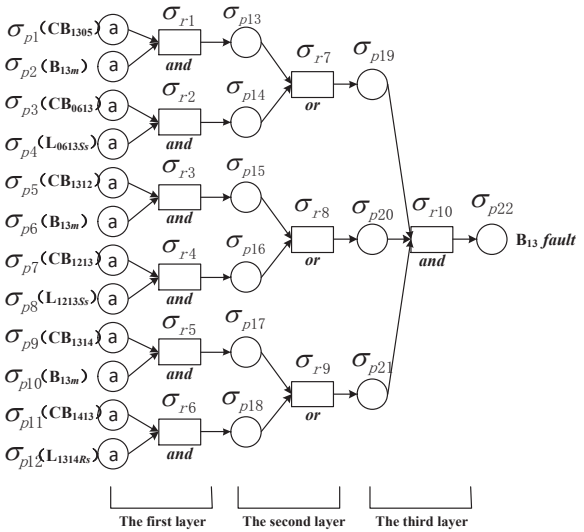


Fig. 12. B_{13} FRSN P system Model

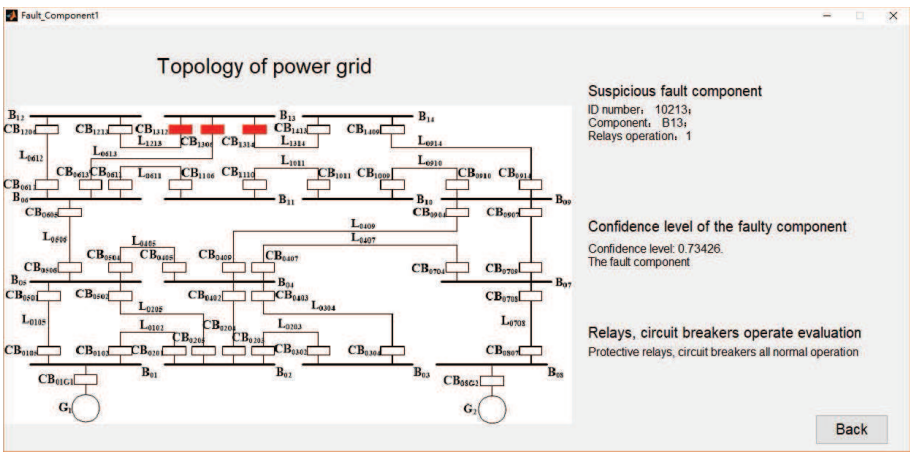


Fig. 13. Fault diagnosis program simulation results

$\boldsymbol{\delta}_3 = (0,0,0,0,0,0,0,0,0,0.7729)^T$.

(5) $g = 4, \boldsymbol{\delta}_4 = (0,0,0,0,0,0,0,0,0,0)^T = \mathbf{0}$. The termination condition is satisfied and output is obtained.

The fault component confidence level of bus B_{13} is 0.7343, according to the decision rule. Bus B_{13} is the fault component. The diagnostic results in the form of a graphical user interface have been displayed in Fig. 13.

(VII). Protective relays and circuit breakers perform all operations normally.

4 Conclusions

In this paper, the whole diagnosis process of power system fault diagnosis is based on FRSN P system and is realized by programming. The whole diagnostic algorithm includes: data structure module, power grid topology analysis algorithm module, suspicious fault component logic analysis algorithm module, and FRSN P system inference algorithm module. IEEE14 node is discussed to verify the diagnosis algorithm program. The diagnosis of the results are consistent with manual calculation results, which verifies the feasibility and reliability of automatic diagnosis algorithm procedures. The FRSN P system is used for fault diagnosis of the entire program diagnostic algorithm and also the speed and complexity of the various modules have been studied. Moreover, the programming method can be adopted to improve the automation of fault diagnosis to explore the superiority of fault diagnosis method based on FRSN P system in large scale power grid fault diagnosis with more nodes.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (61672437, 61702428), the Sichuan Science and Technology Program (2018GZ0185,2018GZ0086), New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0044) and the Fundamental Research Funds for the Central Universities (2682016CX038).

References

1. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences, 61(1), 108-143 (2000).
2. Păun, Gh.: Membrane Computing: An Introduction. Berlin Heidelberg: Springer-Verlag (2002).
3. Păun, Gh.: Membrane Systems: A quick introduction. Lecture Notes in Computer Science(UPP2004), 3566, 188-195 (2005).
4. Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J.: Sequential spiking neural P systems with structural plasticity based on max/min spike number. Neural Computing and Applications, 27(5), 1337-1347 (2016).
5. Frisco, P., Gheorghe, M., Pérez-Jiménez, M.J. (Eds.).: Applications of Membrane Computing in Systems and Synthetic Biology, in Emergence, Complexity and Computation Series, Springer (2014).
6. Pan, L., Păun, Gh.: Spiking neural P systems: an improved normal form. Theoretical Computer Science, 411(6), 906-918 (2010).
7. Păun, Gh., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press, New York (2010).
8. Zhang, G., Gheorghe, M., Pérez-Jiminez, M.J.: Real-life Applications with Membrane Computing. Springer (2017).
9. Fang, P., Li, Y., Yang, X.: A fault diagnosis method of transmission network combining Petri net with expert system. Proceedings of the Chinese Society of Universities for Electric Power System and Automation, 17(2), 26-30 (2005).

10. He, J., Xiao, J., Liu, X., Wu, T., Song, T.: A novel membrane-inspired algorithm for optimizing solid waste transportation. Optik–International Journal for Light and Electron Optics, 126(23), 3883-3888 (2015).
11. Păun, Gh., Rozenberg, G., A Guide to Membrane Computing.: Theoretical Computer Science, 287, 73-100 (2002).
12. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science, 296(2), 295-326 (2003).
13. 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(3), 1528-1542 (2013).
14. Song, B., Zhang, C., Pan, L.: Tissue-like P systems with evolutional symport/antiport rules. Information Sciences, 378, 177-193 (2017).
15. 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).
16. Pan, L., Song, B., Valencia-Cabrera, L., Mario, J.: The computational complexity of tissue P systems with evolutional symport/antiport rules. Complexity (2018).
17. Song, B., Hu, Y., Adorna, H., Xu, F.: A quick survey of tissue-like P systems. Romanian Journal of Information Science and Technology, 21(3), 310-321 (2018).
18. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae, 71(2-3), 279-308 (2006).
19. Ionescu, M., Păun, A., Păun, Gh., Mario J.: Computing with Spiking Neural P Systems: Traces and Small Universal Systems. DNA Computing, 4287, 1-16 (2006).
20. Păun, A., Păun, Gh.: Small universal spiking neural P systems. Biosystems, 90(1), 48-60 (2007).
21. Păun, Gh.: Spiking Neural P Systems with Astrocyte-Like Control. Journal of Universal Computer Science, 13(11), 1707 (2007).
22. Ishdorj, T., Leporati, A.: Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources. Natural Computing, 7(4), 519-534 (2008).
23. Zhang, G.: Membrane Calculation: Theory and Application. Beijing: science press (2015).
24. Leporati, A., Mauri, G., Zandron, C., Păun, Gh., Mario J.: Uniform solutions to SAT and Subset Sum by spiking neural P systems. Natural Computing, 8(4), 681-702 (2009).
25. Jiang, K., Chen, W., Zhang, Y., Pan, L.: On string languages generated by sequential spiking neural P systems based on the number of spikes. Natural Computing, 15(1), 87-96 (2016).
26. Jiang, K., Chen, W., Zhang, Y., Pan, L.: On string languages generated by sequential spiking neural P systems based on the number of spikes. Natural Computing, 15(1), 87-96 (2016).
27. Luo, X., Kezunovic, M.: Implementing fuzzy reasoning petri-nets for fault section estimation[J]. IEEE Transactions On Power Delivery, 23(2), 676-685 (2008).
28. 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(20), 106-116 (2013).
29. 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).
30. Sun, J., Qin, SH., Song, Y.: A fault diagnosis method of power system based on Petri net and probabilistic information. Automation of Electric Power Systems, 27(13), 10-14 (2003).
31. Sun, J., Qin, S., Song, Y.: Fault diagnosis of electric power systems based on fuzzy petri nets[J]. IEEE Transactions On Power Systems, 19(4), 2053-2059 (2004).
32. Tu, M., Wang, J., Peng, H., Shi, P.: Application of adaptive fuzzy spiking neural P systems in fault diagnosis of power systems. Chinese Journal of Electronics, 23(1), 87-92 (2014)
33. Wen, F., Han, ZH., Tian, L.: The analytical model and method of fault diagnosis of power system based on genetic algorithm. Proceedings of the Chinese Society of Universities for Electric Power System and Automation, 10(3), 1-7 (1998).

34. Wen, F., Han, ZH., Tian, L.: The analysis model and method of power system fault diagnosis based on genetic algorithm are implemented in the second part. Proceedings of the Chinese Society of Universities for Electric Power System and Automation, 10(3), 8-14 (1998).
35. Wang, J., Peng, H.: Adaptive fuzzy spiking neural P systems for fuzzy inference and learning. International Journal of Computer Mathematics, 90(4), 857-868 (2013).
36. Wang, J., Peng, H., Tu, M., & Shi, P.: A fault diagnosis method of power systems based on an improved adaptive fuzzy spiking neural P systems and PSO algorithms. Chinese Journal of Electronics, 25(2), 320-327 (2016).
37. Wu, T., Zhang, Z., Pan, L.: On languages generated by cell-like spiking neural P systems. IEEE transactions on nanobioscience, 15(5), 455-467 (2016).
38. Wang, T., Zhang, G., Pérez-Jiménez, M.J., Cheng, J.: Weighted fuzzy reasoning spiking neural P systems: Application to fault diagnosis in traction power supply systems of high-speed railways. Journal of Computational and Theoretical Nanoscience, 12(7), 1103-1114 (2015).
39. Wang, T., Zhang, G., Pérez-Jiménez, M.J.: Fuzzy membrane computing: theory and applications. International Journal of Computers Communications & Control, 10(6), 144-175 (2015).
40. Wang, T., Zhang, G., Zhao, J., He, Z., Wang, J., Pérez-Jiménez, M.J.: Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. IEEE Transactions on Power Systems, 30(3), 1182-1194 (2015).
41. He, Y., Wang, T., Huang, K., Zhang, G., Pérez-Jiménez, M.J.: Fault diagnosis of metro traction power systems using a modified fuzzy reasoning spiking neural P system. Romanian Journal of Information Science and Technology, 18(3), 256-272 (2015).
42. Huang, K., Wang, T., He, Y., Zhang, G., Pérez-Jiménez, M. J.: Temporal fuzzy reasoning spiking neural P systems with real numbers for power system fault diagnosis. Journal of Computational and Theoretical Nanoscience, 13(6), 3804-3814 (2016).
43. Huang, K., Zhang, G., Wei X., et al.: A novel approach for fault classification of power transmission lines using fuzzy reasoning spiking neural P systems. International Conference on Bio-Inspired Computing: Theories and Application 2016.
44. Xiong, G., Shi, D., Zhu, L., et al.: A new approach to fault diagnosis of power systems using fuzzy reasoning spiking neural P systems. Mathematical Problems in Engineering, 2013(1), 211-244 (2013).
45. Zhou, L.,Wang, Y.,Zhao, M.:In 2004, the state grid relay protection and safety automatic device operation. Power System Technology, 29(16), 42-48 (2005).

Fault Location of Distribution Network with Distributed Generations Using Electrical Synaptic Transmission-Based Spiking Neural P Systems

Qing Wang[1,2] Jun Wang[1,2] Tao Wang[1,2] Hong Peng[3]

[1] School of Electrical Engineering and Electronic Information, Xihua University, Chengdu 610039, China
[2] Key Laboratory of Fluid and Power Machinery, Ministry of Education, Xihua University, Chengdu 610039, China
[3] School of Computer and Software Engineering, Xihua University, Chengdu, Sichuan, China

Abstract: This paper proposed a new electrical synaptic transmission-based Spiking Neural P system (SNP system) based on SNP systems. Some new elements are added into the original definition of SNP systems, such as new synapses, bidirectional model, two types of neurons and canceling the delay of axon. Because SNP system is easy to express the logical relationship between graphics and has strong ability to process information in parallel, the bidirectional characteristics of electrical synaptic transmission is effectively combined with the electrical quantity (direction of current) for fault location of distribution network with distributed generations (DGs) in this paper. The fault location model and reasoning algorithm of the electrical synaptic transmission-based SNP system are studied with the advantages of high accuracy, less computation, simple and intuitive model and reasoning. Furthermore, the algorithm is applied reasonably in the bidirectional power flow characteristics of distribution network with DGs. Finally, this paper verifies the effectiveness, accuracy and reliability of the method through two cases which involve the single fault, multiple fault and misinformation fault.
Keywords: distribution network with DGs; fault location; Electrical Synaptic Transmission-Based Spiking Neural P System

1 Introduction

With the large-scale DGs (distributed generation) access to the distribution network, the traditional single centralized power generation mode shifts to the power generation mode which combines the centralized and distributed

power generation mode, which changes the power flow structure and operation mode of the passive distribution network, making the fault location process more complicated. The traditional protection strategy of the passive distribution network is no longer applicable. It is of great significance to study a new method of fast and accurate fault location for the reliability of power generation with DGs [1]-[2].

In order to eliminate the influence of DGs and meet the development needs of the distribution network, many new methods of fault location have been proposed in recently years. Ref. [3]-[4] proposed a method based on high frequency impedance, which is only suitable for phase-to-phase faults and ground faults in neutral grounded power systems, and the scope of application is very limited. Ref. [5]-[6] proposed a method of fault location based on single-ended impedance and a method of fault location based on double-ended impedance respectively. And these methods have advantages of less investment and simple principles, but are greatly affected by the line impedance and the feeding power of distribution power supply in the system. Ref. [7] proposed a matrix algorithm for fault location, which is easy to implement, but it only applies to a single fault of the single power distribution network. Ref. [8] proposed an improved matrix algorithm with higher accuracy but this algorithm requires actual amplitude of short-circuit current and great computational complexity. The method based on electrical quantity, such as the method of fault diagnosis using three-phase current proposed in Ref. [9] with simple principles, which is easy to understand, but it is not easy to implement. Ref. [10] proposed a Petri net-based method with simple principles and complicated calculation. The optimization algorithms used in distribution networks with DGs include: Particle Swarm Optimization Algorithm (PSO), Heuristic Algorithm [11], Heuristic Method [12], Ant Colony Optimization Algorithm [13], QPSO Algorithm [14], etc. However, these algorithms generally have the disadvantages of great computational complexity major calculation and easy to get into local optimum. Ref. [15] proposed a neural network-based approach that focuses on the improvement of neural networks and is only applicable to single-phase faults and three-phase faults. Nonetheless, most of the ANN-based diagnosis systems suffer from the black-box phenomenon since it is difficult to extract domain knowledge encoded in a trained network to explain its results intuitively. In addition, the performance of ANN-based diagnosis systems is highly restricted without the extensive confirmation of the quality of training process and the quantity of training samples.

Membrane computing (also known as P system) is a new branch of natural computing, proposed in 1998 by Gheorghe Păun [16]. The Spiking Neural P system (SNP system) is one of the three main types of P systems, which is a distributed parallel computing model with good comprehensibility [17]. The spiking neural membrane system has the characteristics of strong distributed parallel computing capability, information processing capability and image graphics capability, which is suitable for solving problems such as fault information redundancy, complicated process and fault misjudgment in the

process of fault identification and fault location. In Ref. [18]-[26], several SNP systems are proposed and the synapses mentioned in the variants are all chemical synapses; chemical synapses rely on chemical signals for the transmission of neural information, which can only be transmitted in one direction, while electrical synapses can transmit spikes quickly and directly between neurons without the delay of axon [27]-[28].

In the brain neural network, the interconnection of neurons is called synapse [29]-[30]. Based on the SNP systems, this paper combines the bidirectional characteristics of electrical synaptic transmission into SNP systems, the bidirectional characteristic of electrical synaptic transmission corresponds to the bidirectional power flow of distribution networks with DGs, and the delay of axon is reasonably canceled. In the meantime, the rules are on the electrical synapse, so that each synapse can use different rules, and a method of fault location for distribution network with DGs using electrical synaptic transmission-based spiking neural P system is proposed [31]. At the same time, for sub-areas of the distributed power distribution network ,the fault location model of the distribution network with DGs using electrical synaptic transmission-based spiking neural P system is established, and the reasoning algorithm is used to locate the fault and verify the fault current information; the method has the following characteristics: (1) it can accurately and quickly locate fault locations including single faults, multiple faults, and misinformation faults in the distribution network with DGs; (2) it can verify the fault current information while fault location, and improve the accuracy of fault current information and fault location results; (3) its principles and system expressions are intuitive, easy to understand, strong in visibility and high in versatility; (4) and the method has strong traceability, high support of biological theory and rich theoretical content.

This paper is organized as follows. Section 2 proposes the electrical synaptic transmission-based spiking neural P system. Section 3 introduces the method of fault location of distribution network with DGs using electrical synaptic transmission-based spiking neural P system. Three cases will be studied in Section 4, which the single fault, multiple faults and misinformation faults in each case will be verified. Finally, conclusions are discussed in Section 5.

2 Electrical Synaptic Transmission-Based Spiking Neural P Systems

2.1 SNP System

Here, we briefly review the basic concepts of SNP systems in the computing form.

A SNP system of degree $m \geq 1$, is a construct of the form [17]

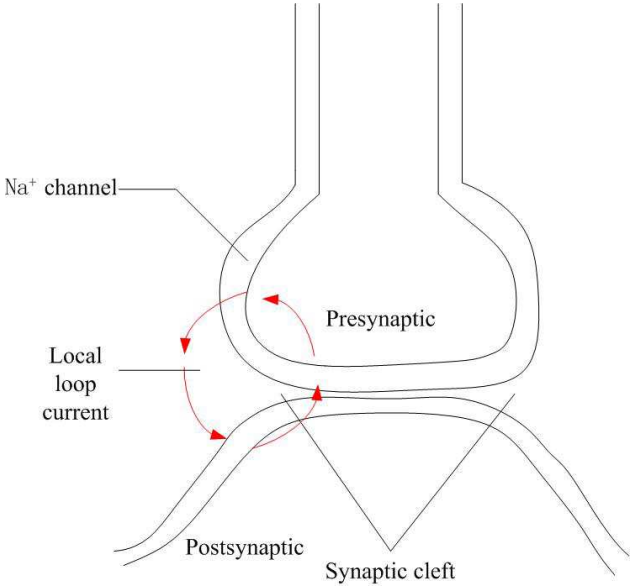


Fig. 1. Electrical synapse diagram

$$\Pi = (A, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, I, O)$$

where

- $A = \{a\}$ is the singleton alphabet (the object a is called spike).
- $\sigma_1, \sigma_2, \cdots, \sigma_m$ are neurons, of the form $\sigma_i = (n_i, R)$ with $1 \leq i \leq m$, where

- $n_i \geq 0$ is the initial number of spikes contained by neurons σ_i;
- R is a finite set of rules of the form $E/a^c \rightarrow a^p; d$

where E is a regular expression over $a, c \geq 1$ and $p, d \geq 0$, if $p = 0$, then E is the empty string, $d = 0$, and a^s belongs to the language generated by expression E for no rule $E/a^c \rightarrow a^p; d$;

- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for all $(i, j) \in syn$,$1 \leq i \leq m$;
- I and O are input neuron set and output neuron set, respectively.

2.2 Electrical Synaptic Transmission-Based Spiking Neural P Systems

2.2.1 Synapse

Synapses in the brain neural network system refer to the interconnected structure of spikes from one neuron to another. According to its structure and

function, it can be divided into two major categories: electrical synapse and chemical synapse.

Electrical synapse: presynaptic neurons transmit neural information by means of electrical signals;

Chemical synapse: presynaptic neurons transmit neural information through the synaptic cleft and eventually to postsynaptic neurons by means of chemical signals.

Electrical synapse refers to the site where two neuronal membranes are in close contact, and the site has a protein-forming channel, and the membrane impedance of the site is low. The information transmission on the electrical synapse is an electrical (electrical signal) transmission, and in addition to the fast transmission rate of the nerve impulse, that is, no delay of axon, since the ion flow of the connecting channel is bidirectional, the signal transmission is bidirectional[10].

2.3 Electrical Synaptic Transmission-Based Spiking Neural P Systems

According to the introduction of the definition and characteristics of electrical synapses in Section 2.1.1, we will propose an electrical synaptic transmission-based spiking neural P system.

A electrical synaptic transmission-based spiking neural P system of degree $m \geq 1$, is a construct of the form

$$\Pi = (A, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, I/O, O/I)$$

where

- $A = \{a\}$ is the singleton alphabet (the object a is called spike).
- $\sigma_1, \sigma_2, \cdots, \sigma_m$ are neurons in system $\prod$, of the form $\sigma_i = (\alpha_i)\,(1 \leq i \leq m)$ with $1 \leq i \leq m$; where $\alpha_i \in (-1, 0, 1)(1 \leq i \leq m)$ represents the spike value;
- *syn* represents a collection of electrical synapses between neurons in the system, of the form $((i, j), R(i, j))$, where

(1) $(i, j)(1 \leq i, j \leq m)$ represents electrical synapses $i \neq j$ that connect neurons σ_i and σ_j.

(2) $R(i, j)$ is a finite set of rules of the form on electrical synapses:

A. $E/a^{\alpha} \rightarrow a^{\beta}$ the rule is a firing rule; $E/a^{\beta} \rightarrow a^{\alpha}$ the rule is a backward firing rule;

B. If $\beta = 0$, the rule is written in the form $E/a^{\alpha} \rightarrow \lambda$, that is forgetting rule. If $\alpha = 0$, the rule is written in the form $E/a^{\beta} \rightarrow \lambda$, that is backward forgetting rule. where E is regular expression over; where $\alpha \in (-1, 0, 1)$, $\beta \in (-1, 0, 1)$.

- syn $\subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for all $(i, j) \in$ syn, $1 \leq i \leq m$ is a directed graph between electrical synapses;

- I/O and O/I are input/output neuron set and output/input neuron set, respectively.

Wherein, in the electrical synaptic transmission-based spiking neural P system, the spike value contained in the neuron is no longer the number of spikes, but a discrete value. According to the characteristics of electrical synaptic transmission described in 2.1.1, which is bidirectional and has no delay of axon, the delay of axon is canceled in the electrical synaptic transmission-based spiking neural P system. At the same time, new firing rules corresponding to the bidirectionality of electrical synaptic transmission are added to the electrical synaptic transmission-based spiking neural P system. Backward firing rules and backward forgetting rules, corresponding to the backward reasoning process of the algorithm and model below. And the rules in the system are located on the synapse rather than in the neurons, i.e. the neurons only contain spikes to ensure that different rules can be used for each synapse during the forward and backward reasoning process of the model.

Secondly, there are two types of neurons in the electrical synaptic transmission-based spiking neural P system: input/output neurons, output/input neurons and section neurons.

3 Method of Fault Location Using Electrical Synaptic Transmission-Based Spiking Neural P Systems

3.1 Fault Diagnosis Process

In the distribution network with DGs, it is divided into areas by using the normally interconnection switch as the dividing point. In the fault location process, only the area containing the fault information is selected, and the fault current information reported by the FTU configured at each switch is

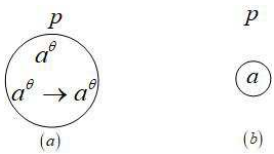


Fig. 2. (a) Input/output neuron, output/input neuron and (b) its simplified form

Fig. 3. (a) section neuron and (b) its simplified form

used for calculation. And in a distribution network with DGs, a sub-net surrounded by switches or terminal points, which no longer includes a switch, is referred to as a section, and the section is the smallest unit of the method of fault location herein. Secondly, establishing a corresponding electrical synaptic transmission-based spiking neural P system for the candidate fault areas. Thirdly, running a forward reasoning algorithm and a backward verification algorithm on the established model. Fourthly, realizing accurate fault location of the fault section and verification of the fault current information through the fault judgment standard; finally, obtaining the final results.

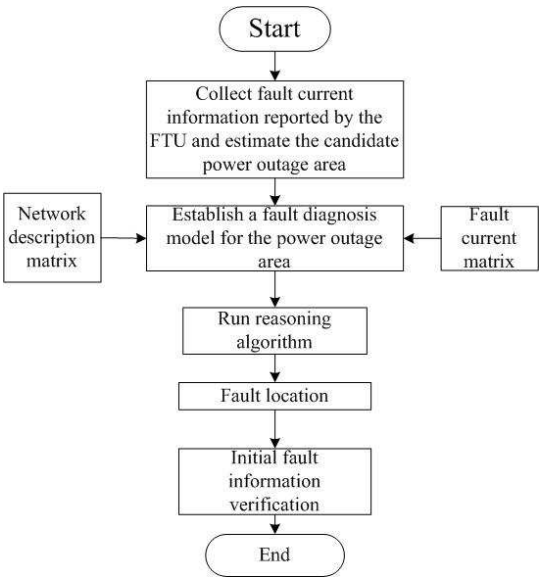


Fig. 4. Flow chart of fault location of distribution network with DGs

3.2 Model Establishing Principle

In section 2.2, we have learned that the electrical synaptic transmission-based spiking neural P system is a bidirectional system that can perform bidirectional reasoning. In this section, the forward reasoning process and the backward verification process of the system are introduced in two subsections.

3.2.1 Forward Reasoning Process

In the distribution network with DGs, only the power outage section is selected to establish a fault location model. Firstly, the direction in which the main power source in the distribution network with DGs is directed to the feeder

line or DG is defined as the positive direction. Then, according to the fault current information detected by the FTU, different working modes are set, and the modes are divided into -1, 0, and 1. Where mode -1 indicates that the fault current locates at node j and the current direction is opposite to the positive direction; mode 1 indicates that the fault current locates at node j and the current direction is the same as the positive direction, as shown in Table 1.

Table 1. Working Mode of Fault Current Information

$(I/O)_j$	The fault current is present at node j and the current direction is opposite to the assumed positive direction	-1
	The fault current is present at node j and the current direction is consistent with the assumed positive direction	1
	No fault current at the node	0

Next, In the distribution network, the network description matrix D is constructed according to the topology of the distribution network with DGs. Node j is the exit point of section i, and the element d_{ij} in D is -1; node j is the enter point of section i, and the element d_{ij} in D is 1. If node j is not associated with section i, it is 0, as shown in Table 2.

Table 2. Element d_{ij} in Network Description Matrix D

d_{ij}	Node j is the exit point of section i	-1
	Node j is the enter point of section i	1
	Node j is not associated with section i	0

Then, get the result by $O/I = D \cdot (I/O)^T$, if $(O/I)_i \leq 0$, no fault occurs in section i; if $(O/I)_i > 0$, fault occurs in section i. To intuitively and easily express the working mode of fault state outputs, and the jump function $H(x) = \begin{cases} 1, x > 0 \\ 0, x \leq 0 \end{cases}$ is applied to obtain the result as shown in Table 3.

3.2.2 Backward Verification Process

Firstly, according to the actual current flow direction on the feeder line of the power outage section, the current direction matrix S is constructed, and the elements S_{ij} in matrix S, as shown in Table 4.

Table 3. Working Mode of Fault State Outputs

$(O/I)_i$	No fault occurs in section i	0
	Fault occurs in section i	1

Table 4. Actual Current Direction Matrix S

S_{ij}	The fault current flows into section j via node i and is opposite to the assumed positive direction	-1
	The fault current flows into section j via node i and is consistent with the assumed positive direction	1
	Node i is not associated with section j	0

According to the final result obtained in 3.2.1, the final backward verification process result is expressed intuitively and simply by $(I/O)^1 = S^T \cdot (O/I)$, and the sgn function in the formula (1) is applied to the obtained result, that is,

$$\begin{cases} x > 0, \operatorname{sgn} x = 1 \\ x = 0, \operatorname{sgn} x = 0 \\ x < 0, \operatorname{sgn} x = -1 \end{cases} \tag{1}$$

Table 5. Working Mode of Fault State Outputs

$(I/O)^1 \wedge (I/O)$	1	If the vector elements in G and I/O are consistent, the input fault information is complete and correct.
	0	If one of vector elements in G and I/O is different, the input fault information is incomplete or distorted.

According to $(I/O)^1 \wedge (I/O)$, the working mode of fault authenticity verification shown in Table 5 is obtained.

Note: $(I/O)^1$ is the input/output neuron I/O, because the results contained in the neuron are different during the forward reasoning process and the backward verification process, the two forms are used to indicate the distinction, but they are actually the same neuron.

3.3 Reasoning Algorithm

This section will describe the reasoning algorithm of the electrical synaptic transmission-based spiking neural P system, which is divided into two parts.

Firstly, forward reasoning part; if discrete values of input/output I/O neurons are known, discrete values of other unknown neurons can be derived from the forward reasoning part of the algorithm. The reasoning result is a sequence of spikes formed by discrete values of the output/input neuron O/I.

Secondly, backward reasoning part; in forward reasoning part, the sequence of spikes formed by the discrete values of the output/input neuron O/I derived from the forward reasoning part can be inversely derived from the discrete values of the input/output neuron I/O.

In the reasoning process, there are $p = m + n + 1$ neurons, including m input/output neurons I/O, n section neurons σ_j, and 1 input/output neuron O/I. To illustrate this reasoning algorithm better and more conveniently, we now introduce some matrices as follows:

(1) $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)^T$ is a vector containing discrete values in m input/output neurons I/O, where $\alpha = (-1, 0, 1)$ represents discrete values $(1 \leq i \leq m)$ in the ith input/output neuron I/O.

(2) $\beta = (\beta, \beta_2, \ldots, \beta_n)$ is a vector containing discrete values in section neurons σ_j, where $\beta = (-1, 0, 1)$ represents discrete values $(1 \leq j \leq n)$ in the jth section neuron σ_j.

(3) $O/I = (\beta_1, \beta_2, \ldots, \beta_n)^T$ is an input/output neuron, and the output result is a sequential spike train of the output result in section neurons;

(4) $D = (d_{ij})_{m \times n}$ is the network description matrix between input/output neurons I/O and section neurons σ_j in the forward reasoning part, constructed according to the description of Table 2;

(5) $S = (s_{ji})_{n \times m}$ is the current direction matrix between section neurons σ_j and output/input neurons O/I in the backward verification part, constructed according to the description of Table 4;

Forward reasoning part: firstly, the initial values of the input/output neurons I/O and the section neurons σ_j are set separately. Next, according to $\beta = D \times \alpha$, the discrete values of the section neurons are calculated; then, the discrete values of the section neurons will appear in the output/input neurons O/I in the form of spike trains. If $\beta = 1$, fault occurs in section i . Conversely, if $\beta = 0$, no fault occurs in section i.

Backward verification part: according to the results obtained in the output/input neurons O/I and $\alpha = S \times \beta$, get discrete values in the input/output neurons $(I/O)^1$.

Finally, combining the results obtained from the input/output neurons of the backward verification algorithm with the fault current information in the input/output neurons of the forward reasoning algorithm, and verifying whether the fault current information is wrong.

Note: $(I/O)^1$ is the input/output neuron I/O, because the results contained in the neuron are different during the forward reasoning process and the backward verification process, the two forms are used to indicate the distinction, but they are actually the same neuron.

4 Case Study

4.1 Fault Location of Simple Distribution Network with DGs

In order to facilitate the understanding of the electrical synaptic transmission-based SNP model, this section will illustrate several examples of distribution networks with DGs. According to the fault location principle of distribution network with DGs and the electrical synaptic transmission-based SNP, three fault conditions of each example are analyzed: single fault, multiple faults, and misinformation fault.

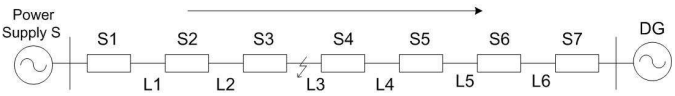


Fig. 5. A simple feeder network

A simple power distribution system is shown in Figure 5, where S1-S7 are section switches and L1-L6 are feeder sections. Fig. 6 is the electrical synaptic transmission-based SNP model.

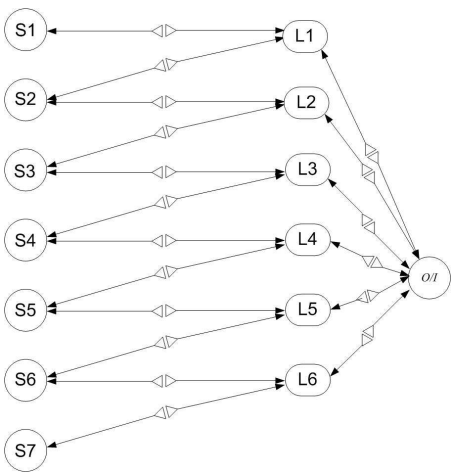


Fig. 6. Electrical synaptic transmission-based SNP model

The section switches and the like on the feeder line are numbered, that is, S1 to S7, and are represented by input/output neurons $\sigma_i(1 \leq i \leq 7)$; and sections of the feeder line are numbered, that is, L1 to L6, and are represented by section neurons $\sigma_j(1 \leq j \leq 6)$. The electrical synaptic transmission-based SNP model of Fig. 6 is established according to the actual relationship between section switches and sections. The bidirectional connection between the

input/output neurons and the section neurons, the section neurons and the output/input neurons corresponds to the bidirectional transmission characteristics of the electrical synaptic transmission; it also conforms to the bidirectional current flow characteristics in distribution networks with DGs; at the same time, in Biology, the neurons transmit information in both directions through electrical synapses, and the information processing is performed in both directions.

4.1.1 Single fault: when section L3 fail

A. Forward reasoning process

The fault information reported by the FTU obtains the flow direction of the fault current experienced on each section switch, and the fault current column vector is constructed as

$$I/O = [1, 1, 1, -1, -1, -1, -1]^T$$

According to the principle of fault diagnosis, a network description matrix D is established as

$$D = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

According to reasoning algorithm $B = D \times \alpha$, we can obtain as

$$O/I = [0, 0, 1, 0, 0, 0]^T$$

That is, the section L3 fails.

B. Backward verification process

In this reasoning model, we can obtain as

$$O/I = [0, 0, 1, 0, 0, 0]^T$$

Establish the actual current direction matrix as

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

According to $\alpha = S^T \times \beta$, $(I/O)^1 = [1, 1, 1, -1, -1, -1, -1] = I/O$, this result proves that the forward reasoning result is correct and the fault current information is correct. That is, it is proved by the forward reasoning process and the backward verification process that the fault has occurred at the section L3. And the fault current information has no misinformation.

4.1.2 Multiple faults: when section L3 and L5 fails

A. Forward reasoning process

$$I/O = [1, 1, 1, 0, 0, -1, -1]^T$$

Since the topology of the distribution network with DGs has not changed, the network description matrix D remains unchanged as

$$D = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

According to reasoning algorithm $\beta = D \times \alpha$, we can obtain as

$$O/I = [0, 0, 1, 0, 1, 0]^T$$

That is, the section L3and L5 fail.

B. Backward verification process

In this reasoning model, $O/I = [0, 0, 1, 0, 1, 0]^T$.

Establish the actual current direction matrix as

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

According to $\alpha = S^T \times \beta$, $(I/O)^1 = [1, 1, 1, 0, 0, -1, -1] = I/O$, this result proves that the forward reasoning result is correct and the fault current information is correct. That is, it is proved by the forward reasoning process and the backward verification process that the fault has occurred at the section L3 and L5. And the fault current information has no misinformation.

4.1.3 Misinformation fault: when section L3 fail

A. Forward reasoning process

When the fault has occurred at the section L3, the fault information reported by the FTU obtains the flow direction of the fault current experienced on each section switch, if the FTU incorrectly reports the current direction information on the section switch S7, then the fault current column vector is constructed as follows

$$I/O = [1, 1, 1, -1, -1, -1, 1]^T$$

According to the principle of fault diagnosis, a network description matrix D is established as follows

$$D = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

According to reasoning algorithm $\beta = D \times \alpha$, we can obtain as

$$O/I = [0, 0, 1, 0, 0, 0]^T$$

That is, the section L3 fails.

B. Backward verification process

In this reasoning model, $O/I = [0, 0, 1, 0, 0, 0]^T$
Establish the actual current direction matrix as

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

According to $\alpha = S^T \times \beta$, $I/O \neq [1, 1, 1, -1, -1, -1, -1] = I/O$, this result proves that the forward reasoning result is correct and the fault current information is correct. That is, it is proved by the forward reasoning process and the backward verification process that the fault has occurred at the section L3. And the fault current information has misinformation.

4.2 Multi-Power Distribution Network

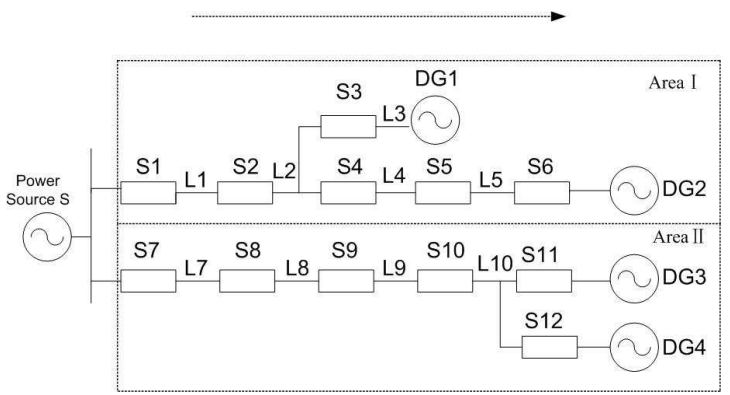


Fig. 7. Multi-power distribution network

As shown in Fig. 7, the distribution network for high reliability requirements is in closed loop mode, and the actual operation mode is area operation. Then the classic distribution network with DGs in Fig. 7 is divided into two areas: Area I and Area II.

4.2.1 Single fault: when section L3 fail

The corresponding electrical synaptic transmission-based SNP model is shown in Fig. 8.

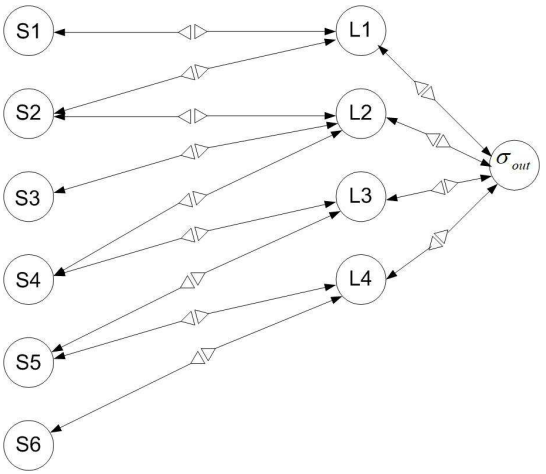


Fig. 8. Electrical synaptic transmission-based SNP model for multi-power distribution nerwork

A. Forward reasoning process

The fault information reported by the FTU obtains the flow direction of the fault current experienced on each section switch, and the fault current column vector is constructed as

$$I/O = [1, 1, 0, 1, -1, -1]^T$$

According to the principle of fault diagnosis, a network description matrix D is established as

$$D = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

According to reasoning algorithm $\beta = D \times \alpha$

$$O/I = [0, 0, 1, 0]^T$$

That is, the section L3 fails.

B. Backward verification process

In this reasoning model $O/I = [0, 0, 1, 0]^T$.

Establish the actual current direction matrix as

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

According to $\alpha = S^T \times \beta$, $I/O = [1, 1, 0, 1, -1, -1] = I/O$, this result proves that the forward reasoning result is correct and the fault current information is correct. That is, it is proved by the forward reasoning process and the backward verification process that the fault has occurred at the section L3. And the fault current information has no misinformation.

4.2.2 Multiple faults: when section L3 and L4 fails

A. Forward reasoning process

$$I/O = [1, 1, 0, 1, 0, -1]^T$$

Since the topology of the distribution network with DGs has not changed, the network description matrix remains unchanged as

$$D = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

According to reasoning algorithm $\beta = D \times \alpha$

$$O/I = [0, 0, 1, 1]^T$$

That is, the section L3 and L4 fail.

B. Backward verification process

In this reasoning model $O/I = [0, 0, 1, 1]^T$. Establish the actual current direction matrix as

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

According to $\alpha = S^T \times \beta$, $I/O = [1, 1, 0, 1, 0, -1] = I/O$, this result proves that the forward reasoning result is correct and the fault current information is correct. That is, it is proved by the forward reasoning process and the backward verification process that the fault has occurred at the section L3 and L4. And the fault current information has no misinformation.

4.2.3 Misinformation fault: when section L3 fail

A. Forward reasoning process

$$I/O = [1, 1, 0, 1, -1, 1]^T$$

According to the principle of fault diagnosis, a network description matrix is established as

$$D = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

According to reasoning algorithm $\beta = D \times \alpha$.

$$O/I = [0, 0, 1, 0]^T$$

That is, the section L3 fails.

B. Backward verification process

In this reasoning model $O/I = [0, 0, 1, 0]^T$.

Establish the actual current direction matrix as

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

According to $\alpha = S^T \times \beta$, $I/O \neq [1,1,0,1,-1,-1] = I/O$, this result proves that the forward reasoning result is correct and the fault current information is correct. That is, it is proved by the forward reasoning process and the backward verification process that the fault has occurred at the section L3. And the fault current information has misinformation.

4.3 Complex Distribution Network

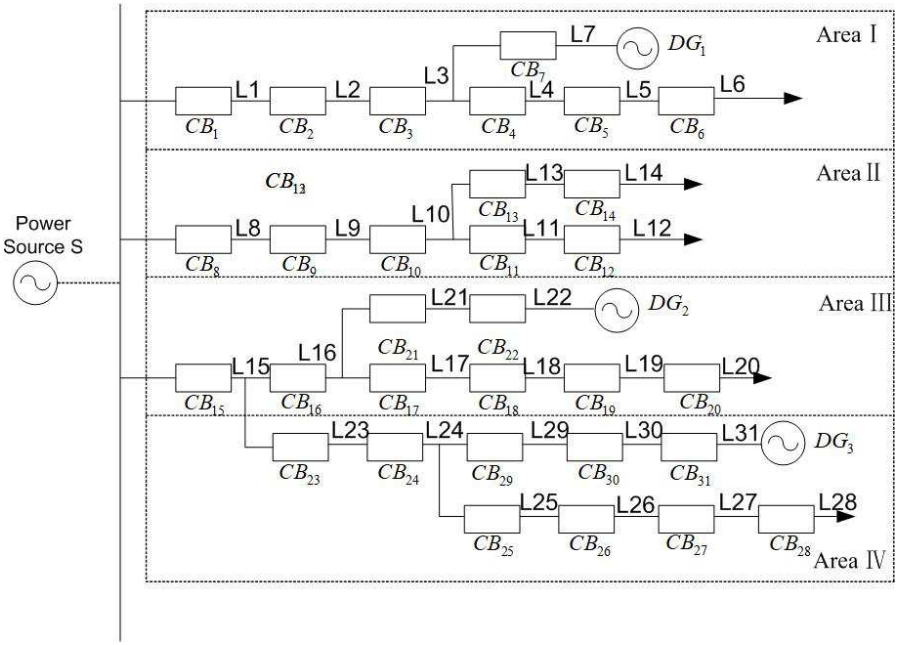


Fig. 9. Complex distribution network with DGs

Complex distribution network with DGs is shown in Fig. 9. In the actual complex distribution network with DGs, fault location can be performed according to the fault location scheme of the multi-power distribution network: the distribution network with DGs will be divided into areas, and then fault location will be performed as described above. The reasoning is not repeated here.

4.4 Analysis and Comparison

A method of fault location of distribution network is proposed with DGs using electrical synaptic transmission-based spiking neural P system in this paper. Case 1 has also been studied in Ref. [7], but only the single fault condition of

the distribution network for feeder line is discussed in it, and the scope of application is narrow. Ref. [8] improved the matrix algorithm based on Ref. [7], but the matrix algorithm used in [8] requires a large amount of current amplitude data, and the calculation is large and the fault tolerance is poor. The single-ended impedance and double-ended impedance-based methods used in Ref. [5]-[6] are greatly affected by the line impedance and the feeding power of distribution power supply in the system, and the fault location cannot be performed quickly and accurately. The particle swarm optimization algorithm used in ref.[11]-[14] needs to continuously update the position and velocity of each particle in the operation process to obtain the optimal inertia weight, and it is easy to fall into local optimum in the operation process, and Ref. [13] mentioned that if multiple faults occur on feeders that are not connected by DG, only one fault location can be located, and there is a problem of missing fault locations. The neural network-based fault location method used in Ref. [15] has a positioning error. In the meantime, the algorithms involved in [20]-[21], [32]-[33] require historical data from circuit breakers CB, relays R, etc., and do not involve electrical quantity related information, and this method has strong dependence on historical data and empirical data, which leads to poor fault tolerance. The method presented in this paper is not only applicable to a variety of fault conditions for distribution network with DGs, but also can verify the fault current information, improve the reliability of the fault current information and the accuracy of the fault location result while in fault location.

5 Conclusion

Based on the traditional SNP system, this paper proposes a method of fault location of distribution network with DGs using electrical synaptic transmission-based spiking neural P system. This method adds a new element, the electrical synapse, and cancels the delay of axon; the electrical quantity and network topology are utilized in the reasoning algorithm to effectively combine the bidirectionality and rapidity of the electrical synapse. The method presents the effectively combined results in a graphical model according to the model establishing principle, visually represents the relationship between the various parts of the distribution network and the logical relationship of the graphics, and visually represents the fault location process.

In this paper, the three cases from simple to complex distribution networks with DGs are verified separately, so that readers can more easily understand the method of fault location and fault location process proposed in this paper. Through the verification of these three cases, it is proved that the method of fault location is not affected by the complexity of the distribution network.

References

1. Sun Jingliao, Chen Rongzhu, Cai Shi, Li Qi, Zhao Pu, Dong Danhuang. A New Fault Location Scheme for Distribution System with Distributed Generations[J]. Power System Technology, 37(6), 2013: 1645–1650.
2. Atwa M, El-Saadany E F, Salama M M A, et al. Adequacy evaluation of distribution system including wind/solar dg during different modes of operation[J]. IEEE Transactions on Power Systems, 26(4), 2011: 1945–1952.
3. Ke Jia, Tianshu Bi, Zhefeng Ren, David W. P. Thomas, and Mark Sumner. High Frequency Impedance Based Fault Location in Distribution System with DGs[J]. IEEE Transactions on Smart Grid, 9(2), 2018: 807–816.
4. W. C. Santos, F. V. Lopes, N. S. D. Brito, and B. A. Souza. High-Impedance Fault Identification on Distribution Networks[J]. IEEE Transactions on Power Delivery, 32(1), 2017: 23–32.
5. Ke Jia, David Thomas, and Mark Sumner. A New Single-Ended Fault-Location Scheme for Utilization in an Integrated Power System[J]. IEEE Transactions on Power Delivery, 28(1), 2013: 38–46.
6. Ke Jia, David Thomas, and Mark Sumner. A New Double-Ended Fault-Location Scheme for Utilization in Integrated Power Systems[J]. IEEE Transactions on Power Delivery, 28(2), 2013: 594–603.
7. Liu Jian, Ni Jianli, Du Yu. A unified matrix algorithm for fault section detection and isolation in distribution system[J]. Automation of Electric Power System, 23(1), 1999: 31–33.
8. Li Kaiwen, Yuan Rongxiang, Deng Xiangtian, Li Timing, Improved matrix algorithm for fault location in ring distribution system with distributed generations[J]. Proceedings of the CSU-EPSA, 12 (26), 2014: 63–68.
9. Mahfouz, MMA, El-Sayed, MAH. Smart Grid Fault Detection and Classification with Multi-Distributed Generation Baes on Current Signals Approach[J]. IET Generation Transmission and Distribution, 10(16), 2016: 4040–4047.
10. V. Calderaro, A. Piccolo, V. Galdi, P. Siano. Identifying Fault Location in Distribution Systems with High Distributed Generation Penetration, IEEE Africon, 2009: 1–6.
11. Xu Pengfei, Xing Renzhou. Fault location of distribution network with distributed power supply by particle swarm optimization algorithm[J]. Chinese Journal of Power Sources, 4(42), 2018: 591–600.
12. Giovanni Manassero, Silvio Giuseppe Di Santo and Laiz Souto. Heuristic Method for Fault Location in Distribution Feeders with the Presence of Distributed Generation. IEEE Transactions on Smart Grid, 8(6), 2017, 2849–2957.
13. Tao, W.Q., Yang, G., Zhang, J.Y.: Fault section locating for distribution network with DG based on improved ant colony algorithm. In: 8th International Power Electronics and Motion Control Conference, China, 2016: 1–4.
14. ZB Shi, Y Li, YF Song, T Yu. Fault Diagnosis of Transformer Based on Quantum-Behaved Particle Swarm Optimization-Based Least Squares Support Vector Machines. In 2009 International Conference on Information Engineering and Computer Science, 2009.
15. Rezaei, N., Haghifam, M.R. Protection scheme for a distribution system with distributed generation using neural networks[J]. International Journal of Electrical Power and Energy Systems, 30(4), 2008: 235–241.
16. Go. Pun, Computing with membranes[J]. Journal of Computer and System Sciences, 61 (1) , 2000: 108–143.

17. M. Ionescu, G. Paun, and T. Yokomori, Spiking neural P systems [M]. IOS Press, 7(2), 2006: 279–308.
18. Hong Peng, Jun Wang, Jun Ming, Peng Shi, Mario J. Pérez-Jiménez, Wenping Yu, Chengyu Tao. Fault diagnosis of power systems using intuitionistic fuzzy spiking neural P systems[J]. IEEE Transaction on Smart Grid, 9(5), 2018: 4777–4784.
19. Hong Peng, Jun Wang, Peng Shi, Mario J. Prez-Jimnez, Agustn Riscos-Núñez. Fault diagnosis of power systems using fuzzy tissue-like P systems[J]. Integrated Computer-Aided Engineering, 24(3), 2017: 1–11.
20. Hong Peng, Jun Wang, Mario J. Pérez-Jiménez, Hao Wang, Jie Shao, Tao Wang. Fuzzy reasoning spiking neural P system for fault diagnosis[J]. Information Science, 235(6), 2013: 106–116.
21. Tao Wang, Gexiang Zhang, Junbo Zhao, Zhengyou He, Jun Wang, and Mario J. Prez-Jimnez. Fault Diagnosis of Electric Power Systems Based on Fuzzy Reasoning Spiking Neural P Systems[J]. IEEE Transaction on Power Systems, 30(3), 2015: 1182–1194.
22. T. Wu, A. Paun, Z. Zhang, L. Pan. Spiking neural P systems with polarizations[J]. IEEE Transactions on Neural Networks and Learning Systems, 29(8),2018: 3349–3360.
23. T. Wu, F.-D. Bilbie, A. Paun, L. Pan, F. Neri. Simplified and yet Turing universal spiking neural P systems with communication on request[J]. International Journal of Neural Systems, 28(8),2018: 1850013.
24. L. Pan, G. Pun, G. Zhang G, F. Neri. Spiking neural P systems with communication on request. International journal of neural systems[J], 27(08),2017: 1750042.
25. L. Pan, T. Wu, Y. Su, A. V. Vasilakos. Cell-Like spiking neural P systems with request rules[J]. IEEE Transactions on Nanobioscience, 16(6),2017: 513–522.
26. T. Wu, Z. Zhang, G. Paun, L. Pan: Cell-like spiking neural P systems, Theoretical Computer Science, 623,2016: 180–189.
27. Rong Wang, Jiajia Li, Mengmeng Du, Jinzhi Le, Ying Wu. Transition of spatiotemporal patterns in neuronal networks with chemical synapses[J]. Commun Nonlinear Sci Number Simlat., 40, 2016: 80–88.
28. Lin Lingpeng. Modeling research and verification of dynamic STDP synaptic system [D]. University of Electronic Science and Technology of China. 2012.
29. Omar Ouachikh,Aziz Hafidi, Yves Boucher and Wisam Dieb. Electrical Synapses are Involved in Orofacial Neuropathic Pain[J]. Neuroscience, 382, 2018: 69–79.
30. Jun Wang, Peng Shi, Hong Peng, Mario J. Péerez-Jiméenez, and Tao Wang. Weighted Fuzzy Spiking Neural P Systems[J]. IEEE Transactions on Fuzzy Systems, 21(2), 2013: 209–220.
31. Audrey J. Marsh, Jennifer Carlisle Michel, Anisha P. Adke, Emily L. Heckman, Adam C. Miller. Asymmetry of an Intracellular Scaffold at Vertebrate Electrical Synapses[J]. Current Biology, 27(22), 2017: 3561–2567.
32. Song-Won Min, Jin-Man Sohn, Jong-Keun Park, and Kwang-Ho Kim. Adaptive Fault Section Estimation Using Matrix Representation with Fuzzy Relations[J]. IEEE Transactions on Power Systems, 19(2), 2004: 842–848.
33. Wen-Hui Chen. Fault Section Estimation Using Fuzzy Matrix-Based Reasoning Methods[J]. IEEE Transactions on Power Delivery, 26(1), 2011: 205–213.

Insertion Based Picture Array P Systems

G. Samdanielthompson[1], Atulya K. Nagar[2], N. Gnanamalar David[**,2], Gexiang Zhang[3], and K.G. Subramanian[**,2]

[1] Department of Mathematics, Hindustan College of Arts and Science, Rajiv Gandhi Salai (OMR), Padur, Kelambakkam, Chennai 603103 India
samdanielthompson@gmail.com
[2] Department of Mathematics and Computer Science, Faculty of Science, Liverpool Hope University, Liverpool L16 9JD UK
[**] Honorary Visiting Professor
nagara@hope.ac.uk, kgsmani1948@gmail.com, ngdmcc@gmail.com
[3] School of Electrical Engineering, Southwest Jiaotong University, Chengdu 610031, China
gexiangzhang@gmail.com

Abstract. A new picture array generating model of array P system with a restricted type of picture insertion rules and picture array objects in its regions, is introduced. The generative power of such a system is exhibited by comparing with certain related picture array generating models. As an application, certain floor-design patterns, called "kolams", are generated using such an array P system.

Keywords: Array P system Insertion Two-dimensional picture array

1 Introduction

In the area of membrane computing [10, 11], the novel computing model of P system in its basic form and in its several variants, has served as a framework for dealing with problems in different areas of application [20]. One such area is in formal language theory with different models of P systems having been developed for handling the problem of generation of classes of languages, starting with the seminal work of Păun [9]. In the extension of language theory to two dimensions, P systems have also played a significant role with different kinds of P systems with array objects and array evolving rules having been introduced (see, for example, [10, 16, 19]).

The operation of insertion on words has been studied in string language theory in the context of DNA computing [8]. Fujioka [2] considered this operation in two-dimensional picture arrays and introduced a picture array generating model. This model has a feature which is analogous to the pure 2D context-free grammar [3] where the operation is rewriting in parallel all symbols in a column or row of a (rectangular) picture array by strings of equal length while in [2], insertion (instead of rewriting) of strings of equal length is done in parallel

between columns or rows. Here we consider a restricted type of insertion rules with the "contexts" of length at most one, in the regions of a cell-like P system with the objects being picture arrays in the regions. We examine the generative power of the resulting array P system model introduced here with certain other picture array generating models. We also exhibit an application to generation of "kolam" patterns [14].

2 Preliminaries

For formal language theory related notions, the reader can refer to [12] and to [3, 17] for two-dimensional array grammars and languages. For P systems and array P systems we refer to [9, 16]

A finite set V of symbols is called an alphabet. A word or a string $\alpha = a_1 a_2 \dots a_m$ $a_i \in V, 1 \leq i \leq m, (m \geq 1)$ of length m over an alphabet V is a finite sequence of symbols belonging to V. The length of a word α is denoted by $|\alpha|$. The set of all words over V, including the empty word λ with no symbols, is denoted by V^*. For any word $w = a_1 a_2 \dots a_n$, ${}^t w$ is the vertical word with the word w written vertically. For example, if $\alpha = bab$ over the alphabet $\{a, b\}$, then ${}^t\alpha$ is $\begin{matrix} b \\ a \\ b \end{matrix}$. If w is a single symbol of the alphabet, then we write ${}^t w$ as w itself. A rectangular $p \times q$ array (also called picture array) X over an alphabet V is of the form

$$X = \begin{matrix} a_{11} & \cdots & a_{1q} \\ \vdots & \ddots & \vdots \\ a_{p1} & \cdots & a_{pq} \end{matrix}$$

where each $a_{ij} \in V, 1 \leq i \leq p$, $1 \leq j \leq q$. The number of rows of X and the number of columns of X are respectively denoted by $|X|_r$ and $|X|_c$. The set of all rectangular arrays over V is denoted by V^{**}, which includes the empty array λ. $V^{++} = V^{**} - \{\lambda\}$. A picture language is a subset of V^{**}.

The column catenation $X \circ Y$ of two arrays X and Y with the same number of rows is formed by juxtaposing Y to the right of X. The row catenation $X \diamond Y$ of two arrays X and Y with the same number of columns is formed by juxtaposing Y below X.

3 Array P System with Restricted Picture Insertion Rules

A picture-insertion system has been considered in [2] with insertion in a picture array being done between columns or rows, extending the notion of insertion in words [8]. We recall the definition of this system with a special case, namely, with "contexts" of length one in the rules and call this system as a restricted picture-insertion system.

Definition 1. *A restricted picture-insertion system (RPIS) is a 4-tuple* $\Gamma = (\Sigma, I_c, I_r, A)$ *where*

i) Σ *is an alphabet;*

ii) $I_c = \{c_i \mid 1 \leq i \leq m\}$, $(m \geq 1)$ *where* c_i, *called a column insertion table, is a finite set of column insertion rules with alphabetic context of the form* (a, α, b), $a, b \in \Sigma \cup \{\lambda\}, \alpha \in \Sigma^*$ *such that for any two rules* $(a_1, \alpha, b_1), (a_2, \beta, b_2)$ *in* c_i, *we have* $|\alpha| = |\beta|$ *and either both the left contexts* a_1 *and* a_2 *are in* Σ *(likewise the right contexts* b_1 *and* b_2 *are in* Σ*) or both are* λ *;*

iii) $I_r = \{r_j \mid 1 \leq j \leq n\}$, $(n \geq 1)$ *where* r_j, *called a row insertion table, is a finite set of row insertion rules with alphabetic context of the form* $(d, {}^t\gamma, e)$, $d, e \in \Sigma \cup \{\lambda\}, \gamma \in \Sigma^*$ *such that for any two rules* $(d_1, {}^t\gamma, e_1), (d_2, {}^t\delta, e_2)$ *in* r_j, *we have* $|\gamma| = |\delta|$ *and either both the up contexts* d_1 *and* d_2 *are in* Σ *(likewise the down contexts* e_1 *and* e_2 *are in* Σ*) or both are* λ*;*

iv) $A \subseteq \Sigma^{**} - \{\lambda\}$ *is a finite set of axiom arrays.*

For picture arrays $P, Q \in \Sigma^{**}$, a one-step derivation in the *RPIS* Γ, denoted by $P \Rightarrow Q$, yields Q from P whenever either (i) or (ii) holds: (i) if ${}^t a_1 \cdots a_m$ and ${}^t b_1 \cdots b_m$ are two adjacent columns, namely, columns j and $j+1$ for some $j, 1 \leq j \leq n-1$, in the picture array P of size $m \times n$, and if $(a_i, \alpha_i, b_i), 1 \leq i \leq m$ are column insertion rules in a column insertion table in I_c, then the rules can be applied in parallel by inserting α_i in the i^{th} row between $a_i \in \Sigma$ and $b_i \in \Sigma$ for all i, $1 \leq i \leq m$ or (ii) if $d_1 \cdots d_n$ and $e_1 \cdots e_n$ are two adjacent rows, namely, rows k and $k+1$ for some $k, 1 \leq k \leq m$, in the picture array P of size $m \times n$, and if $(d_i, {}^t\beta_i, e_i), 1 \leq i \leq m$ are row insertion rules in a row insertion table in I_r, then the rules can be applied in parallel by inserting β_i in the i^{th} column between $e_i \in \Sigma$ and $d_i \in \Sigma$ for $1 \leq i \leq n$. Likewise insertions to the immediate left or right (respy. immediate up or down) of a column (respy. row) in the picture array P can be defined by requiring the corresponding left or right or up or down contexts in the rules used to be λ.

The picture language $L(\Gamma)$ generated by Γ consists of picture arrays derived in one or more finite number of derivation steps starting with an axiom array in A. The family of picture languages generated by *RPIS* is denoted by *RPIL*.

Example 1. Consider the *RPIS* $\Gamma_1 = (\Sigma, I_c, I_r, A\})$ where $\Sigma = \{a, b\}$, $I_c = \{c_1\}, I_r = \{r_1\}$, where $c_1 = \{(a, ab, b)\}$, $r_1 = \{(a, a, \lambda), (b, b, \lambda)\}$ and A consists of the array $\begin{matrix} a & b \\ a & b \end{matrix}$

Γ_1 generates a picture language L_1 consisting of picture arrays $X \circ Y$ with X over a and Y over b and $|X|_r = |Y|_r, |X|_c = |Y|_c$. A member of L_1 is shown in Fig. 1. Starting with the axiom array in A, in any step of a derivation, the application of the column insertion rule of c_1 inserts ab in every row between a and b while the application of the row insertion rule of r_1 inserts in every column and just below a row, the symbol a below a and the symbol b below b. In other

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

Fig. 1. A picture array in the language L_1

words, for example, a derivation in which rules of c_1 and then rules of r_1 are applied, is shown below:

$$\begin{array}{l} a\ a\ b\ b \\ a\ a\ b\ b \\ a\ a\ b\ b \end{array} \Rightarrow^{c_1} \begin{array}{l} a\ a\ a\ b\ b\ b \\ a\ a\ a\ b\ b\ b \\ a\ a\ a\ b\ b\ b \end{array} \Rightarrow^{r_1} \begin{array}{l} a\ a\ a\ b\ b\ b \\ a\ a\ a\ b\ b\ b \\ a\ a\ a\ b\ b\ b \\ a\ a\ a\ b\ b\ b \end{array}$$

We now define an array P system with picture arrays as objects and with restricted picture insertion rules in the regions.

Definition 2. *An array P System with restricted picture insertion rules (APRPIS) is $\Pi = (\Sigma, \mu, F_1, \ldots, F_m, R_1, \ldots, R_m, i_o)$, where: Σ is the alphabet, μ is a membrane structure with m membranes labelled in a one-to-one way with $1, 2, \cdots, m$; F_i, $1 \leq i \leq m$, is a finite set (can be empty) of axiom picture arrays over Σ in the i^{th} region of μ; R_i, $1 \leq i \leq m$ is a finite set of column or row insertion tables as in a RPIS in the i^{th} region μ; each of the tables has an attached target here, out, in. (as usual, here is omitted and is understood). i_o is the label of an elementary membrane of μ (the output membrane).*

A computation in APRPIS is defined in the same way as in an array-rewriting P system [16, 19] with the successful computations being the halting ones: each array, from each region of Π, which can be obtained by applying the restricted picture insertion rules to the arrays associated with that region, should be obtained but rules of only one table is applied ; the array obtained in a region is placed in the region indicated by the target associated with the table of rules used; the target (here indicates that the array remains in the same region, out indicates that the array is sent to the immediate outer region except for the outermost skin membrane in which case the array sent out is "lost" in the environment ; and in indicates that the array is sent to the immediately inner membrane, nondeterministically chosen (but if no innner membrane exists, then the table of rules with the target indication in cannot be used). A computation is successful only if it stops and a configuration is reached where no table of rules can be applied to the existing arrays. The result of a halting computation consists of the arrays placed in the output membrane with label i_o in the halting configuration. The set of all such arrays computed or generated by the system Π is denoted by $AL(\Pi)$. The family of all array languages $AL(\Pi)$ generated by systems Π as above, with at most m membranes, is denoted by $AP_m(RPIS)$.

We illustrate with an example.

Example 2. Consider the $APRPIS$ $\Pi_1 = (\Sigma, [_1\,[_2\,]_2\,]_1, F_1, F_2, R_1, R_2, 2)$ where $\Sigma = \{a, b, e\}$, $F_1 = \{\begin{smallmatrix} a & b \\ a & b \end{smallmatrix}\}$, $F_2 = \phi$. R_1 consists of the column insertion tables $(c_1, in), (c_2, in)$ and R_2 consists of row insertion table (r, out) where
$c_1 = \{(a, ab, b)\}$, $c_2 = \{(a, e, b)\}$, $r = \{(a, a, \lambda), (b, b, \lambda)\}$.

In a computation in Π_1, since only the region 1 has an axiom picture array $\begin{smallmatrix} a & b \\ a & b \end{smallmatrix}$, application of the rule of the table c_1 will insert ab between a and b in every row and the resulting array is sent to region 2 due to target indication in. Application of the rules in r in region 2 inserts a row below one of the rows with a below a and b below b and the array is sent back to region 1 due to target indication out. The process can repeat. If in region 1, the rule of the table c_2 is applied then e is inserted between a and b in every row resulting in an array of the form $\begin{smallmatrix} a \cdots a & e & b \cdots b \\ \vdots \ddots \vdots & \vdots & \vdots \ddots \vdots \\ a \cdots a & e & b \cdots b \end{smallmatrix}$ and the array is sent to the output region 2. The computation halts as no other rule could be applied. Note that in the rules in r there is no rule that allows insertion below e and hence the rules of the table r can no longer be applied. The arrays generated are of the form $M_1 \circ M_2 \circ M_3$ where M_1 is an array over a, M_3 is an array over b and M_2 is an array with only one column of $e's$. Also, $|M_1|_c = |M_3|_c = |M_1 \circ M_2 \circ M_3|_r$. In otherwords, the number of rows of the array generated equals the number of columns of M_1 which equals the number of columns of M_3. One such array is shown in Fig. 2.

$$\begin{matrix} a\ a\ a\ e\ b\ b\ b \\ a\ a\ a\ e\ b\ b\ b \\ a\ a\ a\ e\ b\ b\ b \end{matrix}$$

Fig. 2. A picture array generated by Π_1

Theorem 1.

$$RPIL = AP_1(RPIS) \subset AP_2(RPIS)$$

Proof. In a $RPIS$ as well as $APRPIS$ with one membrane, the rules are only the $RPIS$ rules and so it is clear that $RPIL = AP_1(RPIS)$. Also by definition, the inclusion $AP_1(RPIS) \subseteq AP_2(RPIS)$ follows. The proper inclusion follows from Example 2 on noting that in any picture array M generated, the number of columns = 2× the number of rows +1. If there is only one membrane, there is no way to control application of the rules of the column and row tables as all the tables of rules are in the same region.

We now compare the generative power of $APRPIS$ with certain similar picture generating models. The insertion rules considered both in $RPIS$ and $APRPIS$

are restricted forms of the insertion rules considered in the picture insertion system defining the class $INPA$ [2]. Yet we show that there are picture array languages generated by $APRPIS$ with two membranes that are not in the class $INPA$. Also, in [17], a picture array generating model, called pure 2D context-free grammar ($P2DCFG$), is proposed and the generative power of this model is investigated. In this model, all symbols in a column or in a row of a $m \times n$ picture array are rewritten by pure context-free rules [17] of the form $a \to \alpha$ with α in all the rules in a table having the same length. The family of languages generated by $P2DCFG$ is denoted by $P2DCFL$. Two variants of $P2DCFG$, called $(l/u)P2DCFG$ and $(r/d)P2DCFG$ which are incomparable with $P2DCFG$ have also been considered [5, 18]. In a $(l/u)P2DCFG$ the symbols in the leftmost column or the uppermost row only are rewritten while in a $(r/d)P2DCFG$ the symbols in the rightmost column or the lowermost row are rewritten. We show that there are picture array languages that can be generated by $APRPIS$ with two membranes but cannot be in $P2DCFL$, $(l/u)P2DCFL$ and $(r/d)P2DCFL$.

Theorem 2. *1.* $AP_2(RPIS) - P2DCFL \neq \phi$
2. $AP_2(RPIS) - (l/u)P2DCFL \neq \phi$
3. $AP_2(RPIS) - (r/d)P2DCFL \neq \phi$
4. $AP_2(RPIS) - INPA \neq \phi$

Proof. Consider the picture array language L_2 consisting of $m \times (3n+2)$ arrays $(m \geq 2, n \geq 1)$, such that an array in L_2 is of the form $M_1 \circ M_2 \circ M_3 \circ M_4 \circ M_5$ where M_1, M_3, M_5 are $m \times n$ arrays over $\{a\}, \{b\}, \{d\}$ respectively and M_2, M_4 are $m \times 1$ arrays over $\{e\}$. The language L_2 cannot belong to any of the families $P2DCFL$, $(l/u)P2DCFL$ and $(r/d)P2DCFL$ because in $P2DCFG$ generating in different modes, a language in any of these families, symbols in an array can be rewritten in only one column at a time. The language L_2 cannot also belong to the class $INPA$ for a similar reason. This means that the array with columns of $a's$ in the beginning, columns of $b's$ in the middle and columns of $c's$ in the end, equal in number, cannot be generated.
The language L_2 is generated by the $APRPIS$ Π_2 with two membranes where

$$\Pi_2 = (\{a, b, d, e\}, [_1\, [_2\,]_2\,]_1, F_1, F_2, R_1, R_2, 2)$$

with $F_1 = \{\begin{matrix} a\ b\ d \\ a\ b\ d \end{matrix}, \}$ $F_2 = \phi$. R_1 consists of the column insertion table (c_1, in) and a row insertion table $(r, here)$ and R_2 consists of the column insertion tables $(c_3, out), (c_2, here), (c_4, here)$ where $c_1 = \{(a, ab, b)\}$, $c_2 = \{(a, e, b)\}$, $c_3 = \{(b, d, d)\}$, $c_4 = \{(b, be, d)\}$, and $r = \{(a, a, \lambda), (b, b, \lambda), (d, d, \lambda)\}$.
The region 1 alone has an axiom picture array. So a computation in Π_2 can start in this region with the applicable tables of rules. If the rule of column insertion table c_1 is applied then ab will be inserted in every row between two adjacent columns of $a's$ and $b's$ and the array is sent to region 2. If in this region the rule of column insertion table c_3 is applied then d will be inserted in every row between two adjacent columns of $b's$ and $d's$ and the array is sent back to region

1. The process can repeat. At anytime, the rule of the row insertion table r can be applied, thus inserting in the picture array a row of the form $a \cdots ab \cdots bd \cdots d$. If at anytime, instead of c_3, the rule of the column insertion table c_2 (respy. c_4) is applied in region 2, then a column of $e's$ will be inserted between two adjacent columns of $a's$ and $b's$ (respy. columns of $b's$ and $d's$). A correct sequence of application of the of rules is to apply now in region 2, the rule of the column table c_2 followed by c_4 or c_4 followed by c_2, thus generating a picture array of the form shown in Fig. 3. Region 2 is the output region and arrays collected here form the picture array language generated which is L_2.

$$\begin{array}{c} a \cdots a\ e\ b \cdots b\ e\ d \cdots d \\ \vdots\ \ddots\ \vdots\ \vdots\ \vdots\ \ddots\ \vdots\ \vdots\ \vdots\ \ddots\ \vdots \\ a \cdots a\ e\ b \cdots b\ e\ d \cdots d \end{array}$$

Fig. 3. A picture array generated by Π_2

A well-known class, called the family of context-sensitive matrix languages ($CSML$), of two-dimensional picture array languages was introduced by
Siromoney et al. [13]. The corresponding class of grammars, known as $CSMG$ involves two phases of generation with the first phase generating a string s over intermediate symbols and in the second phase all the intermediate symbols in s are rewritten in parallel by regular nonterminal rules of the form $A \to aB$, or together terminated by regular terminal rules of the form $A \to a$, (A, B are nonterminals and a is a terminal symbol), thus generating the columns of the picture array. The words in the first phase form a context-sensitive language. An extension, called $TCSML$ of the family $CSML$ was introduced in [15] by having in a $CSMG$, an additional feature of tables of nonterminal rules or tables of terminal rules in the second phase. It was shown in [15] that $CSML \subset TCSML$. We show here that there are picture array languages that cannot be in $TCSML$ but can be generated by $APRPIS$ with two membranes.

Theorem 3. $AP_2(RPIS) - TCSML \neq \phi$

Proof. Consider the picture array language L_3 consisting of $(2m+1) \times (2m+1)$ arrays, $(m \geq 1)$, such that an array in L_3 is of the form $(M_1 \circ M_2 \circ M_3) \diamond M_4 \diamond (M_3 \circ M_2 \circ M_1)$ where M_1, M_3 are $m \times m$ arrays over $\{a\}, \{b\}$ respectively, M_2 is an $m \times 1$ array over $\{e\}$ and M_4 is an $1 \times (2m+1)$ array over $\{e\}$. A member of L_3 is shown in Fig. 4. The language L_3 cannot belong to the family $TCSML$. In fact in a grammar generating a picture array in a language in $TCSML$, the number of rows above and below a distinct row of a generated array, cannot be maintained to be equal as the application of the tables of rules cannot be controlled.
The language L_3 is generated by the $APRPIS$ Π_3 with two membranes where

$$\Pi_3 = (\{a, b, e\}, [_1\, [_2\,]_2\,]_1, F_1, F_2, R_1, R_2, 2)$$

with $F_1 = \{ \begin{matrix} a & b \\ b & a \end{matrix}, \}$ $F_2 = \phi$. R_1 consists of the column insertion tables $(c_1, in), (c_2, in)$ and R_2 consists of the row insertion tables $(r_3, out), (r_4, here)$ where

$$c_1 = \{(a, ab, b), (b, ba, a)\}, c_2 = \{(a, e, b), (b, e, a)\},$$

$$r_3 = \{(a, {}^t ab, b), (b, {}^t ba, a)\}, r_4 = \{(a, e, b), (b, e, a), (e, e, e)\}.$$

Starting with the axiom picture array in region 1, a computation in Π_3 can

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

Fig. 4. A picture array generated by Π_3

start in this region with the applicable tables of rules. If the rules of column insertion table c_1 are applied then ab will be inserted in every row between the two symbols a and b in two adjacent columns while ba will be inserted in every row between the two symbols b and a in the same two adjacent columns, thus adding two columns to the picture array. The array generated is sent to region 2. If in this region the rules of row insertion table r_3 is applied then ab will be inserted in all columns between the symbols a and b in two adjacent rows while ba will be inserted between the symbols b and a in the same two adjacent rows, thus adding two rows to the picture array and the array is sent back to region 1. The process can repeat. If at anytime, the rules of the column insertion table c_2 is applied in region 1, then a column of $e's$ will be inserted between the two adjacent columns ${}^t a \cdots ab \cdots b$ and ${}^t b \cdots ba \cdots a$ and the array is sent to region 2. A correct sequence of application of the tables of rules is to apply now in region 2, the rules of the row table r_4, thus generating a picture array of the form shown in Fig. 4. Region 2 is the output region and arrays collected here form the picture array language generated which is L_3.

Head [4] introduced splicing systems while proposing a theoretical model of DNA recombination. Berstel et al. [1] introduced a variant of splicing system, known as flat splicing system and noted [1, p4] that the flat splicing system involving the operation of flat splicing on words and the insertion system [8] involving the operation of insertion on words, have similarity but the systems are incomparable. Extending the operation of flat splicing to picture arrays, a picture array generating model, called array flat splicing system (AFS), was

introduced and investigated in [7]. Here we show that picture array languages that cannot be generated by any AFS can be generated by $APRPIS$ with two membranes. The family of picture array languages generated by AFS is denoted by $L(AFS)$.

Theorem 4. $AP_2(RPIS) - L(AFS) \neq \phi$

Proof. Consider the picture array language L_4 consisting of $m \times (n+2)$ arrays, $(m \geq 2, n \geq 1)$, such that an array in L_4 is of the form $M_1 \circ M_2 \circ M_3$ where M_1, M_3 are $m \times 1$ arrays over $\{x\}, \{y\}$ respectively and M_2 is an $m \times n$ array over $\{a\}$. a member of L_4 is shown in Fig. 5. The language L_4 does not belong to the family $L(AFS)$ [7].
The language L_4 is generated by the $APRPIS$ Π_4 with two membranes where

$$\Pi_4 = (\{x, y, a\}, [_1\, [_2\,]_2\,]_1, F_1, F_2, R_1, R_2, 2)$$

with $F_1 = \{\begin{matrix} x\ a\ y \\ x\ a\ y \end{matrix}, \}$ $F_2 = \phi$. R_1 consists of the column insertion tables $(c_1, here), (c_2, in)$ and a row insertion table $(r, here)$ where

$$c_1 = \{(x, a, a)\}, c_2 = \{(a, a, y)\}, r = \{(x, x, \lambda), (y, y, \lambda), (a, {}^t a, \lambda)\}.$$

We note that the rule of the column insertion table c_1 inserts a column of $a's$

$$\begin{matrix} x\ a\ a\ a\ a\ y \\ x\ a\ a\ a\ a\ y \\ x\ a\ a\ a\ a\ y \\ x\ a\ a\ a\ a\ y \end{matrix}$$

Fig. 5. A picture array generated by Π_4

but the array remains in the region 1 itself. The rule of the column insertion table c_2 has the same effect but the array is sent to the output region 2. The rule of the row insertion table r inserts a row of the form $xa \cdots ay$ while an array is generated.

4 Application to "kolam" pattern generation

A "kolam" pattern [14] is a visually pleasing floor design, more common in South India, drawn with curly lines around points arranged in a particular pattern, resulting in the intended "kolam" drawing. In [14] "kolam" patterns were considered as picture arrays in the two-dimensional plane with the labels of the cells of the picture array representing primitive "kolam" patterns [6]. In fact array grammars developed in [14] were used to generate the picture arrays and the labels were replaced by the corresponding primitive patterns, thus yielding

the kolam pattern composed of these primitive patterns. Here we illustrate by constructing a $APRPIS$ generating picture arrays representing a particular set of "kolam patterns", one member of which is shown in Fig. 7. The labels of the arrays stand for the primitive kolam patterns defining the "kolam". An array representing this kind of a kolam is shown in Fig. 8.

Let L_k be the picture language consisting of $(m+2)\times(2n+3)$, $m \geq 1, n \geq 1$ pic-

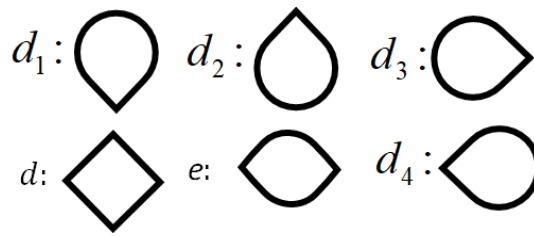


Fig. 6. A set of primitive "Kolam" patterns.

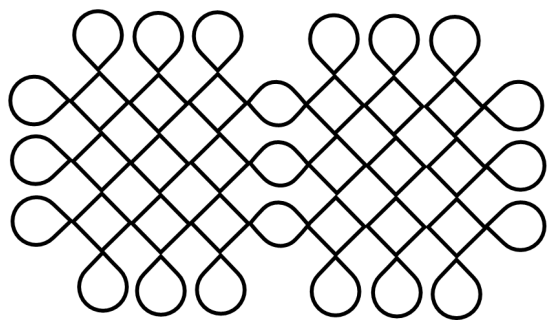


Fig. 7. A "Kolam" pattern.

ture arrays p such that $p(i,1) = p(i,n+2) = p(i,2n+3) = b$, for $i \in \{1, m+2\}$, $p(1,j) = d_1$ for $2 \leq j \leq n+1; n+3 \leq j \leq 2n+2$, $p(n+2,j) = d_2$ for $2 \leq j \leq n+1; n+3 \leq j \leq 2n+2$, $p(i,1) = d_3$, for $2 \leq i \leq n+1$, $p(i,2n+3) = d_4$, for $2 \leq j \leq n+1$, $p(i,n+2) = e$, for $2 \leq i \leq n+1$ and all other $p(i,j) = d$. Here d_1, d_2, d_3, d_4, d, e stand for the "kolam" primitive patterns as in Fig. 6 and b stands for blank.

The picture language L_k is generated by the $APRPIS$ with alphabet $\{d_1, d_2, d_3, d_4, d, e, b\}$, membrane structure $[_1\,[_2\,[_3\,]_3\,]_2\,]_1$, axiom array $\begin{matrix} b & d_1 & b & d_1 & b \\ d_3 & d & e & d & d_4 \\ b & d_2 & b & d_2 & b \end{matrix}$

$$\begin{array}{ccccccccc} b & d_1 & d_1 & d_1 & b & d_1 & d_1 & d_1 & b \\ d_3 & d & d & d & e & d & d & d & d_4 \\ d_3 & d & d & d & e & d & d & d & d_4 \\ d_3 & d & d & d & e & d & d & d & d_4 \\ b & d_2 & d_2 & d_2 & b & d_2 & d_2 & d_2 & b \end{array}$$

Fig. 8. Array representing the "kolam" in Fig. 7.

in region 1 and no axiom array in regions 2 and 3, with 3 as the output region. The sets R_1, R_2, R_3 are the sets of tables of insertion rules in regions 1,2 and 3 respectively, where R_1 contains a column insertion table (c_1, in) and a row insertion table $(r_1, here)$, R_2 contains two column insertion tables (c_2, out) and (c_3, in). Here

$$c_1 = \{(d_1, d_1, b), (d, d, e), (d_2, d_2, b)\}, c_2 = \{(b, d_1, d_1), (e, d, d), (b, d_2, d_2)$$

$$c_3 = \{(d_1, d_1, b), (d, d, d_4), (d_2, d_2, b)\},$$

$$r_1 = \{(b, d_3, d_3), (d_1, d, d), (b, e, e), (b, d_4, d_4)\}.$$

In a computation, the rules of table c_1 insert a column of the form ${}^t d_1 d \cdots d d_2$ to the left of the middle column of $e's$ while the rules of table c_2 insert a similar column to the right of the middle column of $e's$. The generated array is sent to region 2 from region 1 on applying c_1 while it is sent from region 2 to region 1 on applying c_2. The process can repeat adding equal number of columns to the left and right of the middle column. The application of the rules of the table r_1 inserts below the first row, a row of symbols $d_3 d \cdots d e d \cdots d d_4$ as many times as needed and the generated array remains in region 1. When the rules of the table c_3 are applied in region 2, it has the same effect as applying c_2 but the insertion is to the left of the last column and the array is sent to the output region 3, with the computation halting. The generated arrays are the picture arrays of L_k and no other array is generated. On replacing the labels of the squares in a generated picture array by the corresponding primitive kolam patterns shown in Fig. 6, we obtain the "kolam" itself. One such generated "kolam" is shown in Fig. 7.

Acknowledgement

The authors thank the reviewers for their very useful comments and for pointing out corrections which very much helped to improve the presentation of the paper. This work was supported by the National Natural Science Foundation of China (61672437, 61702428), the Sichuan Science and Technology Program (2018GZ0185,2018GZ0086), and the New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0044).

References

1. Berstel, J., Boasson, L., Fagnot, I. : Splicing systems and the Chomsky hierarchy, Theor. Comput. Sci. 436, 2–22 (2012).

2. Fujioka K. : A Two-Dimensional Extension of Insertion Systems. In: Dediu AH., Lozano M., Martín-Vide C. (eds) Theory and Practice of Natural Computing. TPNC 2014. Lecture Notes in Computer Science, vol 8890. Springer, Cham Vol. 8890, 181–192 (2014).
3. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Rozenberg, G., Salomaa, G. (eds.) Handbook of Formal Languages, pp. 215-267. Springer, Berlin (1997).
4. Head, T.: Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours. Bull. Math. Biol. 49, 735–759 (1987).
5. Krivka, Z., Martín-Vide, C., Meduna, A., K. G. Subramanian: A Variant of Pure Two-Dimensional Context-Free Grammars Generating Picture Languages. IWCIA 2014. Lecture Notes in Computer Science. Springer Verlag, Vol. 8466, 123–133 (2014).
6. Nagata, S., Robinson, T. : Digitalization of Kolam Patterns and Tactile Kolam Tools, In : K.G. Subramanian et al. (eds.) Formal Models, Languages and Applications, World Scientific Publishing, 354–363 (2007).
7. Pan, L., Nagar, A.K., K.G. Subramanian., B. Song. : Picture Array Generation Using Flat Splicing Operation. J. Comput. Theoret. Nanoscience. 13, 3568-3577 (2016).
8. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Springer, New York (1998).
9. Păun, Gh. : Computing with membranes, J. Comp. System Sci., 61, 108– 143 (2000).
10. Păun, Gh.: Membrane Computing: An Introduction. Springer-Verlag Berlin, Heidelbrg, 2000.
11. Păun, Gh., Rozenberg, G., Salomaa, A. : The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)
12. Rozenberg, G., Salomaa, A. (Eds.): Handbook of Formal Languages. Vols. 1 - 3, Springer-Verlag, Berlin (1997).
13. Siromoney, G., Siromoney, R., Krithivasan, K.: Abstract families of matrices and picture languages. Computer Graphics and Image Proc. **1**, 284–307 (1972).
14. Siromoney, G., Siromoney, R., Krithivasan, K.: Array grammars and Kolam. Computer Graphics and Image Proc. **3(1)**, 63–82 (1974).
15. Siromoney, R., Subramanian, K.G., Rangarajan, K.: Parallel/sequential rectangular arrays with tables, Int. J. Computer Math. **6(2)** 143-158 (1977).
16. Subramanian, K.G. : P systems and picture languages. Lecture Notes in Computer Science, Springer Verlag, Vol. 4664, 99–109 (2007).
17. 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).
18. Subramanian, K.G., Geethalakshmi, M., David, N.G., Nagar, A.K. : Picture Array Generation Using Pure 2D Context-Free Grammar Rules. IWCIA 2015 Lecture Notes in Computer Science. Springer Verlag, Vol. 4664, 187–201 (2015).
19. Subramanian K.G., Sriram S., Song B., Pan L. : An Overview of 2D Picture Array Generating Models Based on Membrane Computing. In: Adamatzky A. (eds) Reversibility and Universality. Emergence, Complexity and Computation. Springer vol 30, 333–356 (2018).
20. Zhang, G., Pérez-Jiménez, M.J., Păun, Gh. : Real-life Applications with Membrane Computing, In: Emergence, Complexity and Computation Series, 2017.

Asynchronous tissue P systems with local synchronization

Bosheng Song and Linqiang Pan*

Key Laboratory of Image Information Processing and Intelligent Control,
School of Automation, Huazhong University of Science and Technology,
Wuhan 430074, Hubei, China
boshengsong@hust.edu.cn, lqpan@mail.hust.edu.cn

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 [11]. The computational models that investigated in membrane computing are generically called *P systems*. According to the membrane structure of P systems, there exist two main families: *cell-like P systems* (described by trees) and *tissue-like P systems* [8] or *neural-like P systems* [7] (described by directed graphs).

Tissue P systems consider arbitrary graphs as underlying structures [8], where cells placed in nodes and edges correspond to communication channels. In tissue P systems, every cell may have connection with the environment and every cell may have connection with any other cell. The movement of objects through communication channels is controlled by a set of communication (symport/antiport) rules [6, 10], where a communication rule is called a symport rules if a multiset of objects is moved from one region to another region in one direction, and a communication rule is called an antiport rule if simultaneous two multisets of objects are moved between two cells or between a cell and the environment in opposite directions.

There are several strategies for rule application considered in cell-like P systems, such as minimal parallelism [2], asynchronous [3–5], flat maximal parallelism [9, 14], time-freeness [13, 15]. In [1], spiking neural P systems with a non-synchronized use of rules were considered, where in any step, the enabled rules in a neuron can apply or not apply. It is proved that asynchronous spiking neural P systems with extended rules (bounded neurons or unbounded neurons) are not computationally complete [1]. In [12], the notion of local synchronization was introduced into asynchronous spiking neural P systems, where the system is worked in an asynchronous way, furthermore, a family of sets of neurons (also called ls-sets) is specified; if one of the neurons from an ls-set fires, then all neurons from this set should fire if they have enabled rules. It is shown that asynchronous SN P systems with local synchronization consisting of unbounded neurons and using standard spiking rules are proved to be universal [12].

In this work, the notion of local synchronization is introduced into asynchronous tissue P systems, where the application of rules in such P systems is

* Corresponding author.

described as follows: in each time unit, any number of applicable rules in the system may be used; furthermore, for cells in the same ls-sets, if one of rules in these cells is used, then all enabled rules in this set should be used in a maximally parallel way (note that rules in the same ls-set are the rules that all the involved regions belong to the same synchronized component). The computational power of synchronous tissue P systems with local synchronization is investigated. We prove that asynchronous tissue P systems with one cell, using antiport rules of length 3 or using symport rules of length 3 can simulate partially blind register machines; and the converse simulation holds for tissue P systems. Moreover, if we consider local synchronization of using rules in asynchronous tissue P systems, then such P systems with three cells, symport rules of length 2 and antiport rules of length 4 are Turing universal.

References

1. M. Cavaliere, O.H. Ibarra, Gh. Păun, O. Egecioglu, M. Ionescu, S. Woodworth, Asynchronous spiking neural P systems, Theor. Comput. Sci. 410 (2009) 2352–2364.
2. G. Ciobanu, L. Pan, Gh. Păun, M.J. Pérez-Jiménez, P systems with minimal parallelism, Theor. Comput. Sci. 378 (2007) 117–130.
3. R. Freund, Asynchronous P systems and P systems working in the sequential mode, Lect. Notes Comput. Sci. 3365 (2005) 36–62.
4. P. Frisco, About P systems with symport/antiport, in: Proceedings of the second Brainstorming Week on Membrane Computing, Seville, Spain, 2004, pp. 224–236.
5. P. Frisco, G. Govan, A. Leporati, Asynchronous P systems with active membranes, Theor. Comput. Sci. 429 (2012) 74–86.
6. P. Frisco, H.J. Hoogeboom, P systems with symport/antiport simulating counter automata, Acta Inform. 41 (2) (2004) 145–170.
7. M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems, Fund. Informa. 71 (2-3) (2006) 279–308.
8. C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodriguez-Paton, Tissue P systems, Theor. Comput. Sci. 296 (2) (2003) 295–326.
9. L. Pan, Gh. Păun, B. Song, Flat maximal parallelism in P systems with promoters, Theor. Comput. Sci. 623 (2016) 83–91.
10. A. Păun, Gh. Păun, The power of communication: P systems with symport/antiport, New Generat. Comput. 20 (3) (2002) 295–305.
11. Gh. Păun, Computing with membranes. J. Comput. Syst. Sci. 61(1), 108–143 (2000).
12. T. Song, L. Pan, Gh. Păun, Asynchronous spiking neural P systems with local synchronization, Inf. Sci. 219 (2013) 197–207.
13. B. Song, M.J. Pérez-Jiménez, L. Pan, An efficient time-free solution to SAT problem by P systems with proteins on membranes, J. Comput. Syst. Sci. 82 (2016) 1090–1099.
14. B. Song, M.J. Pérez-Jiménez, Gh. Păun, L. Pan, Tissue P systems with channel states working in the flat maximally parallel way, IEEE T. Nanobiosci. 15 (7) (2016) 645–656.
15. B. Song, T. Song, L. Pan, A time-free uniform solution to subset sum problem by tissue P systems with cell division, Math. Struct. Comput. Sci. 27 (1) (2017) 17–32.

A parallel implementation of image edge detection by using enzymatic numerical P systems

Jianying Yuan[1,2,4], Dequan Guo[2,1,4], Gexiang Zhang[3,5] ⋆, Prithwineel Paul[5], Ming Zhu[2], and Qiang Yang[3]

[1] The Postdoctoral Station at Xihua University Based on Collaboration Innovation Center of Sichuan Automotive Key Parts, Xihua University, Chengdu, 610039, China
yuanjy,guodq@cuit.edu.cn
[2] Chengdu University of Information Technology, Chengdu, 610225, China
zhuming@126.com
[3] Robotics Research Center, Xihua University, Chengdu, 610039, China
zhgxdylan@126.com,qiangychd@126.com
[4] School of Aeronautics and Astronautics, University of Electronic Science and Technology, Chengdu, 610054, China
[5] School of Electrical Engeneering, Southwest Jiaotong University, Chengdu, 610031, China
prithwineelpaul@gmail.com

Abstract. Image edge detection is a fundamental problem in image processing and computer vision, particularly in the area of feature extraction. However, the time complexity increases squarely with the increase of image resolution in conventional serial computing mode. This results in time consuming when dealing with large amount of image data. In this work, a novel parallel implementation algorithm for gradient based edge detection (ED), namely EDENP, is proposed. The proposed bio-inspired parallel algorithm is in the framework of enzymatic numerical P system (ENPS). The mathematical model, system structure, execution process, and required resources are elaborated in detail. The performance and efficiency of this algorithm are evaluated on the CUDA platform. The advantage of EDENP is that it has the property of low computational complexity independent of the image resolution, with keeping the same edge detection performance as the original algorithm in image processing.

Keywords: Membrane computing; edge detection; enzymatic numerical P system; gradient based edge detection

1 Introduction

In the past decades, image processing technology has experienced dramatic growth and widespread applications. Nearly no area is not impacted in some way by digital image processing [1]. Normally, digital image processing includes three main levels, i.e., low-level, mid-level and high-level processing [2]. As one of the most important low-level image processing algorithms, edge detection has been extensively used in target

⋆ Corresponding author.

tracking [3], image compression [5], and object recognition [4,6]. Gradient based edge detection is one of the classical edge detection algorithms with the merits of good performance and moderate complexity, which is very suitable for real time image processing [7]. However, convolution calculation (i.e., a classic neighborhood computing in image processing) [8] is involved in the gradient computation of each pixel, and hence the time complexity increases squarely with the increase of image resolution. So it is difficult to deal with an image with large resolution, such as remote sensing image, medical image, etc., in real time processing.

In order to achieve real-time calculation of high resolution images, many researchers have put much effort and several methods have been proposed. The most commonly used solution is to run the program on Graphics Processing Unit (GPU), as described in [10,11]. GPU uses hundreds of parallel processor cores executing tens of thousands of parallel threads to rapidly solve large problems having substantial inherent parallelism [12]. However, with the shrinking volume of chips, semiconductor technology begins to reach its physical limits, which means the performance of conventional computing technique based on silicon chip integrated circuit microprocessors will be difficult to improve further [13]. Under this background, some scholars have turned their attention to non-traditional computing [14,15]. Membrane computing (MC) is a active branch of natural computing, which investigates distributed and parallel computing models, abstracted from the compartmentalized structure and interactions of living cells [16,17,18]. It has been considered as a promising method to resolve the conflict between the ever-increasing amount of data in the image processing field and the backward computing power of conventional computer [19,20]. During the past years, image smoothing [30], obtaining homology groups of 2D images [21,22], counting cell [23], image thinning [24], symmetric dynamic programming stereo [25] and corner detection [26] in MC framework have been vividly studied.

For edge detection, edge based image segmentation procedures by using a tissue-like P system with degree 2 were described [27,28,29]. Although the computational complexity can be greatly reduced, the performances of those tissue-like P system based algorithms are far inferior to the performance of the existing algorithms in image processing. In [50], a new operator, called AGP Segmentator, which is a variation of the Sobel operator was designed and implemented by using tissue-like P system for image segmentation. Although several testing examples were given in [50] to illustrate that the new AGP operator can improve the performance of the traditional Sobel operator, the conclusion is still not theoretically proven.

In this paper, we focus on the parallel implementation of gradient based edge detection algorithm. There are two differences from the previous literature in the MC framework in this work. One is that a variant of enzymatic numerical P systems (ENPS) are introduced. Besides the features as described in [30], ENPS have several good features which makes it suitable for image processing. One of it is that numerical variables and numerical expressions can be used directly in ENPS. Thus some complex mathematical calculations can be achieved without coding and decoding processes. Another important feature is that enzymatic variables can control the execution order of multiple rules, i.e., the algorithms with complex logical relationships can be realized. The other difference is that we did not change the mathematical model of the original edge

detection algorithm. Hence, the parallel implementation results of the proposed algorithm are consistent with the results of the original edge detection algorithm run on the serial computing platform. Most of the original classical image processing algorithms have been widely used in practical engineering because their performance can meet the requirements of practical engineering. Under the precondition of keeping the mathematical model of original algorithm unchanged, the designed parallel implementation algorithm can be directly applied to the real image processing project without the need to perform large-scale testing of the performance of the new algorithm.

The main contribution of this paper is the design of a novel parallel algorithm, named EDENP, for gradient based edge detection in the framework of ENPS. The significant advantage of EDENP is that it can achieve the time complexity of $O(1)$ theoretically, no matter how large the image resolution is. Moreover, the performance is equivalent to the performance of the original gradient based edge detection algorithm in image processing.

The rest of this paper is structured as follows. Section 2 introduces the definition, characteristics and applications of ENPS. The problem statement is elaborated in section 3. Section 4 discusses the EDENP algorithm in detail. The experiments and results are presented in Section 5. Conclusions are drawn in Section 6.

2 Enzymatic numerical P systems

Membrane Computing is a new computing model abstracted from the structure and functioning of living cells with properties of nondeterministic distributed and parallel computing models. According to the way in which membranes are structured, there are three major types of P systems, i.e., cell-like [16], tissue-like [31] and spiking neural P systems [32], and the main basic data structure associated with membranes include multisets, strings or numerical variables. A numerical P system (NPS) is a new special research branch of cell-like P systems, proposed by the founder of Membrane Computing, Păun [33]. The traditional multisets of objects associated to membranes are replaced by sets of numerical variables, and the evolutionary rule are composed of a production function and a repartition protocol in NPS[52][54][55]. Although it was inspired from economic phenomena, it has been widely applied in robot controller design [34,35,36,37]. For example, in [34,37], it was used for obstacle avoidance and wall following, and in [35] a trajectory tracking system of nonholonomic wheeled mobile robot was constructed with ENPS. Since deterministic numerical P systems can only execute one production function per membrane at a time, when there are multiple production functions per membrane, one is selected randomly. This will limit its application in some situation where the rules should be executed deterministically. In order to solve this problem and expand the application of NPS, ENPS were proposed [38]. It is extended from numerical P system by introducing enzyme-like variables. The standard form is defined as follows:

$$\Pi = \Big(m, H, \mu, \big(Var_1, E_1, \mathrm{Pr}_1, Var_1(0)\big), ..., \big(Var_m, E_m, \mathrm{Pr}_m, Var_m(0)\big)\Big).$$

where:

- m is the number of membranes used.
- H is an alphabet that contains m symbols, and $H = \{1, 2, \ldots, m\}$.
- μ is the membrane structure.
- Var_i is the set of variables from membrane i and $Var_i\,(0)$ are the initial values for these variables.
- Pr_i is the set of rules in membrane i, composed of a production function and a repartition protocol. A typical rule is as follows.

$$\mathrm{F}_{l,i}\left(y_{1,i}, ..., y_{k_l,i}\right)|_{e_{j,i}} \rightarrow c_{l,1}|v_1 + c_{l,2}|v_2 + ... + c_{l,n_i}|v_{n_i},$$

where $e_{j,i}$ is a variable from Var_i different from $y_{1,i}, \ldots, y_{k_l,i}$ and $v_1, v_2, \ldots, v_{n_i}$. The rule can be executed at a time t only if $e_{j,i} > \min\left\{y_{1,i}\left(t\right), y_{2,i}\left(t\right), ...y_{k_l,i}\left(t\right)\right\}$. From the definition of ENPS, it is clear that with enzymes-like variables, the system can control multiple production functions to run in parallel in the same membrane deterministically[53]. Hence, it can overcome the disadvantages of traditional numerical P systems that only run one rule nondeterministically at a time in a membrane. The ENPS with deterministic, parallel execution model has already been proved to be Turing universal [39,40]. In [41], it is shown that any ENPS working in all-parallel mode or one parallel model can be simulated by an equivalent one-membrane ENPS working in the same mode. Linear production functions, each depending upon at most one variable, suffice to reach universality for both computing modes. Since the proposal of ENPS, this model has been successfully applied in a wide range of domains, such as robot control [43], big data field [44], pole balancing control [45], and logic operations and sorting [46] fields. In this work, ENPS is used to solve the problem of gradient based image edge detection.

3 Problem Statement

The GBED work is based on the assumption that the edge occurs where the gradient of brightness intensity of each pixel is high. A pair of convolution masks is used to estimate the gradients in the x and y directions, respectively. Sobel, Prewitt, and Roberts operators are classic convolution masks of GBED, as shown in formulas (1-3). In this paper, we take Sobel operator as an example of GBED. When the masks are sliding over the image, a square of pixels are operated. Then both directional gradients ($g_{x_{i,j}}$, $g_{y_{i,j}}$) and absolute gradient magnitude $g_{i,j}$ of a given pixel whose image coordinate is (i, j) are computed, as shown in formulas (4-5), where $2 \leq i, j \leq n-1$ for image with resolution of $n \times n$.

$$Sobel_x = \begin{bmatrix} -1\ 0\ 1 \\ -2\ 0\ 2 \\ -1\ 0\ 1 \end{bmatrix}; \quad Sobel_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1}$$

$$Prew_x = \begin{bmatrix} -1\ 0\ 1 \\ -1\ 0\ 1 \\ -1\ 0\ 1 \end{bmatrix}; \quad Prew_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2}$$

$$Rob_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}; \quad Rob_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{3}$$

$$g_{x_{i,j}} = Sobel_x * I(i,j); \quad g_{y_{i,j}} = Sobel_y * I(i,j) \tag{4}$$

$$g_{i,j} = \sqrt{{g_{x_{i,j}}}^2 + {g_{y_{i,j}}}^2} \tag{5}$$

When the gradient magnitude of one pixel is computed, a predefined threshold θ is used to judge whether this pixel is an edge pixel or not, according to formula (6). More concretely, if the gradient magnitude of the given pixel is greater than or equal to θ, then the pixel is assumed as an edge point, otherwise, it is a background point, as formula (7).

$$d_{i,j} = g_{i,j} - \theta \tag{6}$$

$$edg_{i,j} = \begin{cases} 1 \; if \; (d_{i,j} \geq 0) \\ 0 \; if \; (d_{i,j} < 0) \end{cases} \tag{7}$$

The program pseudo code of GBED run on conventional serial computer is illustrated in Algorithm 1, where the initial value of $edg_{i,j}$ is set to 0. From Algorithm 1, it can be deduced that the computational complexity is $O\left(n^2\right)$ because two loops are involved. When n is large, the calculations are very time-consuming under the serial computing platform.

Algorithm 1 The pseudo code of GBED

Input: $I(n * n)$
Output: $edg(n * n)$
1: **for** $i = 2 : n - 2$ **do**
2: **for** $j = 2 : n - 1$ **do**
3: Computing $g_{x_{i,j}}$
4: Computing $g_{y_{i,j}}$
5: Computing $g_{i,j}$
6: Computing $d_{i,j}$
7: Computing $edg_{i,j}$
8: **end for**
9: **end for**

4 The EDENP algorithm

This section starts with the mathematical model of EDENP followed by the detailed description of EDENP. The execution process and resources needed are discussed at last.

4.1 Mathematical model of EDENP

From section 3, we know that the GBED algorithm contains four steps for a certain pixel in an image, i.e., directional gradients estimation, absolute gradient magnitude estimation, comparison between gradient magnitude and predefined threshold, and edge determination, as illustrated in Fig. 1. The four steps should be executed in a certain logical order under the control of enzyme variables. At each step, all pixels on the image are examined in parallel.

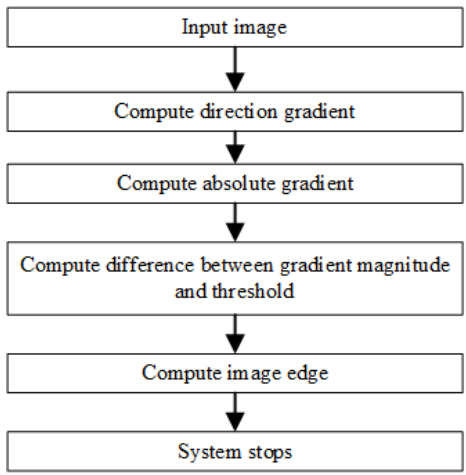


Fig. 1. The computing flowchart of GBED

The mathematical expression of EDENP is as follows, and the corresponding membrane structure is illustrated in Fig. 2.

$$\Pi = \Big(m, H, \mu, \big(Var_1, E_1, \mathrm{Pr}_1, Var_1(0)\big), \big(Var_2, E_2, \mathrm{Pr}_2, Var_2(0)\big)\Big),$$

where

(1) $m = 2$.
(2) $H = \{1, 2\}$.
(3) $u = [[\ ]_2]_1$.
(4) $Var_1 = \{x_{i,j}, edg_{i,j}, \theta, e_{1,1}, E_D\}, Var_2 = \{g_{x_{i,j}}, g_{y_{i,j}}, g_{i,j}, ed_1, ed_2, E_{i,j}, E_{D_{i,j}}\}$.
   (a) $x_{i,j}(1 \le i, j \le n)$, are the gray value of pixel with coordinate of (i, j) on the source image plane.
   (b) $edg_{i,j}(1 \le i, j \le n)$, are the corresponding edge points of the source image with initial value 0.
   (c) $\theta[threshold]$, is a numerical variable which is used as the threshold value for edge detection, and the value of threshold should be predefined.

(d) $g_{x_{i,j}}(1 \leq i,j \leq n)$, are the horizontal derivative approximations at each pixel.

(e) $g_{y_{i,j}}(1 \leq i,j \leq n)$, are the vertical derivative approximations at each pixel.

(f) $g_{i,j}(1 \leq i,j \leq n)$, are the gradient magnitude approximations at each pixel.

(g) $ed_1[0]$, is a numerical variable with initial value 0, which is used as the background value of the edge image.

(h) $ed_2[1]$, is a numerical variable with initial value 1, which is used as the edge point value of the edge image.

(i) $ed_3[-256]$, is a numerical variable with initial value -256, which is used as a intermediate variable.

(5) E_k is a set of enzyme variables from membrane k, i.e., $E_1 = \{e_{1,1}, E_D\}$, $E_2 = \{E_{i,j}, E_{D_{i,j}}\}$.

(6) Pr_k is the set of programs (rules) in membrane k, composed of a production function and a repartition protocol, i.e., $\mathrm{Pr}_1 = \{\mathrm{Pr}_{main}\}$, $\mathrm{Pr}_2 = \{\mathrm{Pr}_{1\sim 14, CE_{i,j}}\}$.

(a) $\mathrm{Pr}_{1,CE_{i,j}} \sim \mathrm{Pr}_{10,CE_{i,j}}$.

(i) $\mathrm{Pr}_{1,CE_{i,j}}$:
$(|x_{i,j+2} + 2x_{i+1,j+2} + x_{i+2,j+2} - x_{i,j} - 2x_{i+1,j} - x_{i+2,j}|)\,|_{e_{1,1}} \rightarrow 1|g_{x_{i,j}}, (2 \leq i,j \leq n-2)$,

(ii) $\mathrm{Pr}_{2,CE_{i,j}}$:
$(|x_{i,j} + 2x_{i,j+1} + x_{i,j+2} - x_{i+2,j} - 2x_{i+2,j+1} - x_{i+2,j+2}|)\,|_{e_{1,1}} \rightarrow 1|g_{y_{i,j}}$,
$(2 \leq i,j \leq n-2)$,

(iii) $\mathrm{Pr}_{3,CE_{1,i}} : 0|_{e_{1,1}} \rightarrow |g_{x_{1,i}}; (1 \leq i \leq n)$,

(iv) $\mathrm{Pr}_{4,CE_{n,i}} : 0|_{e_{1,1}} \rightarrow |g_{x_{n,i}}; (1 \leq i \leq n)$,

(v) $\mathrm{Pr}_{5,CE_{i,1}} : 0|_{e_{1,1}} \rightarrow |g_{x_{i,1}}; (2 \leq i \leq n-1)$,

(vi) $\mathrm{Pr}_{6,CE_{i,n}} : 0|_{e_{1,1}} \rightarrow |g_{x_{i,n}}; (2 \leq i \leq n-1)$,

(vii) $\mathrm{Pr}_{7,CE_{1,i}} : 0|_{e_{1,1}} \rightarrow |g_{y_{1,i}}; (1 \leq i \leq n)$,

(viii) $\mathrm{Pr}_{8,CE_{n,i}} : 0|_{e_{1,1}} \rightarrow |g_{y_{n,i}}; (1 \leq i \leq n)$,

(ix) $\mathrm{Pr}_{9,CE_{i,1}} : 0|_{e_{1,1}} \rightarrow |g_{y_{i,1}}; (2 \leq i \leq n-1)$,

(x) $\mathrm{Pr}_{10,CE_{i,n}} : 0|_{e_{1,1}} \rightarrow |g_{y_{i,n}}; (2 \leq i \leq n-1)$.

Those rules are used to execute formula (1). The enzyme in $\mathrm{Pr}_{1,CE_{i,j}} \sim \mathrm{Pr}_{10,CE_{i,j}}$ must exist in enough amount so that the rules can be activated. Specifically, if the value of the enzyme $e_{1,1}$ is greater than variable $x_{i,j}(1 \leq i,j \leq n)$, then rules $\mathrm{Pr}_{1,CE_{i,j}} \sim \mathrm{Pr}_{10,CE_{i,j}}$ are effective. Since variable $x_{i,j}$ is the gray value of image, the maximum value is 255. So, the initial value of $e_{1,1}$ is set to 256, such that the condition modeled by rule $\mathrm{Pr}_{1,CE_{i,j}} \sim \mathrm{Pr}_{10,CE_{i,j}}$ are satisfied. It is important to note that the number of rules are $n \times n$, and all the rules are executed in parallel.

(b) $\mathrm{Pr}_{11,CE_{i,j}} : \left(\sqrt{g_{x_{i,j}}^2 + g_{y_{i,j}}^2}\right)|e_{1,1} \rightarrow 1|g_{i,j}; 1 \leq i,j \leq n$

$\mathrm{Pr}_{11,CE_{i,j}}$ are the rules which are executed by formula (5). Hence, after executing $\mathrm{Pr}_{1,CE_{i,j}} \sim \mathrm{Pr}_{10,CE_{i,j}}$, the value of the variables $g_{x_{i,j}}$, $g_{y_{i,j}}$ are obtained. The maximum value of $g_{x_{i,j}}$ and $g_{y_{i,j}}$ is 255, and the enzyme $e_{1,1}$ is 256. So the condition of execution for rules $\mathrm{Pr}_{11,CE_{i,j}}$ is satisfied. Hence, all $n \times n$ rules are executed concurrently.

(c) $\mathrm{Pr}_{12,CE_{i,j}} : (2*(\mathrm{g_{i,j}} - \theta))| \rightarrow 1|\mathrm{g}_{i,j}+1|\mathrm{E}_{i,j}; 1 \leq i,j \leq n$
$\mathrm{Pr}_{12,CE_{i,j}}$ are the rules which compute $d_{i,j}$ in formula (6). After executing $\mathrm{Pr}_{12,CE_{i,j}}$, the value of $d_{i,j}$ are obtained, which is equal to variables $g_{i,j}$ and $E_{i,j}$ in rule $\mathrm{Pr}_{12,CE_{i,j}}$.

(d) $\mathrm{Pr}_{13,CE_{i,j}} : (ed_1 + 2 * ed_2)\,|_{\mathrm{E}_{i,j}} \rightarrow 1|\mathrm{edg}_{i,j}+1|E_{D_{i,j}};\ \mathrm{Pr}_{14,CE_{i,j}} : (0 * ed_1 + 0 * ed_3|_{\mathrm{E}_{i,j}} \rightarrow 1|\mathrm{edg}_{i,j} + 1|E_{D_{i,j}};\ 1 \leq i,j \leq n.$
$\mathrm{Pr}_{13,CE_{i,j}}$ and $\mathrm{Pr}_{14,CE_{i,j}}$ are rules for computing edge value as formula (7). If $\mathrm{E}_{i,j}$ is greater than or equal to 0, then $\mathrm{Pr}_{13,CE_{i,j}}$ and $\mathrm{Pr}_{14,CE_{i,j}}$ are executed. Because ed_1 is 0, and ed_3 is -256, so $\mathrm{E}_{i,j} \geq \min(ed_1, ed_2)$ and $\mathrm{E}_{i,j} \geq \min(ed_1, ed_3)$. The execution condition of $\mathrm{Pr}_{13,CE_{i,j}}$ and $\mathrm{Pr}_{14,CE_{i,j}}$ is satisfied. If $\mathrm{d}_{i,j} < 0$, only $\mathrm{Pr}_{14,CE_{i,j}}$ will be executed. Because $\mathrm{E}_{i,j} \geq \min(ed_1, ed_3)$ and $\mathrm{E}_{i,j} < \min(ed_1, ed_2)$, only the execution condition of $\mathrm{Pr}_{14,CE_{i,j}}$ can be satisfied. After executing $\mathrm{Pr}_{13,CE_{i,j}}$ and $\mathrm{Pr}_{14,CE_{i,j}}$, variables $edg_{i,j}$ will be set to 1 if $\mathrm{d}_{i,j} \geq 0$ and every variable $E_{D_{i,j}}$ will be assigned.

(e) $\mathrm{Pr}_{main} : \left(0 * E_{D_{1,1}} + 0 * E_{D_{1,2}} + ... + 0 * E_{D_{m,n}} + 1\right) | \rightarrow 1|E_D$
Pr_{main} is a rule contained in cell 1, which controls the stop condition of the P system. For pixel (i,j) , if all the enzyme variables $E_{D_{i,j}}$ are assigned, the condition for Pr_{main} is meet. Enzyme variable E_D is set to 1 by rule Pr_{main}, and the system stops running.

4.2 The structure and execution process of EDENP

As shown in Fig. 2, the structure of EDENP includes two membranes. The system begins to work when the input variables $x_{i,j}$ represent the gray value of source image at location (i,j) appear in the skin membrane. The whole process includes five steps.

Step 1: Horizontal and vertical derivative approximations of every pixel are computed by using rules of $\mathrm{Pr}_{1,CE_{i,j}}$~$\mathrm{Pr}_{10,CE_{i,j}}$ in a parallel manner;

Step 2: The gradient magnitude of all the pixels are obtained at the same time with rules of $\mathrm{Pr}_{11,CE_{i,j}}$;

Step 3: The comparison between the gradient magnitude of each pixel and the predefined threshold is executed by rules of $\mathrm{Pr}_{12,CE_{i,j}}$;

Step 4: The edge pixels are detected and marked with 1, while the background pixels are marked with 0 by rules of $\mathrm{Pr}_{13,CE_{i,j}}$ and $\mathrm{Pr}_{14,CE_{i,j}}$;

Step 5: The system stop condition is satisfied and the system stops working by rules of Pr_{main}.

So as described above, only five steps are needed to the proposed algorithm for images with arbitrary resolution. Since we did not change the mathematical model, the performance of the algorithm is equal to the original Sobel operator based edge detection algorithm in image processing.

4.3 Complexity and resources analysis

Taking into account that the size of the input data is $n \times n$, and the image is a gray image. The amount of resources needed is illustrated in Table 1. From Table 1, we can see that the time complexity of the proposed method is $O(1)$, and the space complexity is $O(n^2)$.

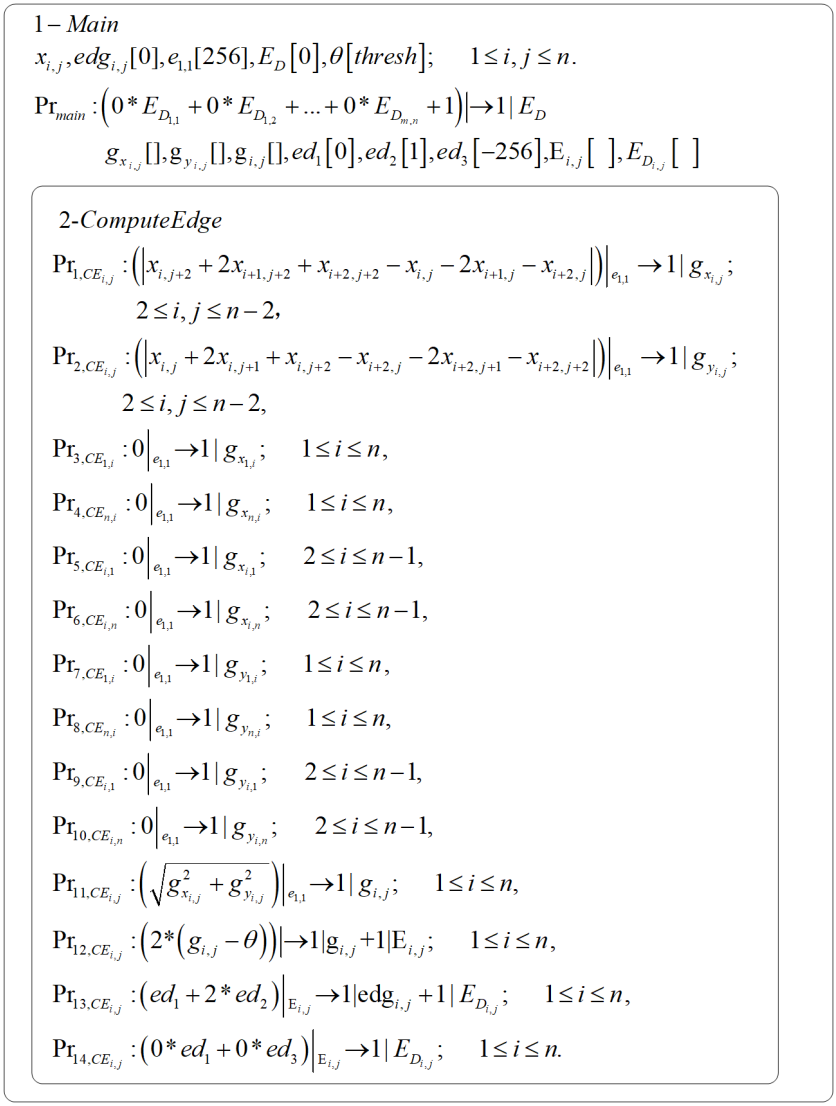


Fig. 2. Enzymatic numerical membrane system for GBED

5 Experiments and Results

In this section, both the performance and efficiency of our proposed EDENP algorithm are evaluated. Since there is no hardware implementation of membrane computing systems at present, the only way to test the behavior of designed P systems is to simulate it in conventional computers. In this paper, a parallel computing architecture, CUDA (Compute Unified Device Architecture), is used as the simulating platform, as it has

been reported in several literature [47,48,49,50]. The parameters of the platform on which our experiments are carried out are illustrated in Table 2. The threshold θ for all the experiments is set to 0.2.

5.1 Performance evaluation

Two case studies are considered to evaluate the performance of the proposed method. For each case study, the comparison is considered in terms of quantitative and qualitative results with the typical methods proposed in [27] and [50]. We will sketched the edge detection algorithm of literature [27,50] and using the MATLAB program to implement it on a CPU platform. The confidence degree of the edge image is one of the mostly used evaluation indicator for edge detection algorithms, and its mathematical definition is presented in reference [51]. In general, the greater edge confidence degree is, the more reliable the edges are. In this article, edge confidence degree is introduced to evaluate the performance of the edge detection algorithm quantitively.

5.1.1 Case study 1 Case study 1 is considered to evaluate the performance of the three algorithms for images with rich textures. Four images named *rice*, *cameraman*, *mri*, and *AT3_lm4_01* randomly collected from the MATLAB Image Tool Box are used as testing samples in this experiment, as shown in Fig.4(a1∼a4). Fig.4(b1∼b4) , (c1∼c4) and (d1∼d4) show the detailed qualitative edge detection results of the three algorithms for the four images. It can be clear observed from Fig.4(b1∼b4) that the counts of the objects can be detected, but meanwhile the noise on the background is also detected, which will make the following image processing, such as object recognition, more difficult to deal with. The results by reference [27] are shown in Fig.4(c1∼c4). It can be seen that there are too many small edges, and the main outlines of the targets can hardly be found even by human eyes. The results of EDENP are illustrated in Fig.4(d1∼d4), from which we can see that not only the main counters of objects can be detected successfully, but also the noise is well suppressed.

5.1.2 Case study 2 Case study 2 is used to test the performance of the three methods for images with less texture, in which images named *toyobjects*, *circbw*, *text*, *testpat1* randomly selected from MATLAB Image Tool Box are used as testing image samples. In image *toyobjects*, each object has a constant gray value, while the other three images are binary images. Like in Case 1, the detected edge results by the three approaches are shown in Fig.5 (b1∼b4), (c1∼ c4) and (d1∼d4). Fig.5(b1),(b4) clear show that there are many discontinuous edges when using algorithm in reference [50], while the other two methods can detect the edges completely. When comparing the thickness of the edges, it is obvious to see that method in reference [27] can achieve the thinnest edges, then the EDENP method, and the edges detected by [50] is the thickest. Although the method in [27] can obtain the finest edges, those edges often have burrs, as shown in Fig.6. Fig.6 gives the enlargement of partial edges, where the red rectangular in Fig.6 (a1) and (a2) are the areas zoomed in. The corresponding enlarged edges are presented in Fig.6 (c1) and (c2), where areas containing burrs are marked with pink rectangle. By comparing Fig.6 (c1),(c2) with (b1),(b2) and (d1),(d2), it is clear that edges by algorithm in [50]

Table 1. Complexity and resources needed for proposed P system

Term	Necessary Resources
Initial number of cells	2
Number of enzymatic variables	$2n^2 + 2$
Number of numerical variables	$5n^2 + 4$
Number of rules	$6n^2 + 4$
Execution steps	5

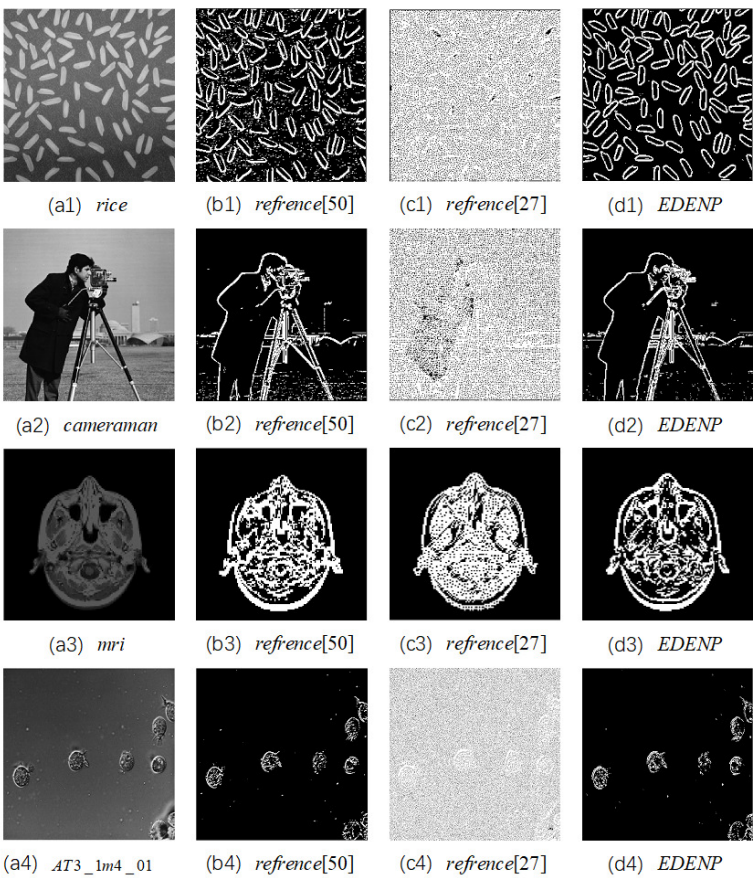


Fig. 3. Edge detection results of images with rich texture (the first column: the source gray images; the second to the last column: results by using methods in reference [50], reference [27] and EDENP respectively)

Table 2. Parameters the computer used

Term	Parameters
CPU model	Intel Pentium 4 650
cache memory	2 MB of L2
main memory	1 GB DDR SDRAM with 64 bits bus wide to 200 MHz
hard disc	160 GB SATA2 with a transfer rate of 300 Mbps in a 8 MB buffer
GPU model	NVIDIA Geforce 8600 GT composed by 4 Stream
execution steps	5

Table 3. The edge confidence degree

	reference [50]	reference [27]	EDENP
rice	0.75	0.56	**0.84**
cameraman	0.66	0.32	**0.74**
mri	0.63	0.56	**0.68**
AT3_lm4_01	0.44	0.12	**0.5**
toyobjects	0.85	0.76	**0.86**
circbw	0.94	0.93	**0.95**
text	0.93	0.90	**0.94**
testpat1	0.81	0.79	**0.86**

and EDENP are smoother than that in [27]. The areas where contain discontinuous edges with method in [27] are marked in green rectangle Fig.6 (b1),(b2).

Table.3 provides the comparison results of the three methods in terms of edge confidence degree. Note that the edge confidence degree refers to the credibility. The greater the edge confidence degree, the greater the probability that the detected edges are the true edges. It can be seen from Table.3 that the EDENP method has the highest edge confidence degree for images with both high and less texture, which means edges detected by EDENP has less false edges.

Through the above quantitative and qualitative results , it can be deduced that method in reference [27] is nearly invalid for grayscale images with rich texture. For images with less textures, this method can get the fine edges of the objects. However, the edges are not smooth in some cases because of the false burr edge points. Approach in [50] can not get the whole counters of the objects due to the discontinous edges detected for images with both rich and less textures. The EDENP algorithm has the highest performance and can obtain edges which are clear, continuous, and authentic for images with both rich and less textures.

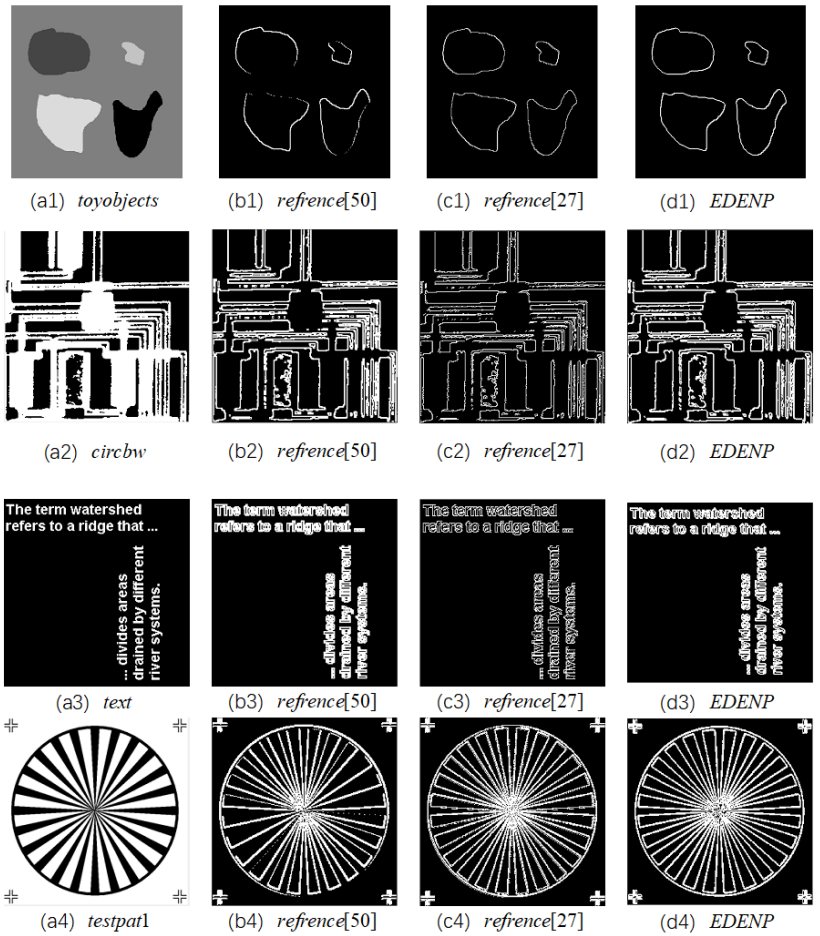


Fig. 4. Edge detection results of images with less texture (the first column: the source gray images; the second to the last columns: results by using methods in [50], [27] and EDENP, respectively)

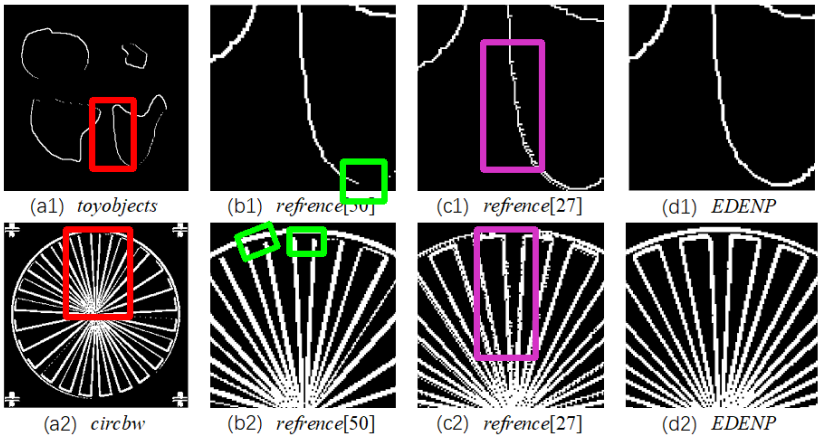


Fig. 5. Edge detection results of images with less texture (the first column: the source gray images; the second to the last columns: results by using methods in [50], [27] and EDENP, respectively)

Table 4. Elapsed of images with different resolution($cameraman$)

Image resolution	256^2	384^2	512^2	768^2	1024^2	2048^2	platform
Elapsed Time(ms)	0.014	0.03	0.05	0.12	0.23	0.86	GPU
Elapsed Time(ms)	3.5	9.1	4.4	9.6	41.9	72.8	CPU
Speedup ratio	250	303	88	80	182	130	

5.2 Efficiency evaluation

To better describe the computation performance of EDENP, a speedup ratio is defined as the elapsed time of algorithm on CPU platform divided by running time on GPU platform. The running times of images with different resolutions under GPU and CPU platform and corresponding speedup ratios for one image are illustrated in Tables 4 and 5 gives the speedup ratios results of the other seven images. From Tables 4 and 5, we can see that the lowest speedup is 53, and the maximum speedup can reach up to 262. It is obvious that the computing power of the proposed algorithm is much superior comparing with the traditional algorithm implemented on CPU platform.

Table 5. The speedup ratio of seven images

Image resolution	256^2	384^2	512^2	768^2	1024^2	2048^2
mri	60	80	77	62	73	66
AT3_lm4_01	80	90	102	172	71	75
circbw	193	213	262	210	176	66
text	167	180	194	118	57	65
testpat1	53	76	100	161	87	64

6 Conclusions

In this paper, a novel GBED parallel implementation method under the membrane computing framework named EDENP has been presented. The core idea of this method is to employ enzyme variables in ENPS to flexibly control the execution order of the rules. So the edge detection based on Gradient operator can be successfully designed. It only needs 5 steps for edge detection regardless of how large the image resolution is. Extensive simulation experiments conducted with different image resolutions show the good performance and effectiveness of the algorithm proposed. Since the gradient operator is a typical linear operator, the algorithm proposed can be extended to other linear convolution calculations in image processing, like mean filter algorithm, opening and closing operations in morphology.

Aknowledgement

This work was partially supported by the National Natural Science Foundation of China (61672437, 61702428), the Sichuan Science and Technology Program China(2018GZ0385, 2017GZ0431, 2018GZ0245, 2018GZ0185, 2018GZ0086), Sichuan education department Program China(17ZB0095). Talent Import Fund of Chengdu University of Information Technology China (KYTZ201633), and New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0044).

References

1. Preet. Kaur, R.: Survey on document image processing. International Journal of Computer Trends and Technology, 15(1), 26–29 (2014).
2. Zheng, S., Yuille, A., Tu, Z.: Detecting object boundaries using low-, mid-, and high-level information. Computer Vision and Image Understanding, 114(10), 1055–1067 (2010).
3. Zhao, L., Zhao, Q., Liu, H., Lv, P., Gu, D.: Structural sparse representation-based semi-supervised learning and edge detection proposal for visual tracking. Visual Computer, 33(9), 1169–1184 (2017).
4. Belongie, S., Malik, J., Puzicha, J.: Shape matching and object recognition using shape contexts. Proceedings of the International Conference on Computer Science and Information Technology, 483-507 (2010).
5. Hua, S., Wen, H.: Moment-preserving edge detection and its application to image data compression. Optical Engineering, 32(7), 1596–1608 (2013).
6. Shin, M., Goldgof, D., Bowyer, K.: Comparison of edge detectors using an object recognition task. Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition, 1596–1608 (1999).
7. Ummer, S., Sreekumar, R.: Real time implementation of edge detection technique for angiogram images on FPGA. International Research Journal of Engineering and Technology, 4(6), 1383–1386 (2017).
8. Saif, J., Hammad, M., Alqubati, I.: Gradient based image edge detection. International Journal of Engineering and Technology, 8(3), 153–156 (2016).
9. Shah, M. Performance analysis of Sobel edge detection filter on GPU using CUDA & OpenGL. International Research Journal of Engineering and Technology, 4(6), 1383–1386 (2017).

10. Jiang B.: Real-time multi-resolution edge detection with pattern analysis on graphics processing unit. Journal of Real-Time Image Processing, 14(2), 293–321, (2018).
11. Zuo H., Zhang Q., Yong, X., Zhao, R.: Fast Sobel edge detection algorithm based on GPU. Opto-Electronic Engineering, 36(1), 8–12 (2009).
12. Nickolls, J., Dally, W.: The GPU computing era, IEEE Micro, 30(2), 56–69 (2010).
13. Stojčev,. M, Tokić, T., Milentijević, I.: The limits of semiconductor technology and oncoming challenges in computer microarchitectures and architectures. Facta Universitatis-Series: Electronics and Energetics, 17(3), 285–312 (2004).
14. Venegas-Andraca, S., Bose, S.: Quantum computation and image processing: new trends in artificial intelligence. Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, 1563–1572 (2003).
15. Caraiman, S., Manta, V.: Image processing using quantum computing, Proceedings of International Conference on System Theory, Control and Computing, 1–6 (2012).
16. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences, 61(1), 108–143 (2000).
17. Păun, Gh.: Membrane computing. An introduction. Springer-Verlag, Berlin. (2002)
18. Păun, Gh., Rozenberg, G., Salomaa, A, Eds.: The Oxford Handbook of membrane computing. Oxford University Press, New York (2010)
19. Alsalibi, B., Venkat, .I, Subramanian, K., Lebai, L., De, W.: The impact of bio-inspired approaches toward the advancement of face recognition. ACM Computing Surveys, 48(1), 1–33 (2015)
20. Yahya, R., Shamsuddin, S., Yahya, S., Hasan, S., AI-Salibi, B., AI-Khafaji, G.: Image segmentation using Membrane Computing: a literature survey. Proceeding of International Conference on Bio-Inspired Computing-Theories and Applications, 314–335 (2016).
21. Alsalibi, B., Venkat, I., Subramanian, K., Christinal, H.: A bio-inspired software for homology groups of 2D digital images. Proceedings of Asian Conference on Membrane Computing (ACMC), 1–4 (2014).
22. Díaz-Pernil, D., Christinal, H., Gutiérrez-Naranjo, M., Real, P.: Using membrane computing for effective homology. Applicable Algebra in Engineering, Communication and Computing, 23(5-6), 233–249 (2012).
23. Ardelean, I., Díaz-Pernil, D., Gutiérrez-Naranjo, M., Peña-Cantillana, F., Reina-Molina, R., Sarchizian, I.: Counting cells with tissue-like P systems. Proceedings of the Tenth Brainstorming Week on Membrane Computing, 69–78 (2012).
24. Reina-Molina, R., Díaz-Pernil, D., Gutiérrez-Naranjo, M. Cell complexes and membrane computing for thinning 2D and 3D images. Proceedings of the Tenth Brainstorming Week on Membrane Computing, 167–186 (2015).
25. Gimel, F., Nicolescu, R., Ragvan, S.: P systems in stereo matching. Proceedings of the International Conference on Computer Analysis of Images and Patterns, 285-292, (2011).
26. Berciano, A.,Díaz-Pernil, D., Christinal, H., Venkat, I.: First steps for a corner detection using membrane computing. Proceedings of the Asian Conference on Membrane Computing, 1–6 (2014).
27. Christinalh, A., Díaz-Pernil, D., Real, P.: Region-based segmentation of 2D and 3D images with tissue-like P systems. Pattern Recognition Letters, 32(16), 2206–2212 (2011).
28. Díaz-Pernil, D., Gutiérrez-Naranjo, M., Molina-Abril, H., Real, P.: Designing a new software tool for digital imagery based on P systems. Natural Computing, 11(3), 381–386, (2012).
29. Carnero, J., Díaz-Pernil, D., Molina-Abril, H., Real, P.: Image segmentation inspired by cellular models using hardware programming. IMAGE-A, 1(3), 143–150 (2010).
30. Peña-Cantillana, F., Díaz-Pernil, D., Christinal, H., Gutiérrez-Naranjo, M.: Implementation on CUDA of the Smoothing Problem with Tissue-Like P Systems. International Journal of Natural Computing Research, 2(3), 25–34 (2011).

31. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science, 296(2), 295–326 (2003).
32. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae, 71(2-3), 279–308 (2006).
33. Păun, Gh., Păun, R.: Membrane Computing and economics: numerical P systems. Fundamenta Informaticae, 73(1-2), 213–227 (2006).
34. Buiu, C., Vasile, C., Arsene, O.: Development of membrane controllers for mobile robots. Information Sciences, 187(1), 33–51, (2012).
35. 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(1), 33–51 (2016).
36. Zhang, G., Gheorghe, M., Pérez-Jiminez, M.J.: Real-life Applications with Membrane Computing. Springer (2017)
37. Mahalingam, K., Rama, R., Sureshkumar, W., Robot motion planning inside a grid using membrane computing. International Journal of Imaging and Robotics, 17(1), 33–51 (2017).
38. Pavel, A., Arsene, O., Buiu, C.: Enzymatic numerical P systems - a new class of membrane computing systems. Proceedings of the Fifth International Conference on Bio-Inspired Computing: Theories and Applications, 1331–1336 (2010).
39. Vasile, C., Pavel, A., Dumitrache, I., Păun, Gh.: On the power of enzymatic numerical P system. Acta Informatica, 49(6), 395–412 (2012).
40. Vasile, C., Pavel, A., Dumitrache, I.: Universality of enzymatic numerical P systems. International Journal of Computer Mathematics, 90(4), 869–879 (2013).
41. Leporati, A., Porreca, A., Zandron, C., Mauri, G.: Improving universality results for parallel enzymatic numerical P systems. International Journal of Unconventional Computing, 9(5-6), 385–404, (2013).
42. Pavel, A., Buiu, C.: Using enzymatic numerical P systems for modeling mobile robot controllers. Natural Computing, 11(3), 387–393 (2012).
43. Pavel, A., Vasile, C., Dumitrache, I.: Robot localization implemented with enzymatic numerical P systems. Proceeding of Conference on Biomimetic and Biohybrid Systems, 204–215 (2012).
44. Li, W., Yang, J., Zhang, J.: Handling big data field with enzymatic numerical P System. Journal of Sichuan University, 45(6), 96–104 (2013)
45. Llorente-Rivera, D., Gutiérrez-Naranjo, M.: The pole balancing problem with enzymatic numerical P Systems. Proceedings of the Thirteenth Brainstorming Week on Membrane Computing, 195–206 (2015).
46. Maeda, S., Fujiwara, A.: Enzymatic numerical P systems for basic operations and sorting, Proceedings of the International Symposium on Soft Computing and Intelligent Systems, 1333–1338 (2014).
47. Cecilia, J., García, J., Guerrero, G., Martínet-del-Amor, M., Pérez-Hurtado, I., Pérez-Jiménez, M.: Simulation of P systems with active membranes on CUDA. Briefings in Bioinformatics, 11(3), 313-322 (2009).
48. Cabarle, F., Adorna, H., Martinez, M.: A spiking neural P system simulator based on CUDA. Proceedings of the 12th International Conference on Membrane Computing, 87–103 (2011).
49. Maroosi, A., Muniyandi, R., Sundararajan, E., Zin, A., Improved implementation of simulation for Membrane Computing on the graphic processing unit. Proceedings of the 4th International Conference on Electrical Engineering and Informatics, 184–190 (2013).
50. Díaz-Pernl, D., Berciano, A., Peňa-Cantillana, F., Gutiérrez-Naranjo, M.: Segmenting images with gradient-based edge detection using Membrane Computing. Pattern Recognition Letters, 34(8). 846–855 (2013).

51. Wang, J., Bi, J., Wang, L., Wang, X.: A non-reference evaluation method for edge detection of wear particles in ferrograph images. Mechanical Systems and Signal Processing, 863-876 (2018).
52. Zhang, Z., Wu, T., Paun, A., Pan, L.: Numerical P systems with migrating variables, Theoretical Computer Science, 641: 85-108, (2016).
53. Zhang, Z., Wu, T., Paun, A., Pan, L.: Universal enzymatic numerical P systems with small number of enzymatic variables, Science China Information Sciences, DOI:10.1007/s11432-017-9103-5, (2018).
54. Pan, L., Zhang, Z., Wu, T., Xu, J.: Numerical P systems with production thresholds, Theoretical Computer Science, 673: 30-41, (2017).
55. Zhang, Z., Pan, L.: Numerical P systems with thresholds. International Journal of Computers Communications Control, 11(2): 292-304, (2016)

P-Lingua Compiler: A Tool for Generating Ad-hoc Simulators in Membrane Computing

Ignacio Pérez-Hurtado[1], David Orellana-Martín[1],
Gexiang Zhang[2], and Mario J. Pérez-Jiménez[1]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville, Seville, Spain
{perezh,dorellana,marper}@us.es
[2] School of Electrical Engineering
Southwest Jiaotong University, Chengdu, Shichuan, China
zhgxdylan@126.com

Abstract. Since the beginnings of membrane computing, software and hardware tools have been implemented for simulating computations of the proposed models. Some of these simulators are relatively generic, providing enough flexibility for a wide variety of models and others are ad-hoc simulators that reproduce computations of a single design that has been hard-coded or computations of a single type of model. On the one hand, generic tools are excellent assistants for the researchers while verifying their designs. On the other hand, the efficiency of specific tools in terms of simulation performance for a given design sacrifices the flexibility of the previous ones. In this paper, it is presented for the first time a tool that breaks this duality, we have implemented a compiler which receives as input the definition of a design in the P-Lingua language and produces as output source code in the C++ language for an ad-hoc simulator that has been optimized for the input design. The objective of this work is twofold: On the one hand, we have extended the P-Lingua framework to include some semantic features concerning to the models, such as rule patterns and derivation modes, that can be written in an explicit manner within their own file. On the other hand, we have developed a GNU GPLv3 command-line tool for Linux which works in the same manner as conventional compilers. Finally, we include in this paper a few examples for different types of cell-like and tissue-like models.

1 Introduction

Membrane computing is an unconventional model of computation within natural computing that was introduced in 1998 by Gh. Păun [17]. The computational devices in membrane computing, also known as P systems, are non-deterministic theoretical machines inspired on the biochemical processes that take place inside the compartments of living cells.

Several kinds of P systems coexist, each of them having different syntactic ingredients, such as different alphabets and structures. The two most studied are

cell-like membrane systems, characterized by their rooted tree structure, where membranes act as filters that let certain elements to pass through them [17], and membrane systems structured as directed graphs, representing the communication between cells within a tissue of a living being, called tissue-like P systems [9] or between neurons in a brain, called spiking neural P systems [7]. The interchange of objects between the different *compartments* is defined by the *rules* of the system, that together with the corresponding semantics, mark the functioning of the system.

A *configuration* of a P system is defined by the structure of the compartments at a certain moment, and the elements (being usually objects, although other kinds of elements can be considered, as strings, catalysts [17] and antimatter [14], among others) contained in each compartment, as well as other characteristics from specific types of P systems, providing a *snapshot* of the system at an instant t. By using the rules specified in a model, we can make its objects change, both evolving and moving between the different compartments (membranes in the case of cell-like P systems and cells in the case of tissue-like P systems).

On the one hand, in P systems with active membranes [19], both objects and membranes change through the application of evolution, communication, division, separation, creation and dissolution. In this framework, membranes can have a polarization associated to each membrane. On the other hand, in tissue P systems [9], symport/antiport rules are devoted to make objects move from a cell to another cell or to the *environment* (a special compartment where there exist an arbitrary number of objects of an alphabet defined *a priori*), while division and separation rules allow an exponential growth in linear time.

We say that a configuration C_t yields to a configuration C_{t+1} if, by applying the rules specified in the model according to its semantics, we can obtain C_{t+1} from C_t. Semantics rules the behavior of the system, determining which rules can be applied and how they affect the system according to a global clock. A *computation* of a P system is a (finite or infinite) sequence of instantaneous configurations.

We consider a *family* (or *model*) of P systems as the definition of a type of P system, that is, its syntax and semantics. According to the specification of a particular family of P systems, we consider a (specific) *model* as the definition of an individual P system, that is, its working alphabet, initial membrane structure with initial multisets of objects and the set of rewriting rules with another characteristics of the correspondent family. By the definition of the family, we can interpret the structure and behavior of a specific model within that family.

Membrane computing is a very flexible framework where different types of devices can be outlined. In fact, the intersection between Membrane Computing and other fields, such as engineering [20], biology [23] and ecology [2], as well as a long list of other scientific lines [5, 13, 24], has generated necessities that could only be filled by the creation of new kinds of P systems, expanding the scope of researchers in this area. For an exhaustive explanation of the different types of P systems, we refer the reader to [18] and [16].

Software and hardware simulators have been implemented from the beginnings of membrane computing. Some of these are very generic and flexible. On the other hand, we define an *ad-hoc* simulator as a simulator for one and only one membrane computing design which has been hard-coded. Such tools are usually the faster simulators since they can be optimized for the input design and the hardware to be used. But the hard-coding process requires an excellent knowledge of the hardware architecture, as well as the design to be implemented. Debugging should be always critical and the results are not very reusable.

In this work, we have extended the P-Lingua framework [6, 25] to include semantic features concerning to the models. On the other hand, we have implemented a GNU GPLv3 command-line tool to compile P-Lingua input files to ad-hoc source code in C++. The output files are optimized for the input designs and all the process can be automatized by using *makefiles*, i.e.,files which specify how to derive the target program.

The paper is structured as follows: In the next section, some preliminaries concepts about P-Lingua are introduced. In Section 3, we propose an extension for the P-Lingua language to directly define model constraints in the own P-Lingua files, providing a more flexible and experimental framework. The next Section is devoted to the new GNU GPLv3 software tool to compile the input P-Lingua files and generate source code in C++, as well as JSON code codifying the input designs for third-party applications. Section 5 introduces the simulation algorithm used in the generated simulators. In Section 6 some examples of the new P-Lingua extension are introduced. Finally, some conclusions and future work are drawn.

2 Preliminaries

P-Lingua [6, 25] is a software framework that includes a definition language for P systems (also called P-Lingua) and a GNU GPLv3 Java library (pLinguaCore) that is able to parse P-Lingua files and simulate computations. The library contains three main components:

- A parser for reading input files in P-Lingua format and checking syntactic and semantic constraints related to predefined models. In order to achieve this, the first line of a P-Lingua file should include a P system model declaration by using an unique identifier. There are several P system models that can be used, each one with its own identifier, such as `transition`, `membrane_division`, `tissue_psystems`, and `probabilistic`. The analysis of semantic ingredients, such as rule patterns, is hard-coded for each model. Several versions of pLinguaCore [6, 8, 10, 21] have been launched to cover different types of models.
- For each type of model, the pLinguaCore library includes one or more built-in simulators, each one implementing a different simulation algorithm. For instance, Population Dynamic P systems [1] (`probabilistic` identifier in P-Lingua) can be simulated within the library by applying three different

algorithms: `BBB`, `DNDP`, and `DCBA` [3, 11]. Remarkable software projects such as MeCoSim (Membrane Computing Simulator) [27, 22] use the simulators integrated in the library to perform P system computations and generate relevant information as result for custom applications.
- Alternatively, the pLinguaCore library is able to transform the input P-Lingua files to other formats such as XML or binary format in order to feed external simulators. The generated files for the given P systems are free of syntactic/semantic errors since the transformation is done after the parser analysis. Several external simulators use this feature, for example, the PMCGPU project (Parallel simulators for membrane computing on GPU) [12, 26] uses definitions generated by pLinguaCore in order to provide the input of CUDA GPU simulators.

The P-Lingua language is currently a standard widely used for the scientific community since the syntax is modular, parametric and close to the common scientific notation. The description of the language can be found in the references [6, 8, 10, 21, 25]. As an example, the definition of a basic transition P system follows:

```
@model<transition>
def main()
{
  @mu = [[[]'3 []'4]'2]'1;
  @ms(3) = a,f;

  [a --> a,bp]'3;
  [a --> bp,@d]'3;
  [f --> f*2]'3;

  [bp --> b]'2;
  [b []'4 --> b [c]'4]'2;
  (1) [f*2 --> f ]'2;
  (2) [f --> a,@d]'2;
}
```

In the example, a module `main` is defined including an initial membrane structure $[\,[\quad]_3\,[\quad]_4\,]_2\,]_1$, an initial multiset for the membrane labelled 3, and a set of seven multiset rewriting rules. The special symbol `@d` is used to specify dissolution. The last two rules include priorities as integer numbers in parenthesis at the beginning of the left-hand side of the rules. More complex examples can be found in the P-Lingua web [25].

3 An extension of P-Lingua for semantic features

As explained above, the analysis of semantic ingredients belonging to P systems is hard-coded in the pLinguaCore library, i.e, the only way to define new types of

models is by implementing code inside the library. In this section, we propose an extension for the P-Lingua language to directly define model constraints in the own P-Lingua files, providing a more flexible and experimental framework. Two types of semantic constraints can be defined with this extension: *rule patterns* and *derivation modes*.

3.1 Rule patterns

The P-Lingua parser is able to recognize rules with the next general syntax:

$$p$$

$$u[v_1[v_{1,1}]^{\alpha_{1,1}}_{h_{1,1}} \dots [v_{1,m_1}]^{\alpha_{1,m_1}}_{h_{1,m_1}}]^{\alpha_1}_{h_1} \dots [v_n[v_{n,1}]^{\alpha_{n,1}}_{h_{n,1}} \dots [v_{n,m_n}]^{\alpha_{n,m_n}}_{h_{n,m_n}}]^{\alpha_n}_{h_n}$$

$$\xrightarrow{q} \text{ or } \xleftrightarrow{q}$$

$$w_0[w_1[w_{1,1}]^{\beta_{1,1}}_{g_{1,1}} \dots [w_{1,r_1}]^{\beta_{1,r_1}}_{g_{1,r_1}}]^{\beta_1}_{g_1} \dots [w_s[w_{s,1}]^{\beta_{s,1}}_{g_{s,1}} \dots [w_{s,r_s}]^{\beta_{s,r_s}}_{g_{s,r_s}}]^{\beta_s}_{g_s}$$

where:

- p is a priority related to the rule given by a natural number, where a lower number means a higher rule priority.
- q is a probability related to the rule given by a real number in $[0, 1]$.
- $\alpha_i, \alpha_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m_i$ and $\beta_i, \beta_{i,j}, 1 \leq i \leq s, 1 \leq j \leq r_i$ are electrical charges.
- $h_i, h_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m_i$ and $g_i, g_{i,j}, 1 \leq i \leq s, 1 \leq j \leq r_i$ are membrane labels.
- $u, v_i, v_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m_i$ and $w_i, w_{i,j}, 1 \leq i \leq s, 1 \leq j \leq r_i$ are multisets of objects.

Next, there is a list of P-Lingua rule examples matching the general rule syntax:

- `a,b [ d,e*2 ]'h --> [f,g]'h :: q;` where `q` is the probability of the rule.
- `(p) [a]'h --> [b]'h;` where `p` is the priority of the rule.
- `[a --> b]'h;`, the left-hand side and right-hand side of evolution rules can be collapsed.
- `+[a]'h --> +[b]'h -[c]'h;` a division rule using electrical charges.
- `[a]'h --> ;` a dissolution rule.
- `a[ ]'h --> [b]'h;` a send-in rule.
- `[a]'h --> b[ ]'h;` a send-out rule.
- `[a --> #]'h;` the symbol `#` can be optionally used as empty multiset.
- `[a]'1 <--> [b]'0;` a symport/antiport rule in the tissue-like framework.

The syntax of the general rule is very permissive, and so different parsers for different models have been developed in order to restrict the rules used in each one. In order to provide the researcher a more flexible framework, not having to depend on the implementation itself but acquiring the capacity of restricting the model by himself, we introduce the next syntax in P-Lingua for rule pattern matching:

```
!rule-type-id
{
pattern1
pattern2
...
patternN
}
```

where `rule-type-identifier` is an unique name for the type of rule that is going to be defined and `pattern1`, `pattern2`, ..., `patternN` are rule patterns following the same syntax than common rules in P-Lingua where anonymous variables beginning with `?` can be optionally used instead of probabilities, charges and priorities. In the patterns, the symbols beginning with `a`, `b` or `c` always mean single objects and symbols beginning with `u`, `v` and `w` always mean multisets of objects. In Section 6, are given several examples of rule patterns in P-Lingua for different types of cell-like and tissue-like models.

3.2 Derivation modes

From an informal point of view, we can see a derivation mode as the way a step of a P system is performed. As a part of semantics, it rules the exact application of rules of the system, deciding when rules can be applied or not when they are applicable. An extensive study of derivation modes can be found in [4]. In order to make the work self-content, we give a minimal definition of a derivation mode.

A derivation mode ϑ is defined as a function that selects different multisets of rules "really applicable" to a configuration C_t of a P system depending on a specification. For this purpose, let Π be a P system with $\mathcal{R}$ as its set of rules, R a multiset of compatible rules applicable to a P system at configuration C_t, and let $\mathbf{R}$ be the set of all multisets applicable to a P system at configuration C_t.

In this extension of P-Lingua we provide two main derivation modes:

- **Maximally parallel derivation mode** (max): It is the default mode for P systems. In this mode, we only take multisets from R that are not extensible, that is:

$$\mathbf{R}' = \{R \mid R \in \mathbf{R} \wedge \nexists R' \in \mathbf{R} : R \subsetneqq R'\}.$$

  The multiset of rules finally applied to C_t is selected non-deterministically from $\mathbf{R}'$.

– **Bounded-by-rule parallel derivation mode** ($bound_{B_1,\ldots,B_r}$): Let $\{a, b, \ldots\}$ be the set of different types of rules present in a P system. B_i can be of the following forms:
  - $B_i = j, j \in \{a, b, \ldots\}$;
  - $B_i = \beta_n(B_{1_i}, \ldots, B_{r_i})$, being $n \in \mathbb{N}$, and for each $B_j = \beta_{m_j}(B_{1_j}, \ldots, B_{r_j})$, $j \in \{1_i, \ldots r_i\}$, $m_j \leq n$;
  - As a restriction, a label for a type of rule cannot appear more than once in the whole definition of the derivation mode.

  We say that n is the *bound* of $B_i = \beta_n$. We say that a type of rule (j) is in the *context* of B_i if:
  - There exists $B_i = \beta_n(j)$ (we call B_i its *immediate context*); and
  - There exists $B_i = \beta_n(B_{1_i}, \ldots, B_{r_i})$ such that B_j is a context of the type of rule (j).

  This mode is defined *recursively*, and we can understand the *applicability* of the rules in a defined bounded-by-rule parallel derivation mode in the following sense:
  - In a *context* $\beta_n(B_1, \ldots, B_r)$, the number of rules that can be applied in parallel in a P system in a configuration C_t is n; and
  - In a bounded-by-rule parallel derivation mode $bound_{B_1,\ldots,B_r}$, if $B_i = j (j \in \{a, b, \ldots\})$, being $1 \leq i \leq r$, then rules of type j can be applied in a maximal way.

  With this mode, we can define the classical mode used in P systems with active membranes, that is, evolution rules (a) can be applied in a maximal parallel mode, while the other types of rules (send-in communication rules (b), send-out communication rules (c), dissolution rules (d), division rules for elementary (e) and non-elementary (f) membranes) can be applied at most once per membrane at each computation step. It would be defined as $bound_{a,\beta_1(b,c,d,e,f)}$. If $\mathcal{R}_j$ is the set of rules from $\mathcal{R}$ of the type j, we formally define the bounded-by-rule maximally parallel mode by

$$\begin{aligned}\mathbf{R}' = \{\ & R \mid R \in \mathbf{R} \\ & \wedge \mid \{r \mid r \in R, r \in \mathcal{R}_j\} \mid \leq n \text{ for all } j \text{ in the context of } B_i = \beta_n \\ & \wedge\ \nexists R' \in \mathbf{R} : R \subsetneqq R'\}\end{aligned}$$

Thus, a model type can be defined in P-Lingua by aggregating the allowed rule patterns and its corresponding derivation modes, the syntax is as follows:

```
@model(id) = rule-type-id1,..., rule-type-idN;
```

where `id` is an unique identifier for the model and `rule-type-id1` ,..., `rule-type-idN` are unique identifiers for the corresponding allowed rule patterns. By default all rules behave in maximally parallel derivation mode, but rules can be grouped in sets to behave in bounded parallel derivation mode as follows:

```
@model(id) = @bound{rule-type-id,..., rule-type-idN};
```

where `bound` is a natural number defining the maximum number of rules in the group that can be applied to a given configuration. In Section 6, several examples of model definitions in P-Lingua are given.

4 A command-line tool for generating ad-hoc simulators

A GNU GPLv3 command-line tool called **pcc** has been implemented in C++ language with Flex [28] and Bison [29]. The source code including examples and instructions can be downloaded from https://github.com/RGNC/plingua.

The tool provides three main functionalities:

- **Parsing P-Lingua files** while printing the syntactic and semantic errors to the standard error output. In this sense, the tool acts as a conventional compiler, showing the name of the file, as well as the number of the line and column for each error with a short description. The analysis of semantic errors is done by using the rule patterns and derivation modes defined in the own P-Lingua files. Several files can be compiled together like conventional programs, furthermore standard *makefiles* can be also used. The developer can decide to write the rule patterns and derivation modes in a set of files and reuse them in several projects. More explanations can be found in the website.
- **Generating JSON files**. The tool is able to translate the definitions contained in P-Lingua files to JSON format [30] for compatibility with third-party simulators. The translation is done after parsing the input files, thus the JSON files are free of syntactic/semantic errors and the third-party applications do not have to check them. Several P-Lingua files can be combined together in one JSON file, including also the selected derivation modes.
- **Generating source code**. The tool can generate all the source files for a command-line executable in C++ which is a complete ad-hoc simulator optimized for the design given by the input files. The generated program is able to simulate computations for the defined P system following the specified derivation modes. It interacts with the user by the command-line as common Linux console applications. Generic front-ends could be easily implemented because the command-line options are common to all the simulators. The simulations could be interrupted and resumed since intermediate configurations can be saved in JSON files. Initial multisets can also be defined before the simulation, as well as setting different halting conditions, such as simulating a fixed number of computation steps or running until the execution of a rule marked in the P-Lingua file as halting rule.
  The *pcc* tool performs several analyses over the input files in order to optimize the memory and time that is going to be used for the simulator. The C++ structures used to represent the membrane tree are selected depending on the type of rules that can be used, for instance, if there are

not send-in/send-out rules, then C++ pointers to parent/child membranes are not necessary. The generated code can be compiled with the GNU g++ tool [31], *makefiles* can also be used to automatize all the process from the P-Lingua files to the Linux executable. Instructions and examples can be found in the web page.

5 The simulation algorithm

The compiler presented in Section 4 generates the source code in C++ for an ad-hoc simulator which is able to reproduce computations for the input design written in P-Lingua. The generated code follows the scheme shown in Fig. 1. The simulation is provided by a sequential loop where each iteration simulates one step of computation. For each iteration, the simulator determines the multiset of rules which is going to be applied and then, it applies it to the current configuration C_t obtaining the next configuration C_{t+1}. The halting condition is checked after each iteration.

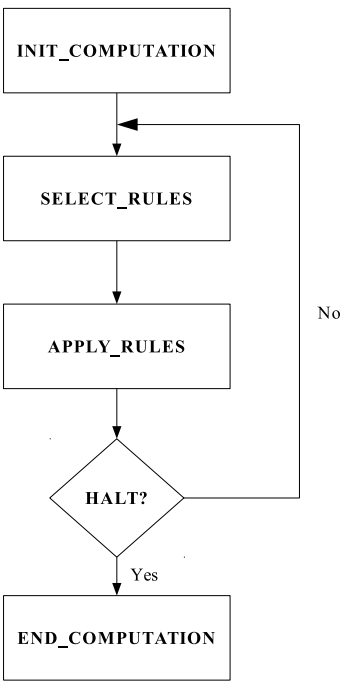


Fig. 1. The main simulation loop

The algorithm used to select rules is described in Pseudocode 1. It returns a multiset B of pairs (m, r) and a configuration C'_t. One pair (m, r) means that rule

r has been selected once to be applied over membrane m in C_t. The configuration C'_t contains a copy of C_t after applying the left-hand side of the selected rules, i.e, after removing from C_t the multisets of objects specified by the left-hand side of the selected rules. On the other hand, the applicability function determines the maximum number of possible applications for a rule r over a membrane m in configuration C'_t. It considers the left-hand side, the charges in the right-hand side, as well as the derivation mode of r. A membrane m in C'_t is marked as $fixed$ if at least one pair (m, r) is contained in B or $unfixed$ otherwise. A rule r cannot be selected if it would change the electrical charge of a $fixed$ membrane.

Finally, Algorithm 2 receives the partial configuration C'_t and generates the next configuration C_{t+1} by applying the right-hand side of the selected rules.

Algorithm 1 SELECT_RULES

Require: Current configuration C_t; Set of rules R;
  $M \leftarrow \emptyset; B \leftarrow \emptyset; C'_t \leftarrow C_t;$
  for each membrane m in C'_t **do**
    $A_m \leftarrow R$; // Copy the set of rules
    $M \leftarrow M \cup m$
    Mark m as $unfixed$
  end for
  while $|M| > 0$ **do**
    $m \leftarrow$ Randomly select one membrane in M
    $r \leftarrow$ Randomly select one rule in A_m
    $k \leftarrow applicability(C'_t, r, m)$
    if $k = 0$ **then**
      Remove r from A_m
      if $|A_m| = 0$ **then**
        Remove m from M
      end if
    else
      $n \leftarrow$ A random natural number in $[1, k]$
      $C'_t \leftarrow apply\_left\_hand\_side(C'_t, r, m, n)$
      $B \leftarrow B \cup \{(m, r)^n\}$
      Mark m as $fixed$
    end if
  end while
  return (C'_t, B)

6 Examples

6.1 Transition P systems

```
!transition_evolution /* Limited to rules with 3 inner membranes */
{
       [a -> v]'h;
```

Algorithm 2 APPLY_RULES

Require: Partial configuration C'_t; Multiset of selected rules B;
  $C_{t+1} \leftarrow C'_t$;
  for each pair (m, r) in B **do**
    $C_{t+1} \leftarrow apply\_right\_hand\_side(C_{t+1}, r, m)$
  end for
  return C_{t+1}

```
       [a -> v, @d]'h;
   (?) [a -> v]'h;
   (?) [a -> v, @d]'h;
       [a [ ]'h1 --> v [w]'h1]'h;
       [a [ ]'h1 --> v [w]'h1]'h;
   (?) [a [ ]'h1 --> v [w]'h1]'h;
   (?) [a [ ]'h1 --> v [w]'h1]'h;
       [a [ ]'h1 [ ]'h2 --> v [w1]'h1 [w2]'h2]'h;
       [a [ ]'h1 [ ]'h2 --> v [w1]'h1 [w2]'h2]'h;
   (?) [a [ ]'h1 [ ]'h2 --> v [w1]'h1 [w2]'h2]'h;
   (?) [a [ ]'h1 [ ]'h2 --> v [w1]'h1 [w2]'h2]'h;
       [a [ ]'h1 [ ]'h2 [ ]'h3 --> v [w1]'h1 [w2]'h2 [w3]'h3]'h;
       [a [ ]'h1 [ ]'h2 [ ]'h3 --> v [w1]'h1 [w2]'h2 [w3]'h3]'h;
   (?) [a [ ]'h1 [ ]'h2 [ ]'h3 --> v [w1]'h1 [w2]'h2 [w3]'h3]'h;
   (?) [a [ ]'h1 [ ]'h2 [ ]'h3 --> v [w1]'h1 [w2]'h2 [w3]'h3]'h;
}

@model(transition) = transition_evolution;
```

6.2 Active membranes with division rules

```
!dam_evolution
{
   ?[a -> v]'h;
   ?[a ->  ]'h;
}

!dam_send_in
{
   a ?[ ]'h -> ?[b]'h;
}

!dam_send_out
{
   ?[a]'h -> b ?[ ]'h;
}

!dam_dissolution
{
   ?[a]'h -> b;
   ?[a]'h ->  ;
}

!dam_division
{
   ?[a]'h -> ?[ ]'h ?[ ]'h;
   ?[a]'h -> ?[b]'h ?[ ]'h;
   ?[a]'h -> ?[ ]'h ?[b]'h;
   ?[a]'h -> ?[b]'h ?[c]'h;
}

@model(membrane_division) =
       dam_evolution, @1{dam_send_in, dam_send_out, dam_dissolution, dam_division};
```

6.3 Tissue-like P systems with communication and cell division

```
!tissue_communication
{
  [u]'h1 <--> [v]'h2;
}

!tissue_division
{
   [a]'h -> [ ]'h [ ]'h;
   [a]'h -> [b]'h [ ]'h;
   [a]'h -> [ ]'h [b]'h;
   [a]'h -> [b]'h [c]'h;
}

@model(tissue_division) =
       tissue_communication, @1{tissue_division};
```

6.4 Population Dynamics P Systems

```
!pdp_evolution
{
   u1 ?[v1]'h -> u2 ?[v2]'h :: ?;
}

!pdp_environment_communication
{
   [[a]'e1 [ ]'e2]'h -> [[ ]'e1 [b]'e2]'h :: ?;
}

@model(probabilistic) =
       pdp_evolution, pdp_environment_communication;
```

7 Conclusions and future work

This paper presents for the first time a compiler for membrane computing which is able to generate C++ source code for optimized ad-hoc simulators. The input P systems are written in P-Lingua, a common language to define membrane computing designs. In this paper we have extended the language to include semantics ingredients, such as rule patterns and derivation modes. The compiler can recognize the rule patterns and show syntactic/semantic errors during the parsing process. The generated simulators are able to simulate computations given by the derivation modes, even if the derivation modes are experimental. Thus, the goal of this tool is twofold: On the one hand, it pretends to be a good assistant for researchers while verifying their designs, even working with experimental models. On the other hand, it provides optimized simulators for real applications, such as robotics or simulation of biological phenomena.

Several lines are open for future work. From the point of view of the language, the semantic ingredients that can be written in P-Lingua should be studied in order to cover more types of models. For instance, defining bounds for the multiplicities of objects in different compartments, such as the environment in tissue-like P systems, where the multiplicity of objects can be infinite. On the other hand, custom directives could be included in P-Lingua files and translated to C preprocessor directives for the simulator. For example, if the design is

confluent, a directive could be written to optimize the simulation time, since it is not necessary to simulate the non-determinism by using random numbers.

From the point of view of the generated simulators, it would be very interesting to produce optimized code for different parallel hardware architectures such as multi-core processors, GPUs or FPGAs. Until now, the faster simulators for parallel architectures are relatively ad-hoc, since several optimizations should be done by analysing the input design. A tool able to automatize this process for a wide variety of input designs could approximate the membrane computing paradigm to other disciplines where it is needed efficient solutions to hard problems. In particular, it could be applied to anytime algorithms for robotics, such as social navigation in crowdy environments or automatic driving, where the robot should have a fast response in real-time, but the solution could be improved by using more computational time.

Acknowledgements

This work was supported by National Natural Science Foundation of China (61672437 and 61702428) and by Sichuan Science and Technology Program (2018GZ0185, 2018GZ0086) and New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0044).

Authors from the University of Seville also acknowledge the support of the research project TIN2017-89842-P, 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. M. Colomer, A. Margalida, and M.J. Pérez-Jiménez. Population Dynamics P System (PDP) Models: A Standardized Protocol for Describing and Applying Novel Bio-Inspired Computing Tools, *Plos One*, 2013 8 (14), 1–13.
2. M. Cardona, M.A. Colomer, M.J. Prez-Jimnez, D. Sanuy, A. Margalida. Modeling ecosystems using P systems: The bearded vulture, a case study. Membrane Computing, 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers. *Lecture Notes in Computer Science*, 5391 (2009), 137-156.
3. M. Colomer, I. Pérez-Hurtado, M.J. Pérez Jiménez, and A. Riscos-Núñez. Comparing simulation algorithms for multienvironment probabilistic Psystem over a standard virtual ecosystem, *Natural Computing*, 11 (2012), 369–379.
4. R. Freund, S. Verlan. A Formal Framework for Static (Tissue) P Systems. *Lecture Notes in Computer Science*, 4860 (2007), 271–284.
5. P. Frisco, M. Gheorghe, M. J. Prez-Jimnez. Applications of Membrane Computing in Systems and Synthetic Biology. *Emergence, Complexity and Computation* (Series ISSN 2194-7287), Volume 7. Springer International Publishing, eBook ISBN 978-3-319-03191-0, Hardcover ISBN 978-3-319-03190-3, 2014, XVII + 266 pages (doi: 10.1007/978-3-319-03191-0).

6. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, and A. Riscos-Núñez. An overview of P-Lingua 2.0, *Lecture Notes in Computer Science*, 5957 (2010), 264–288.
7. M. Ionescu, Gh. Păun, T. Yokomori. Spiking Neural P systems. *Fundamenta Informaticae*, **71**, 2-3 (2006), 279-308.
8. L.F. Macías, I. Pérez-Hurtado, M. García-Quismondo, L. Valencia, M.J. Prez-Jimnez, A. Riscos-Nez. A P-Lingua based simulator for Spiking Neural P systems. *Lecture Notes in Computer Science*, 7184 (2012), 257–281.
9. C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríghez-Patón. Tissue P systems. *Theoretical Computer Science*, **296**, 2 (2003), 295-326.
10. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. A P-Lingua based simulator for Tissue P systems. *Journal of Logic and Algebraic Programming*, 79, 6 (2010), 374–382.
11. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M. García-Quismondo, et al. DCBA: Simulating population dynamics P systems with proportional objects distribution, *Lecture Notes in Computer Science*, 7762 (2013), 257–276.
12. M.A. Martínez-del-Amor, M. García-Quismondo, L.F. Macías-Ramos, L. Valencia-Cabrera, A. Riscos-Núñez, M.J. Pérez-Jiménez. Simulating P systems on GPU devices: a survey. Fundamenta Informaticae, 136, 3 (2015), 269–284.
13. L. Pan, Gh. Paun, M. J. Prez-Jimnez, T. Song. Bio-inspired Computing: Theories and Applications. *Communications in Computer and Information Science* (Series ISSN 1865-0929), Volume 472, Springer-Verlag Berlin Heidelberg, Print ISBN 978-3-662-45048-2, Online ISBN 978-3-662-45049-9, 2014, XX + 672 pages (`doi: 10.1007/978-3-662-45049-9`).
14. L. Pan, Gh. Păun. Spiking Neural P Systems with Anti-Matter. *International Journal of Computers Communications & Control*, **4**, 3 (2009), 273–282.
15. L. Pan, T.-O. Ishdorj. P Systems with Active Membranes and Separation Rules. *Proceedings of the Second Brainstorming Week on Membrane Computing*, 2-7 February, 2004, Sevilla, Spain, pp. 325–341.
16. Gh. Păun, G. Rozenberg, A. Salomaa (eds.). *The Oxford Handbook of Membrane Computing*, Oxford University Press, Oxford, 2010.
17. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143, and *Turku Center for CS-TUCS Report* No. 208, 1998.
18. Gh. Păun. *Membrane Computing. An introduction.* Springer-Verlag, Berlin, 2002.
19. Gh. Păun. P systems with active membranes: attacking **NP**–complete problems, *Journal of Automata, Languages and Combinatorics*, **6** (2001), 75–90.
20. H. Peng, J. Wang, J. Ming, P. Shi, M.J. Prez-Jimnez, W. Yu, Ch. Tao. Fault diagnosis of power systems using intuitionistic fuzzy spiking neural P systems. *IEEE Transactions on Smart Grid*, in press (2017) (`doi: 10.1109/TSG.2017.2670602`).
21. I. Pérez-Hurtado, L. Valencia-Cabrera, J.M. Chacón, A. Riscos-Núñez, M.J. Pérez-Jiménez. A P-Lingua based Simulator for Tissue P Systems with Cell Separation. *Romanian Journal of Information Science and Technology*, 17 , 1 (2014), 89–102.
22. I. Pérez-Hurtado, L. Valencia-Cabrera, M.J. Pérez-Jiménez, M. Colomer, and A. Riscos-Núñez. *MeCoSim: A general purpose software tool for simulating biological phenomena by means of P Systems*, IEEE Fifth International Conference on Bio-inpired Computing: Theories and Applications (BIC-TA 2010), 637–643.
23. F.J. Romero-Campero, M.J. Prez-Jimnez. A model of the Quorum Sensing System in Vibrio Fischeri using P systems. *Artificial Life*, 14, 1 (2008), 95-109 (`doi: 10.1162/artl.2008.14.1.95`).

24. G. Zhang, M. J. Prez-Jimnez, M. Gheorghe. Real-life applications with Membrane Computing. *Emergence, Complexity and Computation* (Series ISSN 2194-7287), Volume 25. Springer International Publishing, Online ISBN 978-3-319-55989-6, Print ISBN 978-3-319-55987-2, 2017, X + 367 pages (doi: 10.1007/978-3-319-55989-6).
25. The P-Lingua web page: http://www.p-lingua.org.
26. The PMCGPU web page: https://sourceforge.net/projects/pmcgpu/
27. The MeCoSim web page: http://www.p-lingua.org/mecosim/.
28. The Flex web page: https://github.com/westes/flexl
29. The Bison web page: https://www.gnu.org/software/bison/
30. The JSON web page: https://www.json.org/
31. The GNU g++ compiler: https://gcc.gnu.org/

Simulation of Pedestrian Behaviors in High-Rise Buildings Based on Intelligence Decision P System

Yunyun Niu[1], Jieqiong Zhang[1], Yulin Chen[1], and Jianhua Xiao[2]

[1] School of Information Engineering,
China University of Geosciences in Beijing,
Beijing 100083, China
[2] The Research Center of Logistics, Nankai University,
Tianjin 300071, China jhxiao2008@163.com

Abstract. Modelling pedestrians evacuation behavior has become an essential issue in improving the safety of high-rise buildings evacuation. The motivation of this paper is to extend the intelligent decision P system (IDPS) to describe and evaluate the behavior selection of pedestrians. T-wo types of pedestrians were studied in the framework of IDPS: queuers, who adhered to the evacuation instructions and had to follow the fron-t ones to maximize the efficiency of global evacuation; and competitors who elbowed their way through the crowd, with frequently overtook each other but no violence whatsoever, maximizing the efficiency of individual evacuation. They corresponded to queuing behavior and competitive be-havior, respectively. The IDPS also modeled obstacle avoidance behavior at two specific fire scenarios, diffused fire scenario and non-diffused fire scenario. The simulation results showed that the cumulative effects of queuing behavior and competitive behavior varied with different walking speeds and different floors. When a fire occurs, the evacuation efficien-cy of queuing behavior was higher than that of competitive behavior, and the number of pedestrian casualties was decreased. The findings of this study can be used to develop new behavioral models of evacuation simulations in high-rise buildings.

Keywords: membrane computing · intelligence decision P system · com-petitive behavior · queuing behavior · high-rise building

1 Introduction

With the development of modern cities, more and more high-rise buildings have risen, and the number of people in these high-rise buildings is usually massive [1]. Thus under the situations such as fire and terrorist attack, the emergency evacuation of such buildings has become an important concern for architects, residents and government [2].

Factors affecting the evacuation characteristics are mainly divided into the following two categories: flow capacity and human behavior [3] [4]. The flow ca-pacity is limited by the geometry characteristics of buildings such as staircase

and door, which has been extensively researched and verified [5] [6]. Human behavior, such as the behavioral preferences and movement characteristics of pedestrians, plays an important role in the evacuation process [7] [8] [9] [10]. Researches have shown that pedestrian's behavior in the next phase largely depends on their behavioral preferences, in which case rational choice will be made by weighing all available options and choosing what s/he thinks is adequate rather than optimal [11] [12]. In addition, significant time pressures caused by crisis or uncertainty can encourage pedestrians to search for more options [13] [14]. Therefore, pedestrian's behavioral differences should be studied to predict the outcome of possible evacuation schemes and to reduce the possible negative effects of disasters [15]. When the situation is threatening pedestrian's live in a short period of time, s/he may speed up to escape and ignore the social norms of politeness. However, when the situation is not so serious, the motion characteristics of evacuation process are similar to normal conditions.

Most studies use modeling to simulate pedestrians' behavior in emergencies. Pedestrians' behavior simulation models in the existing literature can be categorized into two categories: discrete model and continuous model [16]. Most discrete models are based on cellular automata model, with space and time discrete during the evacuation while the continuous models simulate the position of each person per second in succession [17]. The social force model is a certain example of continuous models [18]. Many researches have advanced the simulation progress of pedestrians' behavior with discrete models and continuous models. Discrete model needs to establish a series of rules to meet the discrete requirement for time and space, which makes pedestrian behavior simulations not as accurate as a continuous model. Moreover, the pedestrians in the continuity model are completely affected by the force, and the intelligent behavior of the people is rarely considered. Some assumptions oversimplify the path discovery process of pedestrians, so the continuity model cannot effectively imitate the more complicated behavior of pedestrians. Though behavioral models have been used to describe features of pedestrian flow in normal operation situations, few studies analyze the interaction between individual pedestrians in emergencies.

In traditional P systems, cells were designed for computing rather than moving [19]. The computational results were usually defined according to objects present in the output membrane in the halting condition. Recently, some membrane systems have involved the concept of position, such as spatial P systems [20] and the grid-exploring P system [21]. In this study, the intelligence decision P system (IDPS) was further expanded, taking into account the effect of pedestrian preference [22]. Compared with the previous work, the most prominent difference is modeling pedestrian behaviors in high-rise buildings. IDPS has been adopted to model the effects of the factors which may affect pedestrian behaviors in evacuation of high-rise buildings. The main contributions of this paper can be summarized as follows.

(1) The IDPS is used to study pedestrian behavior considering pedestrian preferences and random attributes. This work contains two behaviors, which correspond to two types of pedestrians. Two types of pedestrians use different

ways of interaction to adjust their behaviors. This model described the evacuation process as realistically as possible to contribute to develop behavioural models for evacuation simulations in high-rise buildings.

(2) During the evacuation process, the difference in the cumulative effects of the two behaviors was found. At the same time, there was no overtaking in the process of evacuation for the queuing behavior. In the process of evacuation for competitive behavior, a process similar to queuing evacuation occurs in staircases.

In the remainder of this paper, some necessary preliminaries are noted in Section 2 with respect to the formal definition of the IDPS model. The detailed implementation method of simulating behavior evacuation is described in Section 3. In Section 4, the evacuation simulation of competitive behavior and queuing behavior is carried out with a discussion of the simulation results. The conclusions are given in the last section.

2 Preliminaries

Formally, an *intelligence decision P system* of degree $n \geq 1$ is defined as follows:

$$\Pi = (\Gamma, E^{(0)}, C_1^{(0)}, \ldots, C_n^{(0)}, \mathcal{R}, G, s_s, s_t),$$

where:

- $n \geq 1$ (the system contains n cells, labeled with $1, 2, \cdots, n$; all these n cells are placed in the environment and the environment is labeled with 0);
- Γ is the *alphabet* of *objects*;
- $E^{(0)}$ is the set of objects in the environment at the beginning of the simulation, each with an arbitrarily number of copies; the environment is defined as the place of cell movement. It includes signal objects, some of which help start or end the migration and others that have a large influence on the speed and directions of cells. It keeps a record of the current position of each cell.
- $C_i^{(j)} = \{p_i^{(j)}, v_i^{(j)}, K_i^{(j)}, m_i^{(j)}\}$ is the state of cell i at step j, where $p_i^{(j)} = (x_i^{(j)}, y_i^{(j)})$ is the real-time location of cell i; $v_i^{(j)}$ is the speed of cell i at step j; $K_i^{(j)} = (k_1, \ldots, k_n)$ describes the knowledge base of cell i at step j; and $m_i^{(j)}$ indicates the type of cell. $C_i^{(0)}$ denotes the initial state of cell i.
- $C_1^{(0)}, \ldots, C_n^{(0)}$ are strings over Γ, describing n *knowledge bases* placed in $1, \ldots, n$ cells respectively at the beginning of the simulation;
- $\mathcal{R}$ includes four types of rules:
  1. *Knowledge base update rules*:
     Cells make decision based on their own knowledge bases. Their initial knowledge bases are set according to their types m_i, their initial locations, etc. Their knowledge bases are then updated in two different ways. The first is by obtaining information from the environment. The second is by exchanging information with other cells.

(1) Interaction with the environment:
Cells obtain information from the environment, such as real-time traffic data, route guidance signals, obstacles, the starting signal and the termination signal. Each piece of information has a perception region. If a cell is in this region, then it can obtain the information as described by the following rules:

$$K_i^{(j)} \to K_i'^{(j+1)}, \forall i \in \{1, \ldots, n\},$$

$$K_i'^{(j+1)} = \begin{cases} K_i^{(j)} \cup \{a\}, & p_i \in R_a \\ K_i^{(j)}, & p_i \notin R_a \end{cases}$$

where R_a indicates the perception region of information a. The knowledge base of cell i is updated by adding information a, if cell i reaches R_a.
(2) Communication with other cells:
Cells can share information with their neighbors. Assume that the distance of two neighbors is no more than a threshold d. If $l_{best} \in K_i'^{(j+1)} \cup K_k'^{(j+1)}$, $|p_i - p_k| \leq d$ is the information that leads to the best running plan of cell i or cell j, and if l_{best} is not in $K_i'^{(j+1)}$, then $K_i'^{(j+1)}$ is updated by adding information l_{best}.

$$K_i'^{(j+1)} \to K_i^{(j+1)}, \forall i \in \{1, \ldots, n\},$$

$$K_i^{(j+1)} = \begin{cases} K_i'^{(j+1)} \cup \{l_{best}\}, & l_{best} \notin K_i'^{(j+1)} \\ K_i'^{(j+1)}, & l_{best} \in K_i'^{(j+1)} \end{cases}$$

2. *Type transition rules*:
Cell type is defined according to the knowledge base in the cell. A knowledge base change may lead to a cell type transition.

$$K_i^{(j+1)} m_i^{(j)} \to K_i^{(j+1)} m_i^{(j+1)}.$$

3. *Decision-making rules*:
According to the knowledge base and the current position, cell i can obtain several running schemes at step j.

$$K_i^{(j)} p_i^{(j)} \to \{Scheme_{i,1}^{(j)}, \ldots, Scheme_{i,i_k}^{(j)}\}.$$

The best scheme $Scheme_{i,best}^{(j)}$ is chosen according to a specific requirement (see Figure 1), e.g., minimizing the distance from the current position to the exit.

$$\{Scheme_{i,1}^{(j)}, \ldots, Scheme_{i,i_k}^{(j)}\} \to Scheme_{i,best}^{(j)}.$$

Priorities can be easily added to these rules for decision-making.

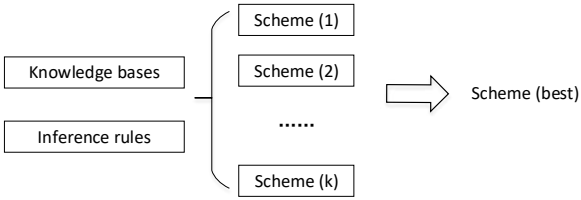


Fig. 1. Decision-making rules.

4. *Position-updating rules*:
   The expected velocity and moving direction can be calculated from $Scheme_{i,best}^{(j)}$,

$$Scheme_{i,best}^{(j)} \rightarrow (v_i'^{(j)}, d_i'^{(j)}).$$

   In the process of moving, the actual situation may not allow cells to follow the planned direction. In that case, the cells must adjust their behavior. For example, when a cell encounters an obstacle, it must bypass the obstacle to move forward. The actual velocity and direction of the cell i at step j are denoted by $v_i^{(j)}$ and $d_i^{(j)}$, respectively.

$$(v_i'^{(j)}, d_i'^{(j)})\{o_1, \ldots, o_q\} \rightarrow (v_i^{(j)}, d_i^{(j)}),$$

   where $o_1, \ldots, o_q$ represents obstacles. Cell i updates its position according to the current position and speed.

$$(x_i^{(j)}, y_i^{(j)})(v_i^{(j)}, d_i^{(j)}) \rightarrow (x_i^{(j+1)}, y_i^{(j+1)}),$$

   where $(x_i^{(j)}, y_i^{(j)})$ is the position of cell i at step j and $(v_i^{(j)}, d_i^{(j)})$ is the speed and the moving direction of cell i at step j.

- G is the set of destinations or exits of the moving cells.
- s_s is the starting signal for cell movement and s_t is the termination signal.

The rules of a system as above are used in a nondeterministic maximally parallel manner. When the starting signal s_s appears in the environment, cells move towards their goals under the guidance of the rules in $\mathcal{R}$. At each step, cells decide what is the next actions to be performed, i.e., velocities and directions. They usually stop moving when they reach their destinations or when the termination signal s_t appears in the environment.

A configuration of IDPS Π is described by the multisets of objects in the cells and the environment. $C_1^{(j)}, \ldots, C_n^{(j)}$ represent the states of all the cells present in the system at step j. They involve four basic cell characteristics, ie.e., position, speed, knowledge base and type. The configuration of the next system is determined by rules in R applied to the previous system. All computations start from

the initial configuration and proceed. Cells stop moving when they reach their destinations or when they get the termination signal s_t from the environment. The system stops evolving when all cells stop moving. The simulation results are obtained by counting the number of cells going through the exits.

3 Model

The behavior of pedestrian evacuation in high-rise buildings is studied based on an intelligence decision P system. Two behavior types are included: queuing behavior, in which pedestrians adhere to the evacuation instructions and maximize the efficiency of global evacuation; and competitive behavior, in which pedestrians ignore the evacuation instructions and try to maximize the efficiency of individual evacuation. The evacuation instructions usually contains: (1) No overtaking, no scrambling, no pushing; (2) Follow the right-hand rule.

In the process of evacuation, pedestrians could walk freely from current location to the door, and wait in line before leaving. Therefore, the estimated evacuation time is comprised of the time of pedestrian walking freely from current position to the door and the time of queuing at the door. Estimated evacuation time is calculated as follows [23]:

$$T_i^b(t) = T_{Wi}^b(t) + T_{Qi}^b(t),$$

where $T_i^b(t)$ is the estimated evacuation time of pedestrian i going through the door; $T_{Wi}^b(t)$ is the time of pedestrian i walking from current position to the door, and $T_{Qi}^b(t)$ is the time of pedestrian i queuing at the door. According to the principle of "first-come-first-served", if a pedestrian arrives at the door earlier, then s/he walks out earlier.

3.1 Knowledge base update

Cells update their knowledge bases in two different ways. The first is by obtaining information from the environment. The second is by exchanging information with other cells.

(1) Interaction with the environment:
Cells obtain information from the environment, such as real-time traffic data, route guidance signals, obstacles, the starting signal and the termination signal. Each piece of information has a perception region. If a cell is in the region, then it can obtain the information as shown in Fig. 2. Cells can perceive exits, fires and congestion through maximum visual angle (e.g., 150°) and maximum visual distance (e.g., $10m$). When the target exit is congested, the cell A can re-direct another path. In contrast, the cell B insists on its initial selection path because the alternative exit is beyond its visual area. The cell may not notice the situation ahead unless the obstacle is within the maximum visual distance. If there are obstacles in the field of vision, cells will take action to avoid obstacles.

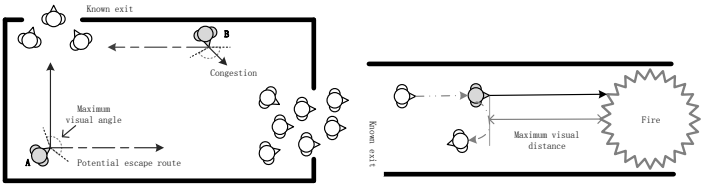


Fig. 2. Visual observation of cells.

(2) Communication with other cells:
When the distance between two neighbors does not exceed the threshold d_{nei}, cell can share information with its neighbors in probability $P_{interaction}$. If $l_{fire} \in K_i'^{(j+1)} \cup K_k'^{(j+1)}$, $|p_i - p_k| \leq d_{nei}$ is the information that leads to the obstacle avoidance behavior of cell i or cell k, and if l_{fire} is not in $K_i'^{(j+1)}$, then $K_i'^{(j+1)}$ is updated by adding information l_{fire}.

$$K_i^{(j+1)} = \begin{cases} K_i'^{(j+1)}, & l_{fire} \in K_i'^{(j+1)} \\ K_i'^{(j+1)} \cup \{l_{fire}\}, & l_{fire} \notin K_i'^{(j+1)} \end{cases}$$

3.2 Decision-making mechanism

According to the knowledge base, the current position and the type of cell, cell can determine it's own evacuation pattern and optimal running scheme. $m_i^{(j)}$ indicates the type of cell, where $m_{i(Q)}^{(j)}$ denotes the queuer with queuing behavior, and $m_{i(C)}^{(j)}$ denotes the competitor with competitive behavior.

$$K_i^{(j)} p_i^{(j)} m_i^{(j)} \rightarrow \{Scheme_{i,1}^{(j)}, \ldots, Scheme_{i,i_k}^{(j)}\}.$$

As shown in Fig. 3, when all cells are evacuated at the same speed, queuer give priority to cells with short evacuation time according to the evacuation time. However, competitor only consider the fastest evacuation method based on the shortest distance. Note that the evacuation time and the shortest distance are only for the next hop position in the overall evacuation direction.

3.3 Position-updating mechanism

Pedestrians have the requirements of keeping distances from the ones in the front during their movement. According to [24], the crowd density is from 1.4 to 2.1 $persons/m^2$. As shown in Fig. 4, (A) presents the larger neighborhood, and (B) is the smaller neighborhood. The grid is designed to show the relative position of each individual. As the density of pedestrians increases, pedestrians

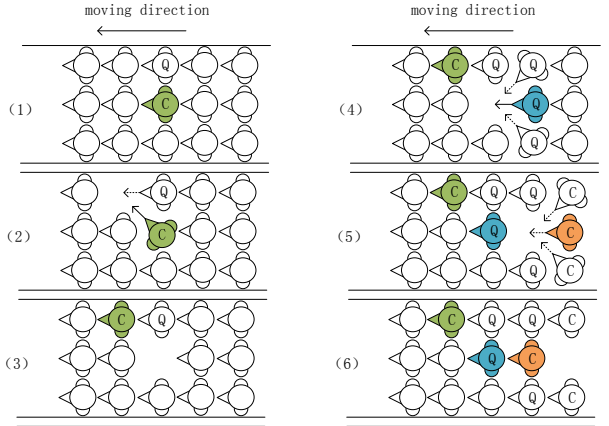


Fig. 3. Decision-making mechanism of different types of cells.

maintain a shorter distance than normal condition, and they try to push their way through the bottleneck, their neighborhood will be changed from Type A to Type B [25]. According to the pedestrian behavior, each pedestrian can choose eight neighbors for interaction and evacuation. Different density conditions are corresponding to Fig. 5, neighborhood type A and B are shown on the left and on the right, respectively.

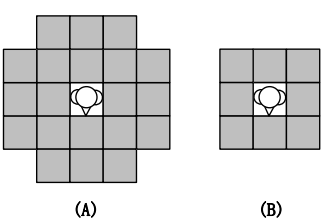


Fig. 4. Neighborhoods in different density conditions.

$P_{m_i^{(j)}}$ indicates the probability that the queuer or competitor will take a step forward when the next position of two or three pedestrians is the same square.

$$m_i^{(j)} P_{m_i^{(j)}}(v_i'^{(j)}, d_i'^{(j)}) \rightarrow (v_i^{(j)}, d_i^{(j)}),$$

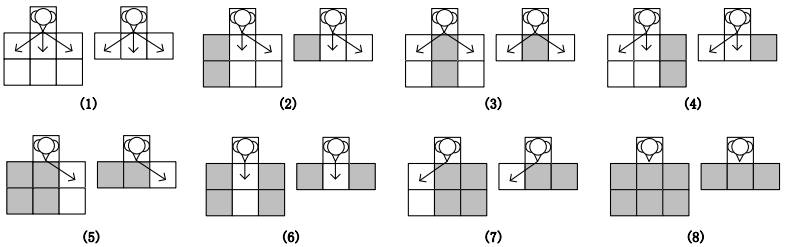


Fig. 5. The evacuation direction.

To be specific, there are four cases when the next position of the two pedestrians is the same position:

$$\begin{cases} P_{Q_x} = 0, P_{Q_y} = 1, & Q_x(t) > Q_y(t) \\ P_{Q_x} = 1/2, P_{Q_y} = 1/2, & Q_x(t) = Q_y(t) \\ P_{C_x} = 1/2 + 3\alpha/2,\ P_{C_y} = 1/2 - 3\alpha/, & C_x(t) < C_y(t) \\ P_{Q_x} = 0, P_{C_y} = 1 \end{cases}$$

The method of calculating probability is the same when the next position of three and two pedestrians is the same square. Q_x represents a queuer standing at the x position, C_y represents a competitor standing at the y position. x, y and z represent the current location of pedestrian, which are adjacent but not identical. To describe that pedestrians try to walk up stairs at the shortest time and distance, parameter α is introduced as the attraction value with $-1/3 \leq \alpha \leq 1/3$.

3.4 Simulation scenario

A 12-story teaching building located at China University of Geosciences in Beijing(CUGB) was the sample evacuation environment in this study. 300 pedestrians distribute randomly on each floor. The ground floor plan include four staircases, two elevators and nine exits leading to outside, as shown in Fig. 6. Considering the fact that the evacuees are not allowed to use elevators during a fire emergency, the two elevators are set as occupied space. The floor plan from the 2nd floor to the 12th floor is shown in Fig. 7, with 17 lecture rooms and four doors leading to staircase. The size of each cell is $1.0m \times 1.0m$. Only one pedestrians can be accommodated in a cell at most. The width of staircase is three cells.

Three are three kinds of pedestrians in the simulation: fast pedestrians, moderate-speed pedestrians and slow pedestrians which are marked as FD, MP and SP, respectively. The three kinds of expectation speeds are defined as follow: $v_{FD} = 0.608m/s$, $v_{MP} = 0.416m/s$, $v_{SP} = 0.224m/s$ [26]. This study mainly concentrates on the influence of queuing behavior and competition behavior on the evacuation efficiency, it simplifies the generation and spread of fire to some

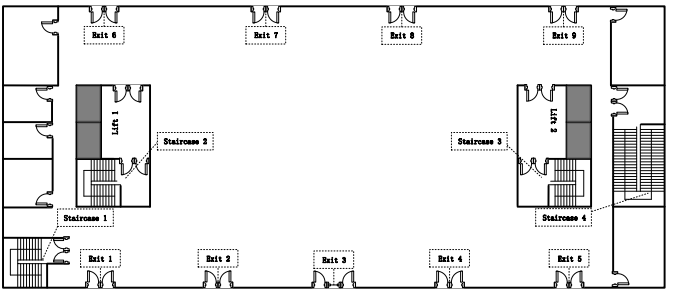


Fig. 6. The floor plan of the first floor.

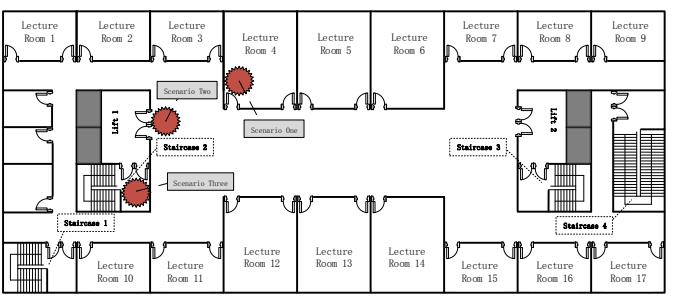


Fig. 7. The floor plan of the second floor and above.

extent. With these assumptions, three possible fire scenarios were designed. The fire scenarios are follows:

- Scenario One: A minor fire occurs in the lecture room 4 on the seventh floor, there are two cases of fire spread and no spread.
- Scenario Two: A minor fire occurs at the gate of Elevator 1 on the seventh floor, the fire will not spread.
- Scenario Three: A minor fire occurs in the staircase 2 on the seventh floor, the fire will not spread.

The semantic representation of the floor plan is constructed and contains the inclusion relation, the connectivity relationship, and the accessibility relationship [27]. Using Scenario One as an example, the fire occurred near the left door of Lecture Room 4, enabling the accessibility relationship between Lecture Room 4 and corridor through the right door. When the pedestrians escape from the Lecture Room 4, choose staircase 2 or 3 for evacuation. The potential escape routes for pedestrians are unblocked routes in Scenario Two and Scenario

Three, are listed in Table 1. During the evacuation, the pedestrians will eliminate the blocked routes and turn back as it encounters the fire, and choose another shortest route from all initial routes.

Table 1. Potential evacuation routes of evacuees evacuate from Lecture Room 4 in three fire scenarios

Experiment	Initial escape route	Escape route			
		Staircase 1	Staircase 2	Staircase 3	Staircase 4
Scenario 1	Staircase 2	×	√	√	×
Scenario 2	Staircase 2	√	×	√	×
Scenario 3	Staircase 2	√	×	√	×

3.5 IDPS-base simulation of evacuation

All cells have initial knowledge base $K_i^{(i)}$ at the beginning. When the s_s appears, cells start moving. In detail, the evacuation process can be simulated according to the following steps.

Step 1: The system checks whether there is a termination signal s_t. If there is a termination signal s_t, the simulation is finished. Otherwise, go to step 2.

Step 2: If the cell finds itself in fire, the experiment of the cell is terminated.

Step 3: If the cell is a competitor, it moves toward the exit by elbowing the crowd. Otherwise, go to step 4.

Step 4: If the cell is a queuer, it follows the evacuation instructions to evacuate.

Step 5: The cell interacts with its neighbours and share information with them.

Step 6: The cell updates the knowledge base.

Step 7: The cell makes path planning according to the decision-making rules.

Step 8: The cell adjusts its direction and moves with the actual velocity.

Step 9: If the cells reach the exit, the evacuation is successful. Otherwise, the simulation continues and goes to step 2.

4 Simulations and analysys

All the algorithms are implemented by using NetLogo. Experiments are run on an Intel (R) Core (TM) i5-2520M CPU with a 4 GB RAM PC. To report the reliable estimation of simulated evacuation times for various experimental configuration settings, each configuration was simulated for 10 times, and the estimates were averaged. The experiment ends when all surviving cells escaped from the building.

4.1 Effect of speed on two evacuation behaviors

In this subsection, the impact of speed on two evacuation behaviors is analyzed. As shown in Table 2, the three kinds of speed are as follow: $v_{FD} = 0.608m/s$, $v_{MP} = 0.416m/s$, $v_{SP} = 0.224m/s$, the difference value between the evacuation times of the two behaviors has almost doubled. As shown in Fig. 8, as the speed

Table 2. The evacuation time of two behaviors at different speeds on staircase 2

Experiment	v_{SP}	v_{MP}	v_{FD}
Queuing	$4320.0s$	$2545.3s$	$1734.1s$
Competition	$4442s$	$2554s$	$1662s$
Difference	$680s$	$342s$	$166s$

decreases, the evacuation time of both behaviors gradually increases. When the minimum speed is $0.224m/s$, both the competitive behavior and the queuing behavior have cumulative effect on the 12th floor. With the decrease of floor and the increase of speed, the cumulative effect of queuing behavior gradually decreases and disappears, while the cumulative effect of competitive behavior decreases but does not disappear.

4.2 The effect of two kinds of evacuation behaviors on the use of stairs

In this subsection, the effects of two evacuation behaviors on the use of four stairs are analyzed. In the scenario 1 and scenario 2, as shown in Fig. 9, at the entrance of four staircases, the number of evacuees for queuing behavior over per unit time is greater than that of the competitive behavior. And the evacuees using staircase 3 are more than those using staircase 2. In scenario 2, staircase 1 takes the largest amount of evacuation. The specific evacuation time for scenarios 1 and 2 is shown in Table. 3.

Table 3. The evacuation time for scenarios 1 and 2

Experiment		Exit 1	Exit 2	Exit 3	Exit 4
Scenario 1	Queuing	$129s$	$90s$	$76s$	$196s$
	Competition	$122s$	$91s$	$124s$	$129s$
Scenario 2	Queuing	$209s$	$12s$	$78s$	$155s$
	Competition	$216s$	$20s$	$113s$	$120s$

At the junction of staircase leading to the lobby of ground floor, capacity changes are shown in Fig. 10, in the scenario 3, irrespective of competition behavior or queuing behavior, the evacuee's number and time of evacuation on staircase 2 are the least, the evacuation time consumed in staircase 1 is the

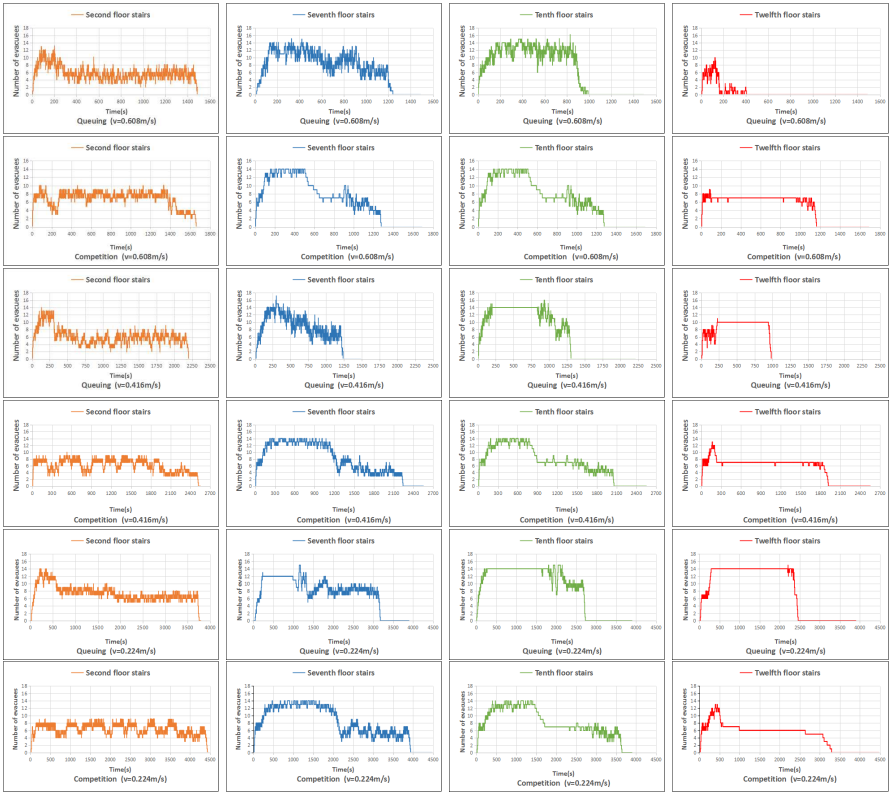


Fig. 8. Capacity changes at different speeds on the staircase 2

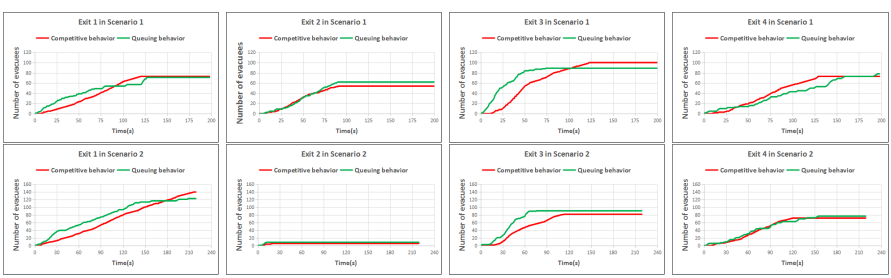


Fig. 9. The cumulative evacuees through each entrance of staircase during scenario 1 and scenario 2

longest, and the evacuation times for staircase 3 and staircase 4 are in one time interval. Meanwhile, the evacuation time of queuing behavior is the shortest. The specific total evacuation time for scenarios 3 is shown in Table. 4.

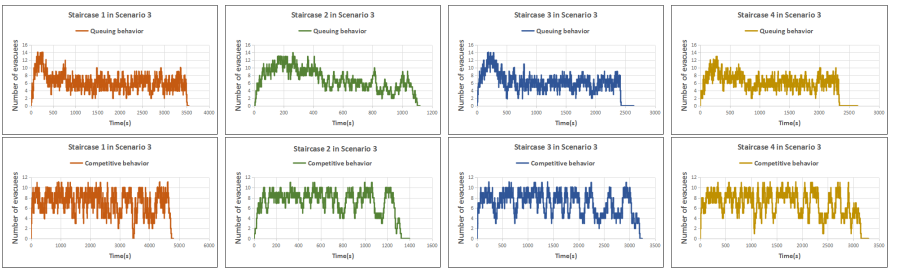


Fig. 10. Capacity changes of two kinds of behavior in scenario 3

Table 4. The total evacuation time for scenarios 3

Experiment		Staircase 1	Staircase 2	Staircase 3	Staircase 4
Scenario 3	Queuing	3512s	1102s	2433s	2339s
	Competition	4746s	1325s	3207s	3155s

4.3 Casualties of two behaviors caused by propagating fire

In this section, the casualties caused by two types of behavioral evacuation in a diffuse fire are summarized. When the speed of evacuator is less than the diffusion rate of fire and the evacuee enters the burning range of fire, the evacuee is deemed dead. The relationship between the ratio of fire spread area and the number of evacuees casualty is shown in Table. 5. When the fire spread over time in Scenario One, the casualties of the two behaviors increase gradually, and the casualties of queuing behavior is less than that of competitive behavior.

Table 5. Number of casualties in different areas of fire spread

Experiment	Proportion of fire diffused area			
	20%	40%	60%	80%
Queuing	8	23	52	80
Competition	25	41	93	129

5 Conclusions

In this work, a series of evacuation simulations were carried out on the 12-story campus building to study the behavior adjustment mechanism based on the IPDS model. The simulation results show that the evacuation efficiency of queuing behavior is much higher than that of competitive behavior, and in specific fire scenarios, the number of casualties in queuing behavior is lower than that in competitive behavior. There is a certain linear relationship between the evacuation time of queuing behavior and competition behavior. At the same time, there has no overtaking in the process of evacuation for the queuing behavior. In the process of evacuation in the staircases, competitive behavior is similar to queuing behavior.

However, it is important to note that the simulation mainly focuses on the impact of two behaviors on evacuation, excluding all the factors that affect the accessibility of the architectural environment. Fire smoke and toxic gases will lead to dynamic spatial accessibility and affect the movement speed and direction of evacuees. In addition, evacuation process simplifies the evacuation movement modeling without considering some factors, such as the representation of the human body and the panic level. Therefore, more factors need to be considered in the future work, such as the reaction of evacuees in the presence of smoke.

Acknowledgment

This work was supported by the National Natural Science Foundation of China [grant numbers 61872325, 61502012, 61772290]; the Asia Research Center in Nankai University [grant number AS1711]; Fundamental Research Funds for the Central Universities; and the Collaborative Innovation Center for China Economy.

References

1. Shields, T.J., Boyce, K.E., McConnell, N.: The behaviour and evacuation experiences of WTC 9/11 evacuees with self-designated mobility impairments. Fire Safety J **44**(6), 881–893 (2009)
2. Zhou, M., Dong, H., Wen, D., Yao, X., Sun, X.: Modeling of Crowd Evacuation With Assailants via a Fuzzy Logic Approach. IEEE T Intell Transp **17**(9), 2395–2407 (2016)
3. Pelechano, N., Malkawi, A.: Evacuation simulation models: challenges in modeling high rise building evacuation with cellular automata approaches. Automat Constr **17**(4), 377–385 (2008)
4. Ma, J., Song, W.G., Tian, W., Lo, S.M., Liao, G.X.: Experimental study on an ultra high-rise building evacuation in China. Safety Sci **50**, 1665–1674 (2012)
5. Cepolina, E.M.: Phased evacuation: an optimisation model which takes into account the capacity drop phenomenon in pedestrian flows. Fire Safety J **44**(4), 532–544 (2009)

6. Kretz, T., Grunebohm, A., Schreckenberg, M.: Experimental study of pedestrian flow through a bottleneck. J STAT MECH-THEORY E **10**, 1–27 (2006)
7. Nicolas, A., Bouzat, S., Kuperman, M.N.: Influence of selfish and polite behaviours on a pedestrian evacuation through a narrow exit: A quantitative characterisation. Proceedings of the 8th International Conference on Pedestrian and Evacuation Dynamics **18**, 1–9 (2016)
8. Shi, D., Zhang, W., Wang, B.: Dynamics of Panic Pedestrians in Evacuation. Physics 1–6 (2017)
9. Tomonori, S., Masanori, Y., Hiroyuki. K., Ai, S.: Human behavior in a staircase during a total evacuation drill in a high-rise building. Fire Mater **41**(4), 375–386 (2017)
10. Hamilton, G.N., Lennon, P.F., Raw, J.: Human behaviour during evacuation of primary schools: Investigations on pre-evacuation times, movement on stairways and movement on the horizontal plane. Fire Safety J **91**, 937–946 (2017)
11. Slovic, P., Fischhoff, B., Lichtenstein, S.: Behavioral decision theory. Annu Rev Psychol **28**(1), 1–39 (1977)
12. Lovreglio, R.: Modelling decision-making in fire evacuation using random utility theory. Politecnico di Bari, Milano (2016)
13. Zakay, D.: The impact of time perception processes on decision making under time stress. In: Svenson O, Maule AJ (eds) Time pressure and stress in human judgment and decision making. Springer, Boston, MA 59–72 (1993)
14. Ben, H., Breznitz, S.J.: The effect of time pressure on risky choice behavior. Acta Psychol **47**(2), 89–104 (1981)
15. Kuligowski, E.D.: Human behavior in fire. Springer, New York, NY 2070–2114 (2016)
16. Wang, L., Zheng, J., Zhang, X., Zhang, J., Wang, Q., Zhang, Q.: Pedestrians behavior in emergency evacuation: Modeling and simulation. Chin. Phys. B **25**(11), 1–10 (2016)
17. Pereira, L., Duczmal, L., Cruz, F.: Congested emergency evacuation of a population using a finite automata approach. Safety Sci **51**(1), 267–272 (2013)
18. Yuen, J., Lee, E.: The effect of overtaking behavior on unidirectional pedestrian flow. Safety Sci **50**(8), 1704–1714 (2012)
19. Ionescu, M., Păun, G., Yokomori, T.: Spiking Neural P Systems. Fund. Inform. **71**(2-3), 279–308 (2006)
20. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Pardini, G., Tesei, L.: Spatial P systems. Nat. Comput. **10**(1), 3–16 (2011)
21. Hinze, T., Weber, L.L., Hatnik, U.: Walking membranes: grid-exploring P systems with artificial evolution for multi-purpose topological optimisation of cascaded processes. CMC 2016: Membrane Computing. **10105**, 251–271 (2017)
22. Niu, Y., Zhang, Y., Zhang, J. Xiao, J.: Running Cells with Decision-Making Mechanism: Intelligence Decision P System for Evacuation Simulation. Int. J. Comput. Commun **13**(5), 865–880 (2018)
23. Li, Y., Jia, H.: Pedestrian evacuation behavior analysis and simulation in multi-exits case. Int J Mod Phys C **28**(10), 1–15 (2017)
24. Ding, N., Zhang, H., Chen, T.: Stair evacuation simulation based on cellular automata considering evacuees' walk preferences. Chin Phys B **24**, 687–693 (2015)
25. Ding, N., Zhang, H., Chen, T.: Simulation-based optimization of emergency evacuation strategy in ultra-high-rise buildings. Natural Hazards **89**, 1167–1184 (2017)
26. Peacock, R., Reneke, P., Kuligowski, E., Hagwood. C.: Movement on Stairs During Building Evacuations. Fire Technol **53**, 845–871 (2017)

27. Tan, L., Hu, M., Lin. H.: Agent-based simulation of building evacuation: Combining human behavior with predictable spatial accessibility in a fire emergency. Inform Sciences **295**, 53–66 (2015)

A semantic frontier of the efficiency in membrane systems

David Orellana-Martín, Luis Valencia-Cabrera,
Agustín Riscos-Núñez, and Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
{dorellana,lvalencia,ariscosn,marper}@us.es

Abstract. The (presumed) efficiency of computing models is expressed by means of their ability to solve (**NP**-complete) *presumably intractable* problems in an efficient way or not. The relevance of this kind of frontier lies in the fact that each of them provides a tool to attacking the **P** versus **NP** problem.
In the framework of Membrane Computing, different borderlines of the efficiency have been described by means of syntactical ingredients associated with the models. In this paper, a frontier of semantic type is presented within the model of polarizationless P systems with active membranes and membrane creation.

Keywords: Membrane Computing, Computational complexity theory, Membrane creation

1 Introduction

Membrane Computing is a bioinspired computing discipline with contributions by researchers from different fields (computer scientists, formal linguists, biologists, etc.) enriching each other with results, open problems and promising new research lines. Specifically, it is a paradigm aiming to abstract computing models from the structure and functioning of the living cell as well as from the organization of cells in tissues, organs, and other higher-order structures. It starts from the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. Membrane computing provides very flexible and versatile distributed parallel non-deterministic devices, generically called *membrane systems*. The main syntactical ingredients are the following: a finite alphabet (the *working* alphabet whose elements are called *objects*, which are abstractions of chemical substances); a finite set of processor units delimiting *compartments* (called *membranes*, *cells* or *neurons*), interconnected by a *graph-structure*, in such manner that initially each processor contains a multiset of objects; a finite set of *rewriting rules* (*evolution rules*), abstractions of chemical reactions, that provide the dynamics of the system; and

an *environment*. The objects to evolve in a computation/transition step and the rules by which they evolve are chosen in a non-deterministic and maximally parallel manner: we assign objects to rules but in such a way that after that, no further rule can be applied to the remaining objects.

According with the type of structure underlying the systems, there are basically three types of membrane systems: *cell-like* P systems where the compartments are arranged in a hierarchical structure (formally, a rooted tree), like in a living cell [9]; *tissue-like* P systems with a directed graph structure associated inspired from the living tissues where cells bump into each other and communicate through pores or other membrane mechanisms [13]; and *neural-like* P systems with a directed graph structure associated which mimic the way that neurons communicate with each other by means of short electrical impulses, identical in shape (voltage), but emitted at precise moments of time [11].

Basic transition P systems are a kind of cell-like membrane systems whose membrane structure (a labelled rooted tree) does not grow, that is, there are no rules producing new membranes in the system. It is well known ([3]) that by using families of these membrane systems only problems in class **P** can be solved in a uniform and polynomial time. Thus, the ability of a cell-like membrane system to construct an exponential workspace (in terms of number of objects) in polynomial time (e.g. via evolution rules of the type $[\, a \to a^2 \,]_h$) is not enough to efficiently solve **NP**–complete problems (assuming that $\mathbf{P} \neq \mathbf{NP}$).

In cell-like membrane systems, there are different bioinspired mechanisms to produce an exponential workspace (both in terms of number of objects and *number of membranes*) in polynomial time. For instance, *Mitosis* is a process of cell division which results in the production of two daughter cells from a single parent cell. Daughter cells are identical to one another and to the original parent cell. Through a sequence of steps, the replicated genetic material in a parent cell is equally distributed to two daughter cells. While there are some subtle differences, mitosis is remarkably similar across organisms. Another mechanism, called *Membrane fission*, is a process by which a biological membrane is split into two new ones in such a way that the content of the initial membrane is separated and distributed between the new membranes. As another example observed in Nature, *Autopoiesis* is a general process referred to a system capable of reproducing and maintaining itself. Membranes are created in living cells through its own internal processes, for instance, in the process of vesicle mediated transport, and to keep molecules close to each other to facilitate their reactions. Membranes can also be created in a laboratory - see [5].

These biological phenomena of mitosis, membrane fission and autopoiesis were incorporated in cell-like membrane systems through new kinds of rewriting rules, called *membrane division* ([10]), *membrane separation* ([1, 8]), and *membrane creation* ([6, 7]), respectively.

This paper deals with cell-like membrane systems with membrane creation. In [7], *passive P systems* and *active P systems* are considered. The first ones have the property that their membrane structure cannot increase during any computation, while the second ones have the property that their membrane

structure may increase during a computation (e.g. by creating new membranes whose label is different from the existing ones). The first definition of P systems with membrane creation is developed considering transition P systems with rules for membrane creation of the form $[\, a \to [\, u\,]_{h_1}\,]_h$, where a is an object, u is a multiset of objects, and h, h_1 are labels such that h_1 is not the label of the skin membrane. When such a rule is applied in a membrane labelled by h, the object a is replaced by a new membrane, with the label h_1 and the contents as indicated by u. Thus, by applying a membrane creation rule there is no replication of objects into the new membrane, which becomes a daughter of the original membrane (the depth of the membrane structure can increase).

In [2], cell-like P systems with membrane creation are introduced in the framework of polarizationless P systems with active membranes, in such manner that membrane creation rules can be applied in a maximally parallel way. A uniform and linear-time solution to the `Subset Sum` problem was given by means of a family of such kind of membrane systems. This paper analyzes, from a complexity perspective, the behavior of polarizationless P systems with active membranes and membrane creation rules when the semantics is a natural extension of the classical polarizationless P systems with active membranes, that is, in particular membrane, creation rules can be applied at most once on each membrane at each computation step. By using this new semantics, families of polarizationless P systems with active membranes and membrane creation rules, only problems in class **P** can be solved in a uniform and polynomial time. Therefore, a new frontier of the efficiency, expressed in terms of the semantics, is obtained.

The paper is organized as follows: first the methodology to attack the **P** versus **NP** problem by means of the concept of frontier between the tractability and the presumed intractability is recalled in the next section. In section 3 polarizationless P systems with active membranes and membrane creation are presented, and two different semantics (called maximalist and minimalist) are considered, and previous results are presented. Section 4 is devoted to show that only problems in class **P** can be solved by families of polarizationless P systems with active membranes and membrane creation only for elementary membranes, when minimalist semantics is considered. Finally, some conclusions and open problems are presented.

2 A new technique to tackle the P versus NP problem

Each computing model provides a mathematical definition of the informal idea of solving abstract problems by means of a mechanical procedure or *algorithm*. A computing model which is equivalent in power to Turing machines is called *universal*.

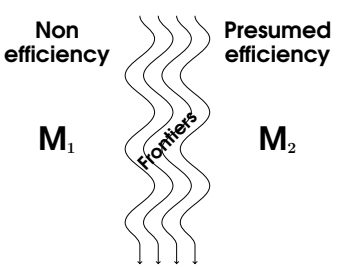
An abstract problem is said to be *tractable* if it can be solved by a deterministic Turing machine working in polynomial time (the upper bound of computational resources is polynomial). The complexity class of tractable decision problems is denoted by **P**. An abstract problem is said to be *intractable* if it

cannot be solved by a deterministic Turing machine working in polynomial time (the lower bound of the computational resources is exponential).

A computing model with the ability to provide polynomial-time solutions to intractable problems is called *efficient*. In a *non-efficient* computing model, only tractable problems can be solved in polynomial time. It is widely believed that $\mathbf{P} \neq \mathbf{NP}$, so **NP**-complete problems (the hardest problems in class **NP**) are commonly considered as *presumably intractable* problems. A computing model with the ability to provide polynomial-time solutions to **NP**-complete problems is called *presumably efficient*.

Given two computing models M_1 and M_2 we say that that M_1 is a *submodel* of M_2, denoted by $M_1 \subseteq M_2$, if each solution of a problem in M_1 is also a solution in M_2, that is, M_2 is an *extension* of M_1 in the sense that M_2 can be obtained from M_1 by adding some syntactic or semantic ingredients. If M_1 is a non-efficient computing model and M_2 is a presumably efficient one such that $M_1 \subseteq M_2$, then the (syntactical or semantic) ingredients allowing to pass from M_1 to M_2 provide a frontier between the efficiency and the presumed efficiency, that is, passing from M_1 to M_2 amounts to passing from being able to solve only tractable problems to solve presumably intractable problems. Therefore, it gives us a novel tool to tackle the **P** versus **NP** problem as follows:

- In order to show that $\mathbf{P} = \mathbf{NP}$, it is enough to find a polynomial-time solution to one **NP**-complete problem in M_2 and translate it to a polynomial-time solution in M_1, that is, the ingredients added to obtain M_2 from M_1 do not play a relevant role in that solution.
- In order to show that $\mathbf{P} \neq \mathbf{NP}$, it is enough to find one **NP**-complete problem that cannot be solved efficiently in M_1, that is, that the ingredients added to obtain M_2 from M_1 are crucial to obtain the presumed efficiency.



In this paper, a new frontier between the efficiency and the presumed non-efficiency is obtained in terms of semantics ingredients of polarizationless P systems with active membranes and membrane creation.

3 Polarizationless P systems with active membranes and membrane creation

In this section, membrane creation rules are considered in the framework of (cell-like) polarizationless P systems with active membranes, and a new semantics

different from the one given in [2] and [4] is defined and studied from a complexity point of view.

3.1 Syntax

A *polarizationless P systems with active membranes and membrane creation* of degree $q \geq 1$ is a tuple of the form $\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$ where:

1. Γ and H are finite alphabets such that $\{1, \ldots, q\} \subseteq H$;
2. μ is a rooted tree with q nodes bijectively labelled with $\{1, \ldots, q\}$;
3. $\mathcal{M}_1, \ldots, \mathcal{M}_q$ are multisets over Γ;
4. $\mathcal{R} = \bigcup_{h \in H} \mathcal{R}_h$ is a finite set of *rewriting rules* over Γ such that for each $h \in H$ the set $\mathcal{R}_h$ consists of the following types:
   (a) $[a \to u]_h$, where $a \in \Gamma$ and u is a multiset over Γ (*object evolution rules*);
   (b) $a[\,]_h \to [b]_h$, where $h \neq 1$ and $a, b \in \Gamma$ (*send-in communication rules*);
   (c) $[a]_h \to [\,]_h\, b$, where $a, b \in \Gamma$ (*send-out communication rules*);
   (d) $[a]_h \to b$, where $h \in H \setminus \{1, i_{out}\}$ and $a, b \in \Gamma$ (*dissolution rules*);
   (e) $[a \to [\, u\,]_{h_1}\,]_h$, where $a \in \Gamma$, $h_1 \in H \setminus \{1, i_{out}\}$ and u is a multiset over Γ (*creation rules*).
5. $i_{out} \in \{1, \ldots, q\} \cup \{env\}$.

A polarizationless P system with active membranes and membrane creation of degree $q \geq 1$, $\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$, can be viewed as a set of q membranes, labelled by $1, \ldots, q$, arranged in a hierarchical structure μ given by a rooted tree whose root (labelled by 1) is called the *skin membrane*, such that: (a) $\mathcal{M}_1, \ldots, \mathcal{M}_q$ represent the finite multisets of objects initially placed into the q membranes of the system; (b) $\mathcal{R}$ is a finite set of rewriting rules over Γ ($\mathcal{R}_h$ is associated with each membrane labelled by $h \in H$); and (c) i_{out} represents a distinguished *region* which will encode the output of the system. We use the term *region* i to refer to membrane i in the case $1 \leq i \leq q$ and to refer to the environment in the case $i = env$.

3.2 Semantics

The concept of *applicability* of a rule of the type $(a), (b), (c), (d)$ is defined as usual in polarizationless P systems with active membranes. With respect to membrane creation rules, a rule of the type $[a \to [\ u\ ]_{h_1}\ ]_h$ where $h, h_1 \in H$, $a \in \Gamma$ is applicable to a configuration $\mathcal{C}_t$ at an instant $t \geq 0$ if the following holds: (a) there exists a membrane labelled by h (different of the skin membrane) at $\mathcal{C}_t$ such that contains a copy of object a; and (b) h_1 is not the label of the skin membrane. When applying such a rule, under the influence of object a, a new membrane with label h_1 having inside the multiset u, is created in such manner that it is a daughter of the membrane labelled by h.

In [2] and [4] the semantics considered (called *maximalist*) is defined in such a way that the rules of the system are applied according to the following principles:

- The rules associated with membranes labelled with h are used for all copies of this membrane.
- At one transition step, one object of a membrane can be used by only one rule (chosen in a non-deterministic way).
- At one transition step, *a membrane* can be subject of several rules of types (a), (b), (c) and (e), that is, these rules are applied in a maximally parallel manner.
- At one transition step, *a membrane* can be subject of *at most one* rule of type (d), that is, dissolution rules can be applied at most once on each membrane at each computation step.
- Rules of types (a), (b), (c) and (e) can be simultaneously applied to a membrane joint with one rule of type (d). In that case, object evolution rules are applied in a maximally parallel manner.
- If at the same time a membrane labelled by h is dissolved and there are objects in this membrane which can evolve by other types of rules, then we suppose that first rules of types (a), (b), (c) and (e) are used, changing the objects and creating new membranes, and then the dissolution is produced. This process takes only one computation step.
- The skin membrane and the output membrane (if any) can never dissolve and any membrane can be created with label $1, i_{out}$.

In this paper, a new semantics (called *minimalist*) is considered, following the usual semantics of polarizationless P systems with active membranes and membrane division, where the role of membrane division rules is played by membrane creation rules. Specifically, the rules of the system are applied according to the following principles:

- The rules associated with membranes labelled with h are used for all copies of this membrane.
- At one transition step, one object of a membrane can be used by only one rule (chosen in a non–deterministic way).
- At one transition step, *a membrane* can be subject of *at most one* rule of types (b), (c), (d) or (e), that is, rules of these types can be applied at most once on each membrane at each computation step.
- Object evolution rules can be simultaneously applied to a membrane joint with one rule of types (b), (c), (d) or (e). In that case, object evolution rules are applied in a maximally parallel manner.
- If at the same time a membrane labelled by h is dissolved and there are objects in this membrane which can evolve by object evolution rules, then we suppose that first the evolution rules are used, changing the objects, and then the dissolution is produced. Of course, this process takes only one computation step.
- The skin membrane and the output membrane can never dissolve.

The difference between the two semantics lies in how rules are applied: in the first case, rules of types (b), (c) and (e) are applied in a maximally parallel way,

while in the second one these rules (together with rules of type (d)) are applied in a sequential manner (at most once on each membrane at each computation step).

It is worth pointing out that the concept of elementary membrane is dynamic, in the sense that by applying a membrane creation rule to an elementary membrane, it becomes non-elementary.

3.3 Recognizer P systems with membrane creation

Let us recall that recognizer P systems (introduced in [12]) are the natural framework to study and solve decision problems since deciding whether an instance has an affirmative or negative answer is equivalent to deciding if a string belongs or not to the language associated with the problem. These systems have an input membrane and the output region is the environment, the working alphabet has two distinguished objects `yes` and `no`, all the computations of the system halt and for each computation, either object `yes` or object `no` (but not both) must have been released into the output zone (the environment), and only at the last step of the computation.

In this paper, $\mathcal{CAM}^0_\alpha(\beta,\gamma)$ denotes the class of all polarizationless P systems with active membranes and with creation rules, where α,β,γ are parameters associated with the kind of semantics, dissolution rules and membrane creation rules, respectively. The meaning of these parameters is the following:

- If $\alpha = max$ (resp. $\alpha = min$) the maximalist (resp. minimalist) semantics is considered.
- if $\beta = -d$ (resp. $\beta = +d$) then dissolution rules are forbidden (resp. permitted).
- if $\gamma = -ne$ (resp. $\gamma = +ne$) then membrane creation rules are restricted to elementary membranes only (resp. membrane creation rules for elementary and non-elementary membranes are permitted).

Previous results

In [2] a uniform and linear-time solution to the `Subset Sum` problem is provided by using a family of polarizationless P systems with active membranes and membrane creation with the maximalist semantics and allowing membrane creation rules only for elementary membranes, that is, we have the following result:

Theorem 1. $\mathbf{NP} \cup \mathbf{co\text{-}NP} \subseteq \mathbf{PMC}_{\mathcal{CAM}^0_{max}(+d,-ne)}$

Moreover, in [4], a uniform and linear-time solution to the `QSAT` problem is provided by using a family of polarizationless P systems with active membranes and membrane creation with the maximalist semantics and allowing membrane creation rules only for elementary membranes, that is, we have the following result:

Theorem 2. $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\mathcal{CAM}^0_{max}(+d,-ne)}$

4 A semantic frontier of tractability

The goal of this section is to provide a new borderline between the non-efficiency and the presumed efficiency in terms of the semantics used by polarizationless P systems with active membranes and membrane creation. Specifically, we show that by using the minimalist semantics, only problems in class **P** can be solved by families of polarizationless P systems with active membranes and membrane creation only for elementary membranes.

In [3] it is shown that $\mathbf{PMC}_{\mathcal{T}} \subseteq \mathbf{P}$, being $\mathcal{T}$ the class of basic recognizer transition P systems. Specifically, if X is a decision problem such that $X \in \mathbf{PMC}_{\mathcal{T}}$ and $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ is a family of systems from $\mathcal{T}$ solving X in a uniform and polynomial time, then for each $n \in \mathbb{N}$, a deterministic Turing machine, $M(n)$, *with multiple tapes*, working in polynomial time is constructed in such manner that given an input multiset $\mathcal{M}$ of $\Pi(n)$, the machine reproduces (only) *one* specific computation of $\Pi(n)$ with input $\mathcal{M}$. Specifically, machine $M(n)$ has one *input tape*, that keeps a string representing the input multiset received as well as:

- One *structure tape* encoding the current membrane structure.
- For each object of the working alphabet of the system: (a) one *main tape*, that keeps the multiplicity of the object, in binary format, in the multiset contained in the membrane; and (b) one *auxiliary tape*, that keeps temporary results of applying the rules associated with the membrane.
- One *rules tape* that encodes the applicability of rules associated with different membranes, assuming a total order in the set of rules.
- For each object we have one *environment tape* that keeps its multiplicity, in binary, in the multiset associated with the environment.

The key of this construction lies in the fact that at any configuration of the system, the total number of membranes is polynomially bounded for the size of $\Pi(n)$. Next, we show that systems from $\mathcal{CAM}^0_{min}(+d, -ne)$ verify the cited property.

In order to prove this, we previously obtain some results and introduce some new notations.

Proposition 1. *Let Π be a membrane system from $\mathcal{CAM}^0_{min}(+d, -ne)$ of degree q and with e elementary membranes. Let $\mathcal{C} = (\mathcal{C}_0, \ldots, \mathcal{C}_r), r \in \mathbb{N}$, be a halting computation of Π. Then we have:*

(a) For each $t, 1 \leq t \leq r$, at configuration $\mathcal{C}_t$ the number of elementary membranes is at most e.
(b) For each $t, 1 \leq t \leq r$, at configuration $\mathcal{C}_t$ the total number of membranes is at most $(t+1) \cdot q$.

Proof. (a) It suffices to note that a new elementary membrane with label h_1 can appear at configuration $\mathcal{C}_{t+1}$ when a membrane creation rule of the form $[\ a \longrightarrow [\ u\ ]_{h_1}\ ]_h$ is applied to an elementary membrane labelled by h at configuration $\mathcal{C}_t$. But in this case, the membrane where the rule is applied ceases to be

elementary. Therefore, bearing in mind that in the minimalist semantics at most one membrane creation rule can be applied to each elementary membrane, the total number of elementary membranes at configuration $\mathcal{C}_{t+1}$ does not increase with respect to configuration $\mathcal{C}_t$

(b) By induction on t. For the base case $t = 1$, let us note that the total number of membranes (resp. elementary membranes) at the initial configuration $\mathcal{C}_0$ is q (resp. e) and, at most, e new membranes can be created by the application of membrane creation rules. Thus, by means of the application of membrane creation rules to elementary membranes (by using the minimalist semantics), at most e new membranes can be created. Therefore, the total number of membranes at configuration $\mathcal{C}_1$ is at most $q + e \leq 2 \cdot q$.

Let $t, 1 \leq t < r$, and let us assume by induction hypothesis that the total number of membranes at configuration $\mathcal{C}_t$ is at most $q+te$. Bearing in mind that the total number of elementary membranes at configuration $\mathcal{C}_t$ is at most e, by means of the application of membrane creation rules to elementary membranes (by using the minimalist semantics), at most e new membranes can be created. Thus, the total number of membranes at configuration $\mathcal{C}_{t+1}$ is at most $q+te+e = q + (t+1) \cdot e \leq (t+2) \cdot q$.

According with the previous remark we have the following:

Proposition 2. $\mathbf{PMC}_{\mathcal{CAM}^0_{min}(+d,-ne)} \subseteq \mathbf{P}$.

On the other hand, for each non-empty recognizer membrane system class $\mathbf{R}$ we have $\mathbf{P} \subseteq \mathbf{PMC}_{\mathbf{R}}$. Indeed, if $X = (I_X, \theta_X)$ is a decision problem in class $\mathbf{P}$ and M is a deterministic Turing machine working in polynomial time solving X, then it suffices to consider the family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of systems from $\mathcal{R}$, where system $\Pi(1)$ has only one membrane (labelled by 1), its working alphabet is $\Gamma = \{\mathtt{yes}, \mathtt{no}\}$, and the set of rules is $[\ \mathtt{yes}\ ]_1 \rightarrow \mathtt{yes}\ [\ ]_1$ and $[\ \mathtt{no}\ ]_1 \rightarrow \mathtt{no}\ [\ ]_1$. The polynomial encoding (cod, s) from X to $\mathbf{\Pi}$ is defined as follows: for each instance $u \in I_X$, $s(u) = 1$ and $cod(u) = \mathtt{yes}$ (resp. $\mathtt{no}$) if $M(u)$ is an accepting computation (resp. rejecting computation), certifying that $X \in \mathbf{PMC}_{\mathbf{R}}$.

Thus, we have the following:

Theorem 3. $\mathbf{PMC}_{\mathcal{CAM}^0_{min}(+d,-ne)} = \mathbf{P}$.

From Theorems 1 and 3 we deduce that passing from mimimalist semantics to maximalist semantics in the framework of polarizationless P systems with active membranes and membrane creation, amounts to passing from non-efficiency to presumed efficiency.

5 Conclusions

The ability of a computing model for solving intractable problems in an efficient way provides its computational efficiency. Different uniform and polynomial-time solutions to many **NP**-complete problems or **PSPACE**-complete problems have been given by using families of membrane systems. Also, the limits of the

efficiency of some models in membrane computing have been established. In this context, some syntactical ingredients have been identified to be the responsible to passing from non-efficiency (only problems in class **P** can be solved in an efficient manner) to the presumed efficiency (**NP**-complete problems can be solved in polynomial time).

In this paper, a new frontier expressed in terms of semantic ingredient is presented in the framework of polarizationless P systems with active membranes and membrane creation. Specifically, passing from applying membrane creation rules in sequential manner to applying them in a maximally parallel manner, amounts to passing from non-efficiency to presumed efficiency.

Finally, it would be interesting to study new semantics in the framework of $\mathcal{CAM}^0(\beta, \gamma)$, as well as to analyze the computational efficiency of membrane systems from $\mathcal{CAM}^0_{min}(+d, +ne)$, $\mathcal{CAM}^0_{min}(-d, +ne)$ and $\mathcal{CAM}^0_{max}(-d, -ne)$, in the attempt to find new frontiers, possibly optimal ones (based on the inclusion or not of a single ingredient).

Acknowledgements

Authors acknowledge the support of the research project TIN2017-89842-P, 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.

This work was also supported by National Natural Science Foundation of China (Grant No. 61320106005).

References

1. A. Alhazov, T.-O. Ishdorj. Membrane operations in P systems with active membranes. In: Proceedings of the Second Brainstorming Week on Membrane Computing, Sevilla, 2-7 February 2004, 37-44.
2. M.Á. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: A linear solution of Subset Sum problem by using membrane creation. *Lecture Notes in Computer Science*, **3561** (2005), 258-267.
3. M.Á. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero, Á. Romero-Jiménez: Characterizing tractability by cell-like membrane systems. In K.G. Subramanian, K. Rangarajan, M. Mukund (eds.) *Formal models, languages and applications*, World Scientific, Singapore, 2006, pp. 137-154.
4. M.Á. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero. A linear time solution for QSAT with membrane creation. *Lecture Notes in Computer Science*, **3850** (2006), 241-252.
5. P.L. Luisi. The Chemical Implementation of Autopoiesis, *Self-Production of Supramolecular Structures* In G.R. Fleishaker et al. (eds.), Kluwer, Dordrecht, 1994.
6. C. Martín-Vide, Gh. Păun, A. Rodríguez-Patón. On P Systems with Membrane Creation. *Computer Science Journal of Moldova*, **9**, 2(26) 2001, 134-145.
7. M. Mutyam, K. Krithivasan: P systems with membrane creation: Universality and efficiency. *Lecture Notes in Computer Science*, **2055** (2001), 276-287.

8. Pan, L.; Ishdorj, T.-O. P systems with active membranes and separation rules. *Journal of Universal Computer Science*, **10**, 5 (2004), 630-649.
9. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1 (2000), 108-143, and *Turku Center for CS-TUCS Report* No. 208, 1998.
10. Gh. Păun. P systems with active membranes: attacking **NP**-complete problems, *Journal of Automata, Languages and Combinatorics*, **6**, 1 (2001), 75-90.
11. Gh. Păun, M.J. Pérez-Jiménez, Gr. Rozenberg. Spike trains in spiking neural P systems. *International Journal of Foundations of Computer Science*, **17**, 4 (2006), 975-1002.
12. M.J. Pérez-Jiménez, Á. Romero-Jiménez, F. Sancho-Caparrini: Complexity classes in cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265-285.
13. G. Zhang, M.J. Pérez-Jiménez, M. Gheorghe. *Real-Life Modelling with Membrane Computing*. Series: Emergence, Complexity and Computation, Volume 25. Springer International Publishing, 2017, X + 367 pages.

The Application of Weighted Spiking Neural P Systems with Rules on Synapses for Breaking RSA Encryption

Huifang Wang[1], Kang Zhou[1] ⋆, Gexiang Zhang[2], Prithwineel Paul[2], Yingying Duan[2], and Huaqing Qi[3]

[1] School of Math and Computer,
Wuhan Polytechnic University, Wuhan 430023, China
[2] School of Electrical and Engineering,
Southwest Jiaotong University, Chengdu 610031, China
[3] School of Economics and Management,
Wuhan Polytechnic University, Wuhan 430023, China
{13667286986@139.com,zhoukang65@whpu.edu.cn}

Abstract. RSA algorithm is one of the most widely used public key encryption algorithm. Breaking RSA encryption is considered very difficult and is well-known as RSA problem. The difficulty of this problem lies in prime factorization of large integers. In this paper, WSN P systems with rules on synapses are constructed to factorize large integers to study the breaking of RSA encryption. Also, we show that WSN P systems with rules on synapses can factorize large integers in linear time.

Keywords: SN P systems, WSN P systems with rules on synapses, RSA algorithm

1 Introduction

RSA algorithm [1] was proposed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman at Massachusetts Institute of Technology. It is one of the most influential public key encryption algorithm [2]. The security of this algorithm depends on the prime factorization of large integers which is a very well-known difficult mathematical problem [3]. To ensure the security of RSA algorithm, the length of keys in the algorithm is continuously increasing and along with that the difficulty of cracking RSA encryption is also increasing.

For many years cryptographers are trying to break RSA encryption. In recent years several attempts have been made to break it using unconventional computing models. The quantum algorithm for breaking RSA encryption was initiated by Shor in [4], where an efficient algorithm for factorization of large integers was introduced. A new quantum algorithm to break RSA encryption based on phase estimation and equation solving has been discussed in [5]. The distributed factorization computation to break RSA encryption was introduced in [6].

⋆ Corresponding author.

Furthermore, factoring large integers using DNA computing was expounded in [7].

Membrane computing was initiated by Păun in [8], and the complexity of membrane computing models has been discussed in [9]. The distributed and parallel computing devices in membrane computing are called P systems [10]. These models are powerful and different variants of P systems have been used to solve computationally difficult problems. Since, the parallel and distributed architecture of P systems can provide exponential workspace, these models have been widely used to solve the **NP**-complete, **PSPACE** problems in polynomial as well as in linear time [11–18]. Hence the membrane computing models are a suitable choice for solving prime factorization of large integers.

In [19], a RSA algorithm based on P systems has been discussed. However, how to use the P system to break RSA encryption has not been considered before. Spiking neural P systems (i.e., SN P systems) are a variant of P systems, which were introduced in [20] as a new biological computing device. SN P systems have strong computing power and have the potential to solve difficult problems of computing [21]. Various variants of SN P systems were constructed by considering different biological sources [22–29], and the computing power, efficiency etc. of these variants were discussed in [30–35]. Also, SN P systems have wide range of applications [36] in optimization [37], fault diagnosis [38, 39] and logic gate [40]. Moreover, the distributed and parallel characteristics of SN P systems were used to solve the directional Hamiltonian path problem [41], implementation of the sorting function [42, 43], and simulation of Boolean circuits [43]. In [44], the high flexibility of weighted spiking neural P systems with rules on synapses (i.e., WSN P systems with rules on synapses) [35] is used to optimize the addition and multiplication systems in [45] effectively, by reducing the number of neurons.

In this paper a WSN P system with rules on synapses is constructed to realize the factorization of large integers using the input module, random number module, multiplication module, comparison module, and output module. These modules explain the working of the system in detail. The parallelism of the SN P systems is used to construct the system and the factorization modules working in parallel help the system to factorize a large integer. The input of the system is a natural number expressed in binary form, which is further encoded as appropriate sequence of spikes. The output neurons also emit the computed numbers to the environment in binary forms which are also encoded as spike trains.

This paper is organized as follows. Section 2 briefly introduces the WSN P systems with rules on synapses. In Section 3, we discuss the structure of WSN P systems with rules on synapses for large integer factorization and the detailed implementation of each module is given. Section 4 verifies the correctness and feasibility of the system through examples, and finally analyzes the system complexity. Conclusion and future work are presented in Section 5.

2 WSN P Systems with Rules on Synapses

In this section, WSN P systems with rules on synapses are described. For more details on SN P systems we refer to [20, 21].

A weighted spiking neural P system of degree $m \geq 1$ with rules on synapses [24] is a construct of the form:

$$\prod = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, \sigma_{in}, \sigma_{out}),$$

where:

(1) $O = \{a\}$ is the singleton alphabet (a is called spike);
(2) $\sigma_1, \sigma_2, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (n_i)$, with $1 \leq i \leq m$, where n_i is initial number of spikes in neuron σ_i;
(3) syn is the set of synapses between neurons. Each element in syn is a pair of the form $((i,j), w_{ij}, R_{(i,j)})$, where (i,j) is a synapse from neuron σ_i to neuron σ_j, $i \in \{1,2,\ldots,m\}, j \in \{1,2,\ldots,m,en\}, i \neq j$, w_{ij} is a positive integer representing the weight of synapse (i,j), and $R_{(i,j)}$ is a finite set of rules on synapse (i,j) of the following forms:
   (a) $E/a^c \rightarrow a^p; d$, where E is a regular expression over O, $c \geq p \geq 1$ and $d \geq 0$. The rule also can be written as $a^c \rightarrow a^p$ when $E = a^c$. Also the rule is called a *standard rule* when $p = 1$ and *extended* when $p > 1$;
   (b) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^c \rightarrow a^p; d$ from any $R_{(i,j)}$;
(4) σ_{in} and σ_{out} indicate the input neuron and the output neuron respectively.

If $d = 0$ and $p = 1$ in the rule $E/a^c \rightarrow a^p; d$, then the rule can be simply written as $E/a^c \rightarrow a$. A rule of the form $a^s \rightarrow \lambda$ is called a forgetting rule.

A firing rule $E/a^c \rightarrow a^p; d$ on synapse (i,j) with weight w_{ij} is applied as follows. If $E/a^c \rightarrow a^p; d \in R_{(i,j)}$, and neuron σ_i contains k spikes such that $a^k \in L(E)$, $k \geq c$, then the rule is enabled. This means c spikes (thus only $k - c$ spikes remain in σ_i) are consumed (removed) from neuron σ_i, and p spikes are produced. Theses p spikes are multiplied w_{ij} times by the weight w_{ij} of the synapse (i,j) in the process of transmission, and then reach neuron σ_j after d time units. If $d = 0$, then $w_{ij}p$ spikes immediately reach the neuron σ_j. If the rule is used in step t and $d \geq 1$, then in steps t,$t+1$,...,$t+d-1$, the synapse (i,j) can not use any rules. In the step $t+d$, neuron σ_j receives $w_{ij}p$ spikes, and in the step $t+d+1$, the synapse (i,j) can apply the rules again.

When neuron σ_i contains exactly s spikes, then forgetting rule $a^s \rightarrow \lambda \in R_{(i,j)}$ is enabled. By using it, $s \geq 1$ spikes are removed from the neuron σ_i.

As usual in SN P systems, a global clock is assumed, marking the time for all neurons and synapses. In each time unit, if a synapse (i,j) can use one of its rules, then a rule from $R_{(i,j)}$ must be used. Since two firing rules, $E_1/a^{k_1} \rightarrow a; d_1$ and $E_2/a^{k_2} \rightarrow a; d_2$, can have $L(E_1) \cap L(E_2) \neq \phi$, it is possible that there are more than one rule that can be used on synapse at some moment. In this case, the synapse will non-deterministically choose one of the enabled rules to use.

The initial configuration of the system is identified by the numbers $n_1, n_2, \ldots,$ n_m of spikes present in each neuron. During the computation, configuration of the system is described by the number of spikes present in each neuron. $< r_1, r_2, \ldots, r_m >$ represents the configuration of the system at any time instance where neuron σ_i contains $r_i \geq 0$ spikes, $i = 1, \ldots, m$. The initial configuration is $C_0 =< n_1, n_2, \ldots, n_m >$ where $n_1, n_2, \ldots, n_m$ represent the number of spikes present in the respective neurons initially. Using the rules as described above, we can define transitions. Any series of transitions starting from the initial configuration is called a computation. A computation is successful if it reaches a configuration where no rule can be applied on any synapse. The result of a computation of the system is defined as the number of spikes sent to the environment by the output neuron.

We use a circle with the initial number of spikes inside to represent a neuron and the directed edge associated with rules and weights to represent the synapse. The input/output neuron has an incoming/outgoing arrow, suggesting their communication with the environment.

When the standard spiking rules on more than one synapses are emitted by a neuron at the same time, the rules are going to work in the following manner:

(1) The number of spikes consumed by each rule on different synapses must be equal;
(2) The sum of the number of spikes consumed by each rule on different synapses must be less than or equal to the number of spikes contained in the neuron.

3 Large Integer Factorization with WSN P Systems with Rules on Synapses

In RSA algorithm the public key consists of a large integer N (modulus) and public exponent e. The private key consists of the modulus N and private exponent d.

The RSA public-key / private-key pair can be obtained in the following manner:

- Generate a pair of large, random primes P and Q.
- Compute the modulus N where $N = PQ$ and $\phi(N) = (P-1)(Q-1)$.
- Select an odd public exponent e between 3 and $N-1$ relatively prime to both $P-1$ and $Q-1$ (i.e., relatively prime to $\phi(N)$).
- Compute the private exponent d from $e.d = 1(\text{mod } \phi(N))$.
- Output (N, e) as the public key and (N, d) as the private key.

From the above algorithm, it is clear that if the prime factorization of the large integer is known, the private key (N, d) can be obtained easily. Hence, it will be possible to break RSA encryption. In this work, we use the WSN P systems with rules on synapses to perform the large integer factorization. With the high parallelism of SN P systems, all possible P and Q can be tested in parallel through hundreds of millions of neurons participating in the computation. With

the increase of the number of parallel structures, the probability of obtaining the correct solution will approach to 1. Moreover, because of the space-time tradeoff, RSA problem can be solved in linear time.

We construct an SN P system with rules and weights on synapses as shown in Fig. 1 to perform the large integer factorization.

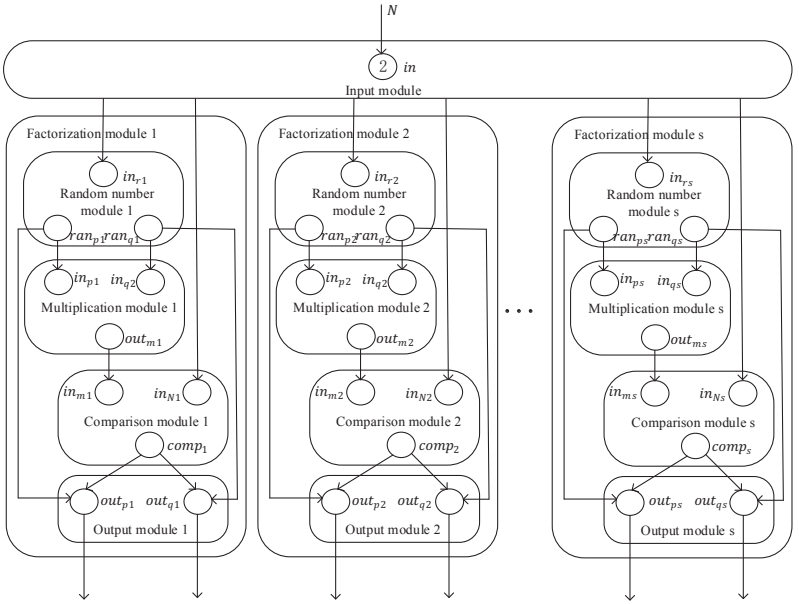


Fig. 1. The WSN P system with rules on sysnapses $\prod_{RSA}$ for the large integer factorization

As shown in Fig. 1, the system $\prod_{RSA}$ consists of an input module and s parallel factorization modules. The number of neurons is massive, in fact s is an exponential number. So it is enough to have at least one parallel structure that can find the correct P and Q. Each factorization module also includes a random number module to generate random numbers, a multiplication module to compute product, a comparison module to compare the product with the large integer N, and an output module to collect the output of the correct P and Q.

The input of this system is a large integer N expressed in binary form, which is further encoded as an appropriate sequence of spikes. After receiving the input N, the input module sends the spike signal to the random number module in each factorization module after a series of processing, and sends the large integer N to the comparison module in each factorization module after a certain delay to compare it with the product. After receiving spikes, the neuron σ_{in_r} in random number module starts to generate random numbers P and Q, and sends them

to the two input neurons σ_{in_p} and σ_{in_q} of the multiplication module to calculate the product. The generated random numbers P and Q are sent to σ_{out_p} and σ_{out_q} of the output module after a certain time delay for preprocessing of output. The product calculated by multiplication module is sent to the comparison module through its output neuron σ_{out_m}. When the input neuron σ_{in_m} of the comparison module starts to receive the product, the another input neuron σ_{in_N} starts to receive N sent from the input module (by setting a certain delay, N and the product can arrive at the comparison module at the same time). After comparing each bit of two binary numbers, neuron σ_{out_p} of the output module receives the spike train of P sent from the random number module, and σ_{out_q} receives the spike train of Q. Both P and Q are processed in σ_{out_p} and σ_{out_q} respectively according to the comparison results. If each bit of the product and N is equal, the spike trains of P and Q are sent to the environment. If there are unequal bits, then both P and Q are discarded. In the s factorization modules, two prime divisors are successfully found and sent by several factorization modules. But only P and Q that can factorize the large integer N will be obtained by the factorization modules.

Fig. 1 shows the overall structure of the system $\prod_{RSA}$, and its specific workflow. The detailed implementation process of each module with WSN P systems with rules on synapses is given in the following subsections.

3.1 Input Module

The input module of Fig. 1 implemented by the WSN P system with rules on synapses is shown in Fig. 2.

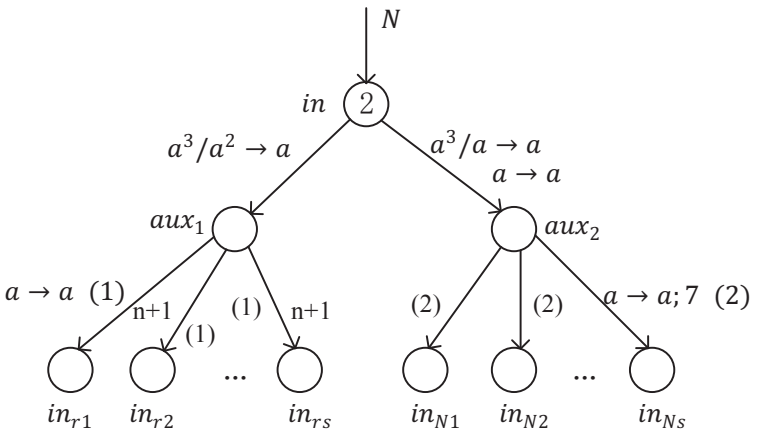


Fig. 2. The input module of $\prod_{RSA}$

The input neuron σ_{in} contains 2 spikes initially. After the system $\prod_{RSA}$ starts working, neuron σ_{in} receives the spike train of the large integer N which is a product of two large prime numbers. Since P and Q are odd, N must be an odd number. Since N is odd , its lowest bit input is 1, and there are 3 spikes in neuron σ_{in} at time $t = 1$. The rule $a^3/a^2 \to a$ on synapse (in, aux_1) and the rule $a^3/a \to a$ on synapse (in, aux_2) are activated at the same time, sending 1 spike to neurons σ_{aux_1} and σ_{aux_2} respectively. If there is 1 in the remaining bits of N, it is directly sent to σ_{aux_2} through the rule $a \to a$ on the synapse (in, aux_2).

At time $t = 2$, both neurons σ_{aux_1} and σ_{aux_2} contain 1 spike. Then the same rule $a \to a$ on synapses (aux_1, in_{r1}), (aux_1, in_{r2}), ..., (aux_1, in_{rs}) emitted by σ_{aux_1} are enabled at the same time, and $n + 1$ spikes are sent to the input neurons $\sigma_{in_{r1}}$, $\sigma_{in_{r2}}$, ..., $\sigma_{in_{rs}}$ of the corresponding random number modules (the weight on synapse is $n + 1$). The rule $a \to a; 7$ on synapses (aux_2, in_{N1}), (aux_2, in_{N2}), ..., (aux_2, in_{Ns}) emitted by σ_{aux_2} are also enabled at the same time, and 1 spike is sent to the input neurons $\sigma_{in_{N1}}$, $\sigma_{in_{N2}}$, ..., $\sigma_{in_{Ns}}$ of each comparison module after 7 time units, respectively. In the next moment, σ_{aux_1} will no longer receive spikes, and σ_{aux_2} will continue to work until the input ends.

The neuron σ_{aux_1} is connected with the input neurons of each random number module, and it sends spikes to start each factorization module. The neuron σ_{aux_2} is connected with the input neurons of each comparison module, to compare the product with the given large integer N.

3.2 Random Number Module

According to the structure of the large integer factorization system $\prod_{RSA}$ in Fig. 1, we can see that P and Q need to be given before calculating the product. Assume that the valid bit length (the valid bit length of 01001_2 is 4) of P is k_1, and the valid bit length of Q is k_2 where $k_1 \leq k_2$. The general multiplication mode is shown in Table 1.

Table 1. the general binary multiplication mode

	n_0	n_1	n_2	...	n_{k_1-1}	...	n_{k_2-1}	...	$n_{k_1+k_2-2}$	$n_{k_1+k_2-1}$
p_{k_1-1}					$p_{k_1-1}q_0$	...	$p_{k_1-1}q_{k_2-k_1}$	...	$p_{k_1-1}q_{k_2-1}$	+
...					...	...	...			
p_2			p_2q_0	...	$p_2q_{k_1-3}$	...	$p_2q_{k_2-3}$			
p_1		p_1q_0	p_1q_1	...	$p_1q_{k_1-2}$	...	$p_1q_{k_2-2}$			
p_0	p_0q_0	p_0q_1	p_0q_2	...	$p_0q_{k_1-1}$	...	$p_0q_{k_2-1}$			
	q_0	q_1	q_2	...	q_{k_1-1}	...	q_{k_2-1}			

From Table 1, we can see that for P and Q whose valid binary bit lengths are k_1 and k_2 respectively, the valid binary bit length of the product is either k_1+k_2 (the highest bit has a carry) or k_1+k_2-1 (the highest bit has no carry). Then, for a given large integer N with a valid binary bit length n, the sum of the valid binary bit length of its prime factors P and Q must satisfy either $k_1+k_2=n$ or $k_1+k_2=n+1$. The values of each of the numbers produced in this module are random, so the highest bit of them is not necessarily 1. The case where the sum of the bit length of two random numbers is $n+1$ contains the case where the sum of the bit length is n. As a result, the input module sends $n+1$ spikes to each random module to generate P and Q after reading a large integer N.

The random number module of Fig. 1 implemented by the WSN P system with rules on synapses is shown in Fig. 3.

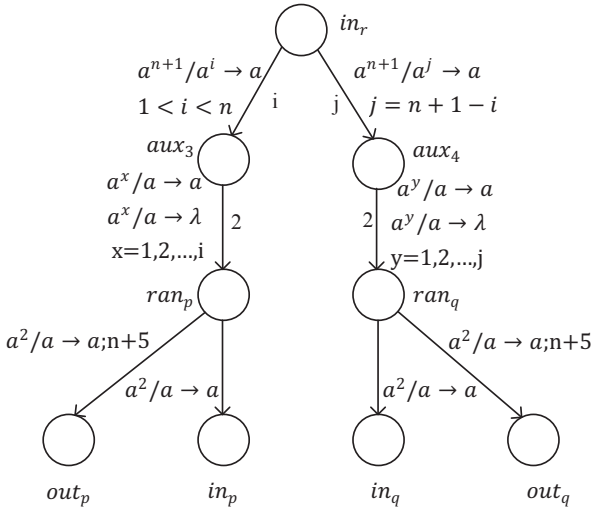


Fig. 3. The random number module of $\prod_{RSA}$

At time $t=3$, the input neuron σ_{in_r} of the random number module receives $n+1$ spikes sent by the input module, and these $n+1$ spikes are allocated to neurons σ_{aux_3} and σ_{aux_4} by using rules $a^{n+1}/a^i \to a$ and $a^{n+1}/a^j \to a$ on synapses (in_r, aux_3) and (in_r, aux_4), where $1<i<n$, $i+j=n+1$. Then at time $t=4$, σ_{aux_3} and σ_{aux_4} contain i and j spikes, respectively.

When neuron σ_{aux_3} contains any number of spikes, a rule on synapse (aux_3, ran_p) is selected non-deterministically. If the forgetting rule $a^x/a \to \lambda$ is selected, one spike will be consumed and no new spike will be generated, which means that the binary bit of P is 0. If the rule $a^x/a \to a$ is fired, one spike will be consumed and 2 spikes will be sent to the neuron σ_{ran_p} at the next time (weight is 2), indicating that the binary bit of P is 1. After receiving these 2 spikes, rule

$a^2/a \to a$ on synapse (ran_p, in_p) emitted by σ_{ran_p} is fired, and 1 spike is sent to the neuron σ_{in_p} at the next moment. The rule $a^2/a \to a; n+5$ on synapse (ran_p, out_p) is also fired and 1 spike is sent to neuron σ_{out_p} after $n+5$ steps.

Neuron σ_{aux_3} continues to select the rules non-deterministically until no spike in it remains to be consumed. The analysis of neuron σ_{aux_4} is the same as that of σ_{aux_3}.

3.3 Multiplication Module

The numbers P and Q are generated in random number module and are sent to the neurons in multiplication module. WSN P system with rules on synapses performs the multiplication module according to the figure shown in Fig. 4.

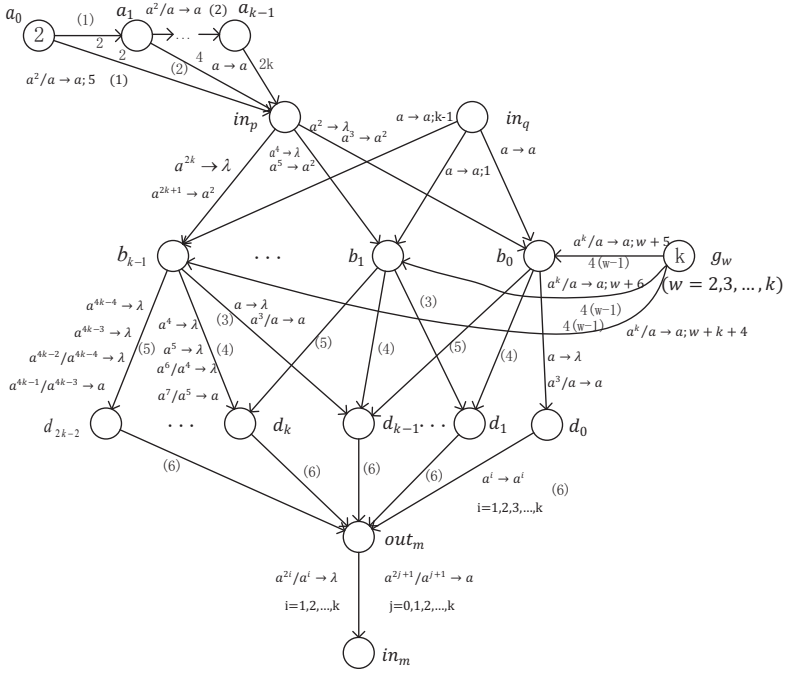


Fig. 4. The multiplication module of $\prod_{RSA}$

This module can calculate the product of two k bits binary numbers. Since the binary bit length of large integer N is n, the maximum effective binary bit length of P and Q is $n-1$. So the multiplication module of $k = n-1$ is selected in Fig. 1.

Assuming that P and Q are two natural numbers with k binary bits, and P and Q are rewritten as follows:

$P = \sum_{i=0}^{k-1} p_i 2^i, Q = \sum_{j=0}^{k-1} q_j 2^j$,then:

$$P \times Q = (\sum_{i=0}^{k-1} p_i 2^i) \times (\sum_{j=0}^{k-1} q_j 2^j)$$
$$= \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} p_i q_j 2^{i+j}$$
$$= q_0 p_0 2^0 + (q_0 p_1 + q_1 p_0) 2^1 + \ldots + (q_1 p_{k-1} + q_2 p_{k-2} + \ldots + q_{k-1} p_1) 2^k + \ldots + q_{k-1} p_{k-1} 2^{2k-2}$$

From the above expression we can see that the product of P and Q can be decomposed into the sum of $(2k-1)$ terms. The specific operation process of this module consists of the following parts:

(1) **Storing the bits of P**

At time $t = 6$, the input neurons of this module receive the spike trains of P and Q sent from the random number module. When each of the corresponding binary bits of P is input to σ_{in_p}, the auxiliary neuron σ_{a_i} sends $2(i+1)$ spikes to σ_{in_p}, where $i = 0, 1, ..., k-1$. Then different bits of this k-bit binary number are sent to different neurons according to different rules on synapses (in_p, b_0), (in_p, b_1), ..., (in_p, b_{k-1}).

The natural number P is input in the order from the digit which is associated with the power 2^0 to the digit which is associated with the power 2^{k-1}. At time $t = 6$, the digit which is associated with the power 2^0 in the binary representation of P is provided to neuron σ_{in_p} and at the same time the auxiliary neuron σ_{a_0} sends 2 spikes to σ_{in_p}. Now we can divide the future behavior of σ_{in_p} in two cases, depending on the number of spikes in it.

(a) If there are 2 spikes, then application of the rule $a^2 \to \lambda$ on synapse (in_p, b_0) will consume 2 spikes and no spike is sent out.

(b) If there are 3 spikes, then the rule $a^3 \to a^2$ on synapse (in_p, b_0) is triggered. As a consequence, two spikes are sent to neuron σ_{b_0} .

Thus, k binary bits of P can be stored in neurons σ_{b_0}, σ_{b_1}, ..., $\sigma_{b_{k-1}}$ by repeating the above operations, where the i-th binary bit of P is stored in neuron σ_{b_i}. If there are 2 spikes, it means the i-th binary bit of P is 1 and if there are no spikes, that means the i-th binary bit of P is 0, where $i = 0, 1, ..., k-1$.

(2) **Input the bits of Q**

At time $t = 6$, the digit which is associated with the power 2^0 in the binary representation of Q is provided to the neuron σ_{in_q}. If the digit is 1, then the rule $a \to a$ on synapse (in_q, b_0) is triggered, and neuron σ_{b_0} will receive one spike at next step. The rules $a \to a; i$ on other synapses (in_q, b_i) which is emitted by neuron σ_{in_q} are also triggered, and neurons σ_{b_i} receive one spike after i steps respectively, where $i = 0, 1, 2, ..., k-1$.

(3) **Obtaining the coefficients**

The number of spikes of each binary bit of P in neuron $\sigma_{b_i} (i = 0, 1, ..., k-1)$ is either 2 or 0. Hence, after receiving q_0 the number of spikes in the neuron σ_{b_i} can be $0, 1, 2$ and 3. So we have the following four cases.

(a) If σ_{b_i} contains 0 spikes, then no rule can be applied and no spikes are sent out. This encodes the operation $q_0p_i = 0 \times 0 = 0$.

(b) If σ_{b_i} contains 1 spike, then it is received from σ_{in_q}. The rule $a \to \lambda$ on synapse (b_i, d_i) consumes one spike and no spike is sent out. This encodes the operation $q_0p_i = 1 \times 0 = 0$.

(c) If σ_{b_i} contains 2 spikes, then all of them are received from σ_{in_p}. No rules can be applied further. This encodes the operation $q_0p_i = 0 \times 1 = 0$.

(d) If σ_{b_i} contains 3 spikes, then two of them are received from σ_{in_p} and the other one is received from σ_{in_q}. The rule $a^3/a \to a$ on synapse (b_i, d_i) is triggered and one spike reaches neuron σ_{d_i} after consuming one spike. It leaves two spikes in the neuron σ_{b_i} for the next step. This encodes the operation $q_0p_i = 1 \times 1 = 1$.

When q_1 reaches neurons σ_{b_0}, σ_{b_1}, ..., $\sigma_{b_{k-1}}$, these neurons receive four spikes from neuron σ_{g_2} (the rule $a^k/a \to a; 7+i$ on synapse (g_2, b_i) is activated at the initial time, and the one spike produced is multiplied by the weight 4 on synapses (g_2, b_i) during the transfer, and will reach σ_{b_i} after $7+i$ steps delay, where $i = 0, 1, \ldots, k-1$). Therefore, the number of spikes contained in neurons σ_{b_0}, σ_{b_1}, ..., $\sigma_{b_{k-1}}$ at this time can be $4, 5, 6$, and 7, respectively, corresponding to the four cases $0, 1, 2$, and 3. Hence, the rules on synapses (b_i, d_{i+1}) is ready to fire and the result q_1p_i is sent to neuron $\sigma_{d_{i+1}}$, where $i = 0, 1, ..., k-1$.

Similarly, we can get q_2p_i, ... , $q_{k-1}p_i$ $(i = 0, 1, \ldots, k-1)$ and q_jp_i is sent to neuron $\sigma_{d_{i+1}}$, where $0 \le i \le k-1$, $0 \le j \le k-1$.

At $t = 8$, q_0p_0 arrives at neuron σ_{d_0} through synapse (b_0, d_0).

At $t = 9$, q_1p_0 and q_0p_1 arrive at σ_{d_1}, and this encodes the operation $q_1p_0 + q_0p_1$.

...

At $t = 8+k$, $q_1p_{k-1}, q_2p_{k-2}, \ldots, q_{k-2}p_2$ and $q_{k-1}p_1$ arrive at σ_{d_k}, and encode the operation $q_1p_{k-1} + q_2p_{k-2} + \ldots + q_{k-1}p_1$.

...

At $t = 8 + 2k - 2$, $q_{k-1}p_{k-1}$ arrives at $\sigma_{d_{2k-2}}$.

(4) **Output the result**

The number of spikes contained in neuron σ_{d_i} above is same as the binary bits of $P \times Q$. The spikes in neurons σ_{d_0}, σ_{d_1}, ... , $\sigma_{d_{2k-2}}$ above are sent to the neuron σ_{out_m} in turn. According to the number of spikes in the output neuron σ_{out_m}, we have the following cases.

(a) If the number of spikes is odd, the rule $a^{2j+1}/a^{j+1} \to a$ on synapse (out_m, in_m) is applied. Hence, one spike is sent to environment after consuming $j+1$ spikes, and j spikes remain in σ_{out_m} for the next step.

(b) If the number of spikes is even, the rule $a^{2j}/a^j \to \lambda$ on synapse (out_m, in_m) is applied. Hence, no spikes are sent to environment after consuming j spikes and j spikes remain in σ_{out_m} for the next step.

At time $t = 6$, the spikes from the random number module starts to enter the multiplication module and at $t = 10$ the spikes starts to leave the multiplication module. The coefficient corresponding to 2^0, i.e., p_0q_0 leaves the multiplication module first. The other coefficients corresponding to the power of 2 also leave the multiplication module in sequence.

3.4 Comparison Module

In the large integer factorization system $\prod_{RSA}$, the comparison module only needs to determine whether the product is equal to the given large integer N or not. It can be compared from the lowest bit to the highest bit of two binary numbers. If any difference is found, the two numbers are not equal. The comparison module of Fig. 1 implemented by the WSN P system with rules on synapses is shown in Fig. 5.

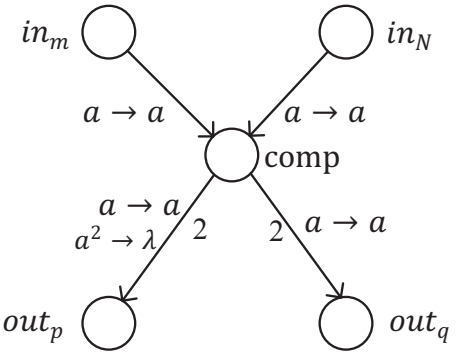


Fig. 5. The comparison module of $\prod_{RSA}$

At time $t = 10$, the input neuron σ_{in_m} of the comparison module receives the product sent by the multiplication module, and the other input neuron σ_{in_N} receives the large integer N sent by the input module (after 7 time steps). If the value of the same bits of two binary numbers is 1, then the rules $a \to a$ on both synapses $(in_m, comp)$ and $(in_N, comp)$ fire. At the next moment, neuron σ_{comp} will get 2 spikes and use the rule $a^2 \to \lambda$ to forget these 2 spikes. When one of these two values is 1 and the other is 0, only one rule $a \to a$ is triggered on synapse $(in_m, comp)$ or $(in_N, comp)$. The neuron σ_{comp} will get 1 spike at the next time, and then the rule $a \to a$ on synapses $(comp, out_p)$ and $(comp, out_q)$ will fire at the same time and send 2 spikes (the weights are 2) to neurons σ_{out_p} and σ_{out_q} of the output module respectively. When the values of the same bits of two binary numbers are all 0, no rules can be applied.

If the value in the same bits of two binary numbers are always same, then the number of spikes received by neurons σ_{out_p} and σ_{out_q} from neuron σ_{comp} is 0, and if there is any difference, the number of spikes received by neurons σ_{out_p} and σ_{out_q} from neuron σ_{comp} is a multiple of two.

3.5 Output Module

As shown in Fig. 6, WSN P system with rules on synapses is used to perform the output module of Fig. 1.

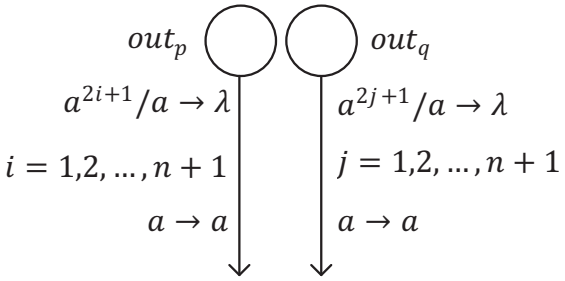


Fig. 6. The output module of $\prod_{RSA}$

After comparing two numbers, at time $t = n + 11$, the neurons σ_{out_p} and σ_{out_q} receive P and Q sent from neurons σ_{ran_p} and σ_{ran_q} of the random number module (the rules on synapses (ran_p, out_p), (ran_q, out_q) fire at $t = 6$ and are delivered after $n+5$ time steps). Next, the number of spikes contained in neurons σ_{out_p} and σ_{out_q} can be divided into four cases. We take σ_{out_p} as an example, and σ_{out_q} can be explained in the same manner as σ_{out_p}.

(1) If the neuron σ_{out_p} has only 1 spike, then this 1 spike is received from random number module. Moreover, this spike is sent to the environment by using the rule $a \to a$. Hence, the product is equal to N, and the binary bit of P is 1.

(2) If the neuron σ_{out_p} contains $(2i+1)$(i=1,2,...,n+1) spikes, then even number of spikes are received from σ_{comp} and 1 spike is received from random number module at the same time. 1 spike is consumed by rule $a^{2i+1}/a \to \lambda$. It means that the product is not equal to N, and the binary bit of P is 1.

(3) If there are $2i$ spikes, where $i = 1, 2, ..., n+1$, no rules can be used. It means that the product is not equal to N, and the binary bit of P is 0.

(4) If there are 0 spikes, no rules can be used. It means that the product is equal to N, and the binary bit of P is 0.

Therefore, the output of each factorization module can be summarized in two cases. If the neurons σ_{out_p} and σ_{out_q} of a factorization module have sent spikes to the environment, it means that the factorization module has found the correct P and Q, and the output values of neurons σ_{out_p} and σ_{out_q} are P and Q. If the neurons σ_{out_p} and σ_{out_q} do not send any spikes to the environment, it means that the two random numbers randomly generated in this factorization module cannot factorize the large integer N.

With the above explanation of different modules the reader can verify the prime factorization of large integers using WSN P systems with rules on synapses. In the next section, we explain the above procedure for $N = 10101_2$.

4 System Analysis

4.1 Example Analysis

To better illustrate the correctness of the system and its specific operation process, we analyze the following example of factorizing integer $N = 10101_2$, which is product of two prime numbers 11_2 and 111_2.

First, we give the integer $N = 10101_2$, and the bit length of it is $n = 5$. Then the binary bit length of P and Q satisfies $1 < k < 5$, and the multiplication module of $k = 4$ is used to compute the product.

Table 2 reports the spikes contained in each neuron of the input module at each step during the computation. The input and output sequences are written in bold.

Table 2. The configurations of the input module at each time step during the computation

t	in	aux_1	aux_2	$in_{ri}(i = 1, 2, ..., s)$	$in_{Nj}(j = 1, 2, ..., s)$
0	2	0	0	0	0
1	**1**+2	0	0	0	0
2	**0**	1	1	0	0
3	**1**	0	0	**6**	O(7)
4	**0**	0	1	0	O(6)
5	**1**	0	0	0	O(5)O(7)
6	0	0	1	0	O(4)O(6)
7	0	0	0	0	O(3)O(5)O(7)
8	0	0	0	0	O(2)O(4)O(6)
9	0	0	0	0	O(1)O(3)O(5)
10	0	0	0	0	**1**
11	0	0	0	0	**0**
12	0	0	0	0	**1**
13	0	0	0	0	**0**
14	0	0	0	0	**1**

The following is a case of the factorization module that finds the correct P and Q. Other factorization modules are not discussed in detail.

Assume that the value of P generated by the random number module in the factorization module $i(1 \leq i \leq s)$ is 11_2, and the value of Q is 0111_2. In this case, the number of spikes contained in each neuron of the random number module at

each step during the process of generating random numbers P and Q is shown in Table 3.

Table 3. The configurations of the random number module at each time step during the process of generating random numbers $P = 11_2$ and $Q = 0111_2$

t	in_r	aux_3	aux_4	ran_p	ran_q	in_p	in_q	out_p	out_q
3	**6**	0	0	0	0	0	0	0	0
4	0	2	4	0	0	0	0	0	0
5	0	1	3	**1**	**1**	0	0	0	0
6	0	0	2	**1**	**1**	**1**+2	**1**	O(10)	O(10)
7	0	0	1	0	**1**	**1**+4	**1**	O(9)O(10)	O(9)O(10)
8	0	0	0	0	**0**	0+6	**1**	O(8)O(9)	O(8)O(9)O(10)
9	0	0	0	0	0	0+8	**0**	O(7)O(8)	O(7)O(8)O(9)
...	...	...	...	...	...	...	...	...	...
15	0	0	0	0	0	0	0	O(1)O(2)	O(1)O(2)O(3)
16	0	0	0	0	0	0	0	**1**	**1**
17	0	0	0	0	0	0	0	**1**	**1**
18	0	0	0	0	0	0	0	**0**	**1**
19	0	0	0	0	0	0	0	**0**	**0**

The generated random numbers $P = 11_2$ and $Q = 0111_2$ are input to the multiplication module of $k = 4$. The multiplication module begins to work at $t = 6$.

Table 4 reports the spikes contained in each neuron of the multiplication module of $k = 4$ at each step during the computation $P \times Q = 0011_2 \times 0111_2$. The larger valid length of binary bit is 3, therefore, the extra neurons of the multiplication module are not listed in Table 4.

From Table 4, it is known that $P \times Q = 010101_2$. While calculating the product, the multiplication module directly inputs this result to the comparison module. Since output of the multiplication module is send to neuron σ_{in_m} at time $t = 10$, the comparison module starts at $t = 10$.

During the process of comparing the value of $P \times Q = 010101_2$ and $N = 10101_2$, the number of spikes contained in each neuron of the comparison module at each step is reported in Table 5.

According to the result in Table 5, at time $t = 16$, the output module receives the P and Q sent from the random number module. The spikes contained in each neuron of the output module, as well as the number of spikes sent to the environment at each step are shown in Table 6.

Table 4. The configurations of the multiplication module at each time step during the computation $P \times Q = 0011_2 \times 0111_2$

t	in_p	in_q	a_0	a_1	a_2	g_2	g_3	b_0	b_1	b_2	d_0	d_1	d_2	d_3	d_4	out_m	in_m
5	0	0	2	0	0	3	3	0	0	0	0	0	0	0	0	0	0
6	**1**+2	**1**	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
7	**1**+4	**1**	0	0	2	0	0	3	0	0	0	0	0	0	0	0	0
8	0+6	**1**	0	0	0	0	0	7	3	0	1	0	0	0	0	0	0
9	0+8	**0**	0	0	0	0	0	11	7	1	0	2	0	0	0	1	0
10	0	0	0	0	0	0	0	2	11	5	0	0	2	0	0	2	**1**
11	0	0	0	0	0	0	0	2	2	9	0	0	0	1	0	3	**0**
12	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	2	**1**
13	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	1	**0**
14	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0	**1**
15	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0	**0**

From the result in Table 6, we know that the two prime numbers that can factorize large integer $N = 10101_2$ are $P = 11_2$ and $Q = 111_2$ respectively. The system ends.

4.2 Complexity Analysis

The complexity of the system in this paper is based on the required time and space to analyze the algorithm. The time complexity is represented by the number of steps taken by the system to obtain the prime factorization and the space complexity is represented by the number of neurons present in the system.

From the above tables we know that if the length of the input N is 5, then the neuron σ_{out_m} in the multiplication module receives input at $t = 10$. If the length of the valid output bits is $n + 1$, then the time at which the P and Q are input to the neurons σ_{out_p} and σ_{out_q} of output module is $t = n + 11$. Hence, the first valid output time of the neurons σ_{out_p} and σ_{out_q} of the output module is $t = n + 12$, and the valid output is $n - 1$ bits at most cases, so the end time of the system is $t = n + 11 + n - 1 = 2n + 10$. On the basis of enough parallel structures, it can be concluded that:

$T = 2n + 10 = O(n)$

The number of neurons required in the rules on synapses of WSN P system is analyzed from the above figures. The number of neurons required for the input module is 3 (the neurons in last row in each graph are the input neurons of next module). The number of neurons required for the random number module is 5 and the number of neurons required for the multiplication module is 2 +

Table 5. The configurations of the comparison module at each time step during the the process of comparing the value of $P \times Q = 010101_2$ and $N = 10101_2$

t	in_m	in_N	$comp$	out_p	out_q
10	**1**	**1**	0	0	0
11	**0**	**0**	2	0	0
12	**1**	**1**	0	0	0
13	**0**	**0**	2	0	0
14	**1**	**1**	0	0	0
15	**0**	0	2	0	0
16	0	0	0	**1**	**1**
17	0	0	0	**1**	**1**
18	0	0	0	0	**1**
19	0	0	0	0	**0**

Table 6. The configurations and outputs of the output module at each time step

t	out_p	out_q	$output_p$	$output_q$
16	**1**	**1**	0	0
17	**1**	**1**	**1**	**1**
18	0	**1**	**1**	**1**
19	0	**0**	**0**	**1**
20	0	0	**0**	**0**

$k + k + (k - 1) + (2k - 1) + 1 = 5k + 1$, where $k = n - 1$. So the number of neurons needed in the multiplication module is $5n - 4$. Furthermore, the number of neurons required for the comparison module is 3 and the number of neurons required for the output module is 2. Therefore, the number of neurons required for a large integer factorization system $\prod_{RSA}$ containing only one factorization module is $S = 3 + 5 + 5n - 4 + 3 + 2 = 5n + 9 = O(n)$. The space complexity of the whole large integer factorization system is also related to the number of parallel factorization modules, i.e., s which is exponential in nature. Hence, to test all possible P and Q, exponential workspace is required. So, the prime factorization of the large integer N can be obtained in linear time with the exponential workspace provided by the rules on synapses of WSN P systems.

4.3 Discussion

In this paper, WSN P systems with rules on synapses are used to implement large integer factorization. In fact, this is the first time to use the P systems to break RSA encryption. Although the system constructed in this paper can break RSA encryption in theory, it still has some limitations. 1. The two natural numbers to compute product in the large integer factorization SN P system $\prod_{RSA}$ are randomly generated by the random number module, which makes the system uncertain; 2. The integers in the RSA algorithm are large numbers, so the system $\prod_{RSA}$ designed in this paper can successfully factorize large integer based on the parallel structures whose number is exponential. The number of the required neurons in the system is also exponential, making the system space too complex.

5 Conclusions and Future Work

In this paper, a WSN P system with rules on synapses is used to break RSA encryption by solving the prime factorization of large integers. Moreover, through the input module, random number module, multiplication module, comparison module and output module the working of the system is analyzed in detail. Since the number of neurons is massive, the probability of successfully factorizing a large integer can approach to 1 by using parallel structures. So, solving the computationally hard problems using SN P systems can be a future direction of research.

Acknowledgments. This work was supported by National Natural Science Foundation of China (61179032, 61672437, 61702428), the Special Scientific Research Fund of Food Public Welfare Profession of China (201513004-3), the Guiding Scientific Research Project of Hubei Provincial Education Department (2017078), the Humanities and Social Sciences Fund Project of Hubei Provincial Education Department (17Y071), Sichuan Science and Technology Program (2018GZ0185, 2018GZ0086) and New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0044).

References

1. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM. 21(2), 120-126 (1978).
2. Guo, S., Jiang, X.: Research and implementation of RSA information security encryption system. Network Security Technology & Application. (1), 35-38 (2018).
3. Siguna, M., Winfried, B.: The security of public key cryptosystems based on integer factorization. Information security and privacy. 9-23 (1998).
4. Shor, P.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM journal on computing. 26(5), 1484-1509 (1997).

5. Wang, Y., Zhang, H., Wu, W.: Quantum algorithms for breaking RSA based on phase estimation and equation solving. Chinese Journal of Computers. 40(12), 2688-2699 (2017).
6. Jaya, I., Hardi, S., Tarigan, J.: Distributed factorization computation on multiple volunteered mobile resource to break RSA key. Journal of Physics Conference Series. 012081 (2017).
7. Zhang, X., Niu, Y., Cui, G.: Breaking the RSA public key cryptosystem using self-assembly of DNA tilings. Systems Engineering and Electronics. 32(5), 1094-1099 (2010).
8. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences. 61(1), 108-143 (2000).
9. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, A., Woods, D.: Complexity: Membrane division, membrane creation. In: Păun et al. (eds.) The Oxford Handbook of Membrane Computing. Oxford University Press. 302-336 (2009).
10. Păun, Gh.: Membrane computing: an introduction. Theoretical Computer Science. 287(1), 73-100 (2002).
11. Alhazov, A., Martin-Vide, C., Pan, L.: Solving a PSPACE-Complete problem by P systems with restricted active membranes. Fundamenta Informaticae. 58(2), 66-77 (2003).
12. Păun, Gh.: P systems with active membranes: attacking NP-Complete problems. Journal of Automata, Languages and Combinatorics. 6(1), 75-90 (2001).
13. Leporati, A., Zandron, C., Ferretti, C., Mauri, G.: Solving mumerical NP-Complete problems with spiking neural P systems. G.k.eleftherakis. 4860, 336-352 (2007).
14. Zandron, C., Ferretti, C., Mauri, G.: Solving NP-Complete problems using P systems with active membranes. Unconventional Models of Computation, Springer, London. 289-301 (2000).
15. Diaz-Pernil, D., Gutierrez-Naranjo, M., Pérez-Jiménez, M.J., et al: A linear-time tissue P system based solution for the 3-coloring problem. Electronic Notes on Theoretical Computer Science. 171, 81-93 (2007).
16. Diaz-Pernil, D., Gutierrez-Naranjo, M., Pérez-Jiménez, M.J., et al: A uniform family of tissue P system with cell division solving 3-COL in a linear time. Theoretical Computer Science. 404, 76-87 (2008).
17. Diaz-Pernil, D., Gutierrez-Naranjo, M., Pérez-Jiménez, M.J., et al: Solving subset sum in linear time by using tissue P system with cell division. Lecture Notes in Computer Science. 4527, 170-179 (2007).
18. Krishna, S.N.: Languages of P systems: computability and complexity. (2001).
19. Guo, P., Xu, W.: A family P system of realizing RSA algorithm. Bio-inspired Computing -Theories and Applications. Springer Singapore. 155-167 (2016).
20. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae. 71, 279-308 (2006).
21. Păun, Gh.: Spiking neural P systems. power and efficiency. Lecture Notes in Computer Science. 4527, 153-169 (2007).
22. Pan, L., Zeng, X., Zhang, X., Jiang, X.: Spiking neural P systems with weighted synapses. Neural Processing Letters. 35(1), 13-27 (2012).
23. Song, T., Pan, L., Păun, Gh.: Spiking neural P systems with rules on synapses. Theoretical Computer Science. 529, 82-95 (2014).
24. Zhang, X., Zeng, X., Pan, L.: Weighted spiking neural P systems with rules on synapses. Fundamenta Informaticae. 134, 201-218 (2014).
25. Song, T., Gong, F., Liu, X., Zhao, Y., Zhang, X.: Spiking neural P systems with white hole neurons. IEEE Transactions on Nanobioscience. 15(7), 666-673 (2016).

26. Wu, T., Zhang, Z., Păun, Gh., Pan, L.: Cell-like spiking neural P systems. Theoretical Computer Science. 623, 180-189 (2016).
27. Pan, L., Wu, T., Su, Y., Vasilakos, A.V.: Cell-like spiking neural P systems with request rules. IEEE Transactions on Nanobioscience. PP(99), 1-1 (2017).
28. Song, T., Rodriguez-Paton, A., Zheng, P., Zeng, X.: Spiking neural P systems with colored spikes. IEEE Transactions on Cognitive & Developmental Systems. PP(99), 1-1 (2017).
29. Wu, T., Păun, A., Zhang, Z., Pan, L.: Spiking neural P systems with polarizations. IEEE Transactions on Neural Networks & Learning Systems. PP(99), 1-12 (2017).
30. Pan, L., Păun, Gh., Zhang, G., Neri, F.: Spiking neural P systems with communication on request. International Journal of Neural Systems. 27(8), 1750042 (2017).
31. Song, T., Pan, L.: Spiking neural P systems with rules on synapses working in maximum spikes consumption strategy. IEEE Transactions on NanoBioscience. 14(1), 38-44 (2015).
32. Zeng, X., Xu, L., Liu, X., Pan, L.: On languages generated by spiking neural P systems with weights. Information Sciences. 278, 423-433 (2014).
33. 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 (2017).
34. Wu, T., Bilbie, F.-D., 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(8), 1850013 (2018).
35. Kong, Y.: Study on the computational power of spiking neural P system based on different biological background. Huazhong University of Science and Technology. (2015).
36. Zhang, G, Pérez-Jiménez, M.J., Gheorghe, M.: Real-life applications with membrane computing. Springer. (2017).
37. 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(5), (2014).
38. Wang, T. Zhang, G., Zhao, J., He, Z., Wang, J., Pérez-Jiménez, M.J.: Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. IEEE Transactions on Power Systems. 30(3), 1182-1194 (2015).
39. 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).
40. Song, T., Zheng, P., Wong, M., Wang, X.: Design of logic gates using spiking neural P systems with homogeneous neurons and astrocytes-like control. Information Sciences. 372, 380-391 (2016).
41. Xue, J., Liu, X.: Solving directed hamilton path problem in parallel by improved SN P system. International Conference on Pervasive Computing and the Networked World. Springer-Verlag. 689-696 (2012).
42. Ceterchi, R., Tomescu, A.: Spiking neural P systems - A natural model for sorting networks. Daniel Diaz-Pernil. 93-106 (2010).
43. Ionescu, M., Sburlan, D.: Some applications of spiking neural P systems. Computing & Informatics. 27(3), 515-528 (2008).
44. Wang, H., Zhou, K., Zhang, G.: Arithmetic operations with spiking neural P systems with rules and weights on synapses. International Journal of Computers, Communications and Controls. 13(4), 574-589 (2018).
45. Zhang, X., Zeng, X., Pan, L., Luo, B.: A spiking neural P system for performing multiplication of two arbitrary natural numbers. Chinese Journal of Computers. 32, 2362-2372 (2009).

Applying learning deterministic stream X-machines from queries to P systems synthesis

Florentin Ipate [1], Marian Gheorghe [2], Gexiang Zhang [3]

[1] Department of Computer Science,
Faculty of Mathematics and Computer Science and ICUB
University of Bucharest,
Str. Academiei nr. 14, 010014, Bucharest, Romania
florentin.ipate@ifsoft.ro
[2] School of Electrical Engineering and Computer Science,
University of Bradford, West Yorkshire, Bradford BD7 1DP, UK
{m.gheorghe}@bradford.ac.uk
[3] School of Electrical Engineering,
Southwest Jiaotong University,
Chengdu, Shichuan, China
{zhgxdylan}@126.com

Abstract. Synthesis of different formalisms is a key element in developing accurate models when some of their components are known or when only some limited information on their behaviour is available. In this case, different learning methods have been developed in order to define models satisfying specific constraints. Amongst the most developed learning methods and techniques are those referring to finite state models and regular languages. Some of these approaches can be applied to other classes of models either be adopting the finite state machine learning methods to those models or by inferring these models from adequate finite state machines defined through learning algorithms.
P systems are computational models inspired by the structure and behaviour of the living cells or other, more complex, similar biological entities, such as tissue, organs, bacterium colonies. These models have been intensively investigated and some learning like methods have been introduced for synthesising such systems. All these approaches had relied on optimisation techniques based on evolutionary computing methods. Here we propose a learning approach based on X-machine learning; an X-machine is a generalisation of finite state machine model. Knowing the association between X-machines and various classes of P systems, one can the translate the X-machine model, obtained via a learning approach, into a P system.
Learning regular languages from queries was introduced by Angluin in a seminal paper that also provides a learning algorithm, called L^*. However, while finite state machines can successfully model the control aspects of a system, their capability of modeling the system data is quite limited. Extended finite state machine formalisms, such as stream X-machines, that combine the control aspects of the system with system data, exist and can be used to alleviate this limitation. We investigates the problem of learning deterministic stream X-machines from queries

and provides solutions to this problem. This involves an adaptation of the L^* algorithm to the more complex case of stream X-machines, in which additional constraints to the underlying automaton appear and, hence, additional, non-trivial, issues need to be solved. Learning X-machines (and consequently P systems) from queries has important practical applications since it provides a way of inferring a *complete* state based (or a P system) model of a system from specification, with very promising applications, ranging from microbiology, neuroscience to software and industrial automation systems.

Acknowledgements

FI and MG are supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-III-P4-ID-PCE-2016-0210, project name MACPS (Modelling and Analysis of Cyber-Physical Systems). The work of GZ is supported by the National Natural Science Foundation of China (61672437, 61702428), the Sichuan Science and Technology Program (2018GZ0185,2018GZ0086), and New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0044).

A Turing Machine Simulation by P Systems without Charges

Alberto Leporati[1], Luca Manzoni[1], Giancarlo Mauri[1], Antonio E. Porreca[1,2], and Claudio Zandron[1]

[1] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336, 20126 Milano, Italy
{leporati,luca.manzoni,mauri,zandron}@disco.unimib.it
[2] Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
antonio.porreca@lis-lab.fr

Abstract. It is well known that the kind of P systems involved in the definition of the P conjecture is able to solve problems in the complexity class P by leveraging the uniformity condition. Here we show that these systems are indeed able to simulate deterministic Turing machines working in polynomial time with a weaker uniformity condition and using only one level of membrane nesting. This allows us to embed this construction into more complex membrane structures, possibly showing that constructions similar to the one performed in [1] for P systems with charges can be carried out also in this case.

1 Introduction

The construction of P systems simulating Turing machines (TM) using as few membranes (or cells) as possible and limiting the depth of the system is one of the "tricks" that allowed the nesting of multiple machines to solve problems in large complexity classes. For example, nesting of non-deterministic machines (where the non-determinism was simulated by membrane division) and a counting mechanism allows to characterize $\mathsf{P}^{\#\mathsf{P}}$, the class of all problems solvable by a deterministic TM with access to a $\#\mathsf{P}$ oracle [1,3]. The same ideas can be applied to tissue P systems [4], where the different communication topology makes even more important to keep TM simulations compact [2].

The P conjecture is a long-standing open problem in membrane computing first presented in 2005 [7, Problem F] that, in its essence, asks what is the power of one charge when compared to two charges. We feel that one important step to determine the computational power of active membrane systems without charges and with membrane dissolution is to see which is the minimal system able to simulate a deterministic polynomial-time TM. Here we show that a shallow system is sufficient to perform such a simulation *without* delegating everything to the machine of the uniformity condition. Hopefully, this construction will allow us to define systems in which different TM can be "embedded" at different levels

in a large membrane structure, thus making possible to mimic the construction performed in [1] for P systems with charges.

This paper is organized as follows: Section 2 will recall some basic notions on P systems. The main construction and result is presented in Section 3, while ideas for further research are presented in Section 4.

2 Basic Notions

For an introduction to membrane computing and the related notions of formal language theory and multiset processing, we refer the reader to *The Oxford Handbook of Membrane Computing* [8]. Here we recall the formal definition of P systems with active membranes using weak non-elementary division rules [6,9].

Definition 1. *A* P system with active membranes with dissolution rules *of initial degree* $d \geq 1$ *is a tuple*

$$\Pi = (\Gamma, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$$

where:

- Γ *is an alphabet, i.e., a finite non-empty set of symbols, usually called* objects*;*
- Λ *is a finite set of labels;*
- 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 *in a one-to-one way;*
- $w_{h_1}, \ldots, w_{h_d}$*, with* $h_1, \ldots, h_d \in \Lambda$*, are multisets (finite sets with multiplicity) of objects in* Γ*, describing the initial contents of each of the* d *regions of* μ*;*
- R *is a finite set of rules.*

The rules in R are of the following types:

(a) *Object evolution rules*, of the form $[a \to w]_h$.
They can be applied inside a membrane labelled by h and containing an occurrence of the object a; the object a is rewritten into the multiset w (i.e., a is removed from the multiset in h and replaced by the objects in w).

(b) *Send-in communication rules*, of the form $a\,[\,]_h \to [b]_h$.
They can be applied to a membrane labelled by h and such that the external region contains an occurrence of the object a; the object a is sent into h becoming b.

(c) *Send-out communication rules*, of the form $[a]_h \to [\,]_h\, b$.
They can be applied to a membrane labelled by h and containing an occurrence of the object a; the object a is sent out from h to the outer region becoming b.

(d) *Dissolution rules*, of the form $[a]_h \to b$.
They can be applied to a non-skin membrane labelled by h and containing an occurrence of the object a; the object a is sent out from h to the outer region becoming b, the membrane h ceases to exist and all the other objects it contains are sent into the outer region.

A computation step changes the current configuration according to the following principles:

- The application of rules is *maximally parallel*: each object appearing on the left-hand side of evolution, communication, or division rules must be subject to exactly one of them. Analogously, each membrane can only be subject to one communication or dissolution rule (types (b)–(d)) per computation step; for this reason, these rules will be called *blocking rules* in the rest of the paper. As a result, the only objects and membranes that do not evolve are those associated with no rule.
- 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 after 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 microsteps whereby each membrane evolves only after its internal configuration (including, recursively, the configurations of the membrane substructures it contains) has been updated.
- The outermost membrane (the root of the membrane structure) cannot be divided, 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 $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ of configurations, 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 $\mathcal{C}_k$.

P systems can be used as language *recognisers* by employing two distinguished objects yes and no: we assume that all computations are halting, and that either one copy of object yes or one of object no is sent out from the outermost membrane, and only in the last computation step, in order to signal acceptance or rejection, respectively. If all computations starting from the same initial configuration are accepting, or all are rejecting, the P system is said to be *confluent*.

In order to solve decision problems (or, equivalently, decide languages), we use *families* of recogniser P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$. Each input x is associated with a P system Π_x deciding the membership of x in a language $L \subseteq \Sigma^\star$ by accepting or rejecting. The mapping $x \mapsto \Pi_x$ must be efficiently computable for inputs of any length, as discussed in detail in [5].

Definition 2. *A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is* (polynomial-time) uniform *if the mapping $x \mapsto \Pi_x$ can be computed by two polynomial-time deterministic Turing machines E and F as follows:*

- *$F(1^n) = \Pi_n$, where n is the length of the input x and Π_n is a common P system for all inputs of length n, with a distinguished input membrane.*
- *$E(x) = w_x$, where w_x is a multiset encoding the specific input x.*
- *Finally, Π_x is simply Π_n with w_x added to a specific membrane, called the* input membrane.

The family $\boldsymbol{\Pi}$ is said to be (polynomial-time) semi-uniform if there exists a single deterministic polynomial-time Turing machine H such that $H(x) = \Pi_x$ for each $x \in \Sigma^\star$.

Any explicit encoding of Π_x is allowed as output of the construction, as long as the number of membranes and objects represented by it does not exceed the length of the whole description, and the rules are listed one by one. This restriction is enforced in order to mimic a (hypothetical) realistic process of construction of the P systems, where membranes and objects are presumably placed in a constant amount during each construction step, and require actual physical space proportional to their number; see also [5] for further details on the encoding of P systems.

3 Simulation of Polynomial-time Turing machines

In this section we provide a simulation of a deterministic TM working in polynomial time by a P system that uses only one level of nesting. Any information exchange between objects can happen only via dissolution. By applying different evolution rules, it is possible for an object to detect whether it is inside or outside an elementary membrane (i.e., to "know" if the elementary membrane where it was has been dissolved). By combining this mechanism with a timer, it is also possible to encode the time when the membrane was dissolved, thus allowing to evolve in different ways according to this additional information.

Let M be a polynomial-time deterministic TM having alphabet Σ, set of states Q, and transition function $\delta : Q \times \Sigma \to Q \times \Sigma \times \{-1, +1\}$. We assume that, for an input of length n machine M halts in time $p(n)$ and, thus, it uses no more than $p(n) + 1$ cells. We are going to define a P system Π that simulates the computation of M in $O(p(n)|\Sigma|)$ steps. That is, the simulation of every step of M will require a number of steps in Π that is proportional to the size of the alphabet of M, thus providing an efficient simulation.

The P system Π has $(p(n) + 1)^2 + p(n)^2 + p(n) + 1$ labels, one for the skin membrane and two for each pair of time and position in the TM tape:

$$\begin{aligned}\Lambda =& \{0\} \cup \{(i, j) \mid i, j \in \{0, \ldots, p(n)\}\} \\ & \cup \{(i, j)' \mid i \in \{0, \ldots, p(n)\},\ j \in \{0, \ldots, p(n) - 1\}\} \quad .\end{aligned}$$

Since we assume that no kind of membrane division is present, in the following we can identify membranes with labels, since each label is used by exactly one membrane. The semantics of the labels is that a membrane with label (i, j) will represent the i-th cell of the TM tape at time j. The additional membrane $(i, j)'$ is used in performing the transition between time steps j and $j + 1$, which also explains why the label is not present for time $p(n)$.

The set of objects of the simulating P system will be:

$$\begin{aligned}\Gamma =&\{a_{i,j,k} \mid i,j \in \{0,\ldots,p(n)\},\ 0 \le k < m+5, a \in \Sigma\}\\ &\cup \{q_{i,j,k} \mid i,j \in \{0,\ldots,p(n)\},\ 0 \le k \le m+5, q \in Q\}\\ &\cup \{q_{i,j,k,a} \mid i,j \in \{0,\ldots,p(n)\},\ 0 \le k \le m+5,\ q \in Q,\ a \in \Sigma\}\\ &\cup \{a_i \mid a \in \Sigma, i \in \{0,\ldots,p(n)\}\} \cup \{q^I\}\end{aligned}$$

where $m = |\Sigma|$ and q^I is the initial state of the TM. The first three sets of the union represent, respectively, the symbols on the tape, the states of the TM, and the states of the TM together with the symbol currently present under the tape head. The last two sets are only used to encode the initial configuration of the TM. The value of k ranges from 0 to $m+5$ because each step of the TM will be simulated in $m+5$ time steps.

Let $a_1, a_2, \ldots, a_{p(n)}$ be the initial contents of the TM tape. It is encoded in the initial configuration of Π as the objects $a_{1,1}, a_{2,2}, \ldots, a_{p(n),p(n)}$ inside the skin membrane. As an example, if the initial content of the tape is $abba$, then it will be encoded by the multiset $a_1 b_2 b_3 a_4$. The initial state q^I is encoded by the object q^I. The following rules send the objects representing the TM tape inside the corresponding membranes: the object a_i is sent into the membrane $(i,0)$ and is rewritten as $a_{i,0,0}$. At the same time the object q^I is rewritten as $q^I_{0,0,0}$:

$$\begin{aligned}&a_i\,[\,]_{(i,0)} \to [a_{i,0,0}]_{(i,0)} && \text{for } a \in \Sigma\\ &[q^I \to q^I_{0,0,0}]_0\end{aligned}$$

These rules will not be further applied during the simulation. After this first "bookkeeping" step the actual simulation of one TM step can start; see Fig. 1 for an example.

Let φ be a bijection from Σ to $\{1,\ldots,m\}$ providing a total ordering of the TM alphabet. The main idea is to have each object representing the symbol a written on position i at time j on the TM tape dissolving the membrane (i,j) when its subscript is $i,j,\varphi(a)$. This means that any other object present in the same membrane (in our case, the object representing the current state of the TM) can infer the symbol under the tape head and act accordingly. The evolution of the objects representing the tape content for the first $m+1$ time steps of each TM step simulation is described by the following rules:

$$\begin{aligned}&[a_{i,j,k} \to a_{i,j,k+1}]_{(i,j)} && \text{for } 0 \le k < \varphi(a) \text{ and } a \in \Sigma\\ &[a_{i,j,k}]_{(i,j)} \to a_{i,j,k+1} && \text{for } k = \varphi(a) \text{ and } a \in \Sigma\\ &[a_{i,j,k} \to a_{i,j,k+1}]_0 && \text{for } \varphi(a) < k \le m \text{ and } a \in \Sigma\end{aligned}$$

Notice how the objects simply "count" in the subscript, except that when $k = \varphi(a)$ the membrane in which they are contained is dissolved.

At the same time the object representing the TM state enters the membrane (i,j), representing that the tape head at time j is in position i and starts to count. When membrane (i,j) is dissolved it is possible to infer the object that dissolved

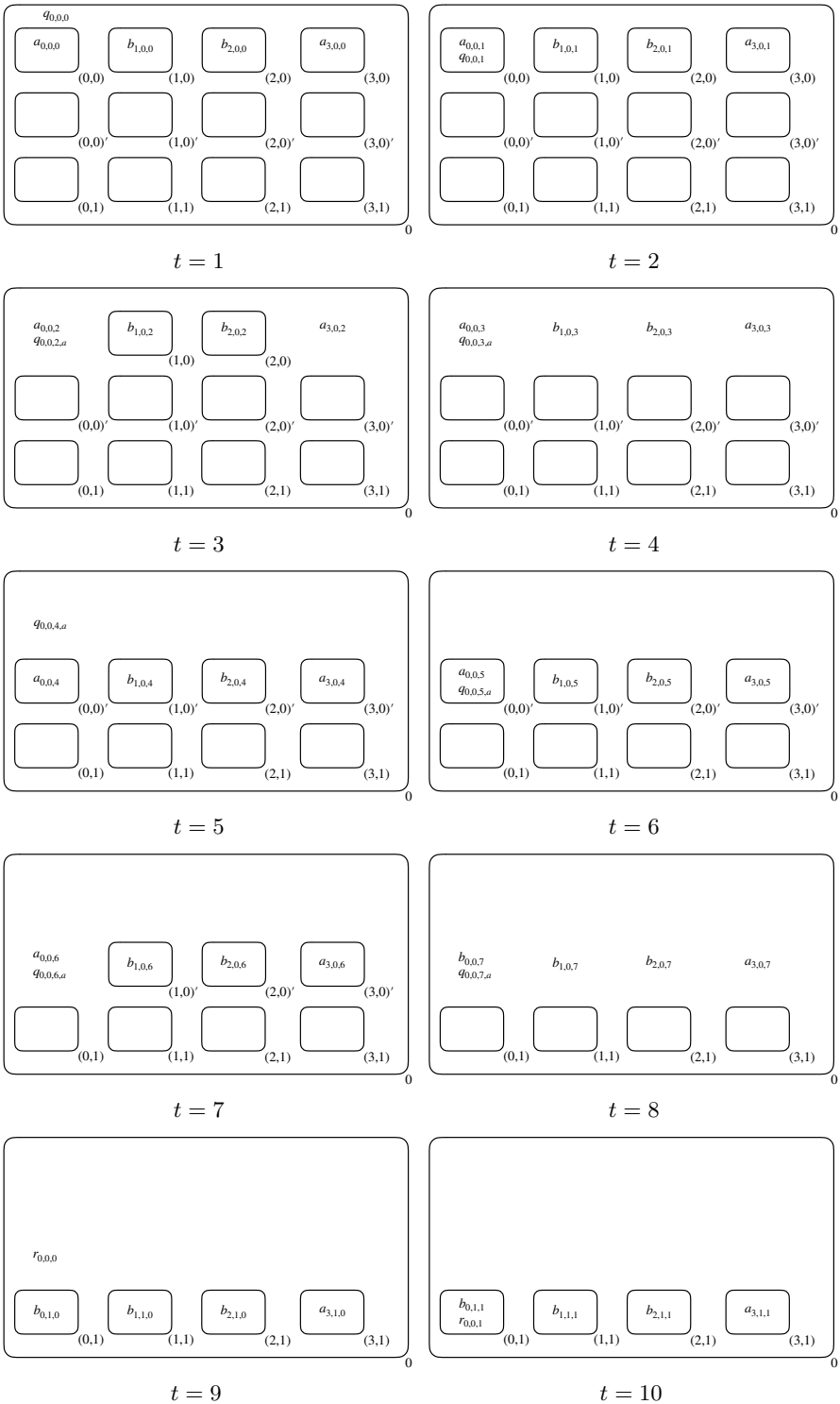


Fig. 1. The simulation of one computation step of the TM M by means of a P system Π. The alphabet Σ is $\{a, b\}$ and the tape contains four cells.

it, and thus the symbol on the tape under the tape head, which is represented by $\varphi^{-1}(a)$ (which is well defined since φ is a bijection between Σ and $\{1,\ldots,m\}$. The corresponding rules are:

$$
\begin{array}{ll}
q_{i,j,0}\,[\,]_{(i,j)} \to [q_{i,j,1}]_{(i,j)} & \text{for } q \in Q \\
[q_{i,j,k} \to q_{i,j,k+1}]_{(i,j)} & \text{for } 1 \le k \le m \text{ and } q \in Q \\
[q_{i,j,k} \to q_{i,j,k+1,\varphi^{-1}(k)}]_0 & \text{for } 1 \le k \le m, \text{ and } q \in Q \\
[q_{i,j,k,a} \to q_{i,j,k+1,a}]_0 & \text{for } 1 \le k \le m,\ a \in \Sigma, \text{ and } q \in Q
\end{array}
$$

At time step $m+1$ in the simulation of the current TM step, all membranes with label (i,j) (for all i and with j the current TM step being simulated) have been dissolved. Now the object representing the TM state continues to wait in the skin membrane while *all* the objects representing the TM tape are sent in into the corresponding membranes $(i,j)'$. These membranes will be employed to delete the current content of the cell under the TM head and to replace it with the new symbol. The rules applied at time step $m+1$ are the following ones:

$$
\begin{array}{ll}
a_{i,j,m+1}\,[\,]_{(i,j)'} \to [a_{i,j,m+2}]_{(i,j)'} & \text{for } a \in \Sigma \\
[q_{i,j,m+1,a} \to q_{i,j,m+2,a}]_0 & \text{for } q \in Q \text{ and for } a \in \Sigma
\end{array}
$$

Ones all the objects of the form $a_{i,j,k}$ have entered the membranes $(i,j)'$, they wait for the object representing the TM state to enter:

$$
\begin{array}{ll}
[a_{i,j,m+2} \to a_{i,j,m+3}]_{(i,j)'} & \text{for } a \in \Sigma \\
q_{i,j,m+2,a}\,[\,]_{(i,j)'} \to [q_{i,j,m+3,a}]_{(i,j)'} & \text{for } q \in Q \text{ and } a \in \Sigma
\end{array}
$$

At time step $m+3$ the membrane containing the object representing the TM state is dissolved. In all other membranes the objects representing the TM tape wait for one more step:

$$
\begin{array}{ll}
[a_{i,j,m+3} \to a_{i,j,m+4}]_{(i,j)'} & \text{for } a \in \Sigma \\
[q_{i,j,m+3,a}]_{(i,j)'} \to q_{i,j,m+4,a} & \text{for } q \in Q \text{ and } a \in \Sigma
\end{array}
$$

One of the focal point of this simulation algorithm happens at time step $m+4$ (always relative to the start of the simulation of the current TM step). Here, all the objects representing the tape content dissolve the membrane $(i,j)'$ in which they are in. The *only* object not performing this step is the one that was sent into the skin membrane by the dissolution triggered by the object representing the TM state. That object is deleted (by being rewritten into the empty multiset ϵ) and the state object produces its replacement according to the transition function δ of the TM:

$$
\begin{array}{ll}
[a_{i,j,m+4}]_{(i,j)'} \to a_{i,j,m+5} & \text{for } a \in \Sigma \\
[a_{i,j,m+4} \to \epsilon]_0 & \text{for } a \in \Sigma \\
[q_{i,j,m+4,a} \to q_{i,j,m+5,a} b_{i+d,j,m+5}]_0 & \text{for } q \in Q,\ a \in \Sigma, \\
 & \text{and } \delta(q,a) = (r,b,i+d)
\end{array}
$$

Notice that the state object will be actually rewritten from q to r during the next time step. Finally, the simulation of the next TM step can start by sending in all the objects representing the TM tape to the membranes $(i, j+1)$ and resetting the last component of their subscript. At the same time the object representing the TM state actually applies the transition function δ and rewrites itself:

$$a_{i,j,m+5}\,[\,]_{(i,j+1)} \to [a_{i,j+1,0}]_{(i,j+1)} \qquad \text{for } a \in \Sigma$$
$$[q_{i,j,m+5,a} \to r_{i+d,j+1,0}]_0 \qquad \text{for } q \in Q,\ a \in \Sigma, \text{ and } \delta(q,a) = (r,b,i+d)$$

Notice that all rules, labels, and objects can be constructed by a logarithmic space TM. In fact, most of them are constructed by iterating either a constant or a polynomial number of times to produce the necessary subscripts. Since the counters are all at most polynomial in the number that they contain, they can be encoded in a logarithmic number of bits.

We can thus state the main result:

Theorem 1. (L,L)-*uniform families of confluent* shallow *P systems with active membranes with dissolution and without division can solve all problems in* P.

The result was already known for non-shallow system [5] but here there are two main innovations: the systems here are *shallow*, i.e., of depth 1, and the construction is via a direct simulation of a Turing machine, which allows one to embed this construction into more complex membrane structures.

Notice that the construction presented here can be modified to simulate a non-deterministic TM by replacing the only two types of rules involving the transition function of the TM in a way to allow for a non-deterministic choice (due to having multiple rules in conflict):

$$[q_{i,j,m+4,a} \to q_{i,j,m+5,(r,b,i+d)} b_{i+d,j,m+5}]_0 \qquad \text{for } q \in Q,\ a \in \Sigma, \text{ and } (r,b,i+d) \in \delta(q,a)$$
$$[q_{i,j,m+5,(r,b,i+d)} \to r_{i+d,j+1,0}]_0 \qquad \text{for } q \in Q \text{ and } a \in \Sigma$$

In the first rule the non-deterministic choice is remembered by writing it in the subscript. In this way, the only rule of the second kind that can fire is the one corresponding to the non-deterministic choice performed. We can then state the following theorem showing that a weaker uniformity condition is still sufficient to solve all NP problems with non-confluent systems:

Theorem 2. (L,L)-*uniform families of* non-confluent shallow *P systems with active membranes with dissolution and without division can solve all problems in* NP.

4 Conclusions

In this paper we showed that P systems without charges can still solve any decision problem in the complexity class P even when the power of the Turing

machines involved in the uniformity conditions is reduced. The TM simulation presented here is quite modular and can be embedded in more complex membrane structures. The resulting simulation is also efficient, requiring a slowdown of only a constant multiplicative factor.

However, some problems remain open, and the most prominent one is to study if the construction presented in [1] can be replicated for systems with charges, possibly adding an additional nesting level to accommodate for the different TM simulation technique. Such a result would show that even without charges the entire counting hierarchy can be computed in constant depth. This is another step in trying to understand what are the features that actually grant P systems the power to go beyond the complexity class P and, in some cases, beyond the entire polynomial hierarchy.

References

1. 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
2. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: Characterising the complexity of tissue P systems with fission rules. Journal of Computer and System Sciences **90**, 115–128 (2017), https://doi.org/10.1016/j.jcss.2017.06.008
3. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: The counting power of P systems with antimatter. Theoretical Computer Science **701**, 161–173 (2017), https://doi.org/10.1016/j.tcs.2017.03.045
4. Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science **296**(2), 295–326 (2003), https://doi.org/10.1016/S0304-3975(02)00659-X
5. 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
6. Păun, G.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics **6**(1), 75–90 (2001)
7. Păun, G.: Further twenty six open problems in membrane computing. In: Gutíerrez-Naranjo, M.A., Riscos-Nuñez, A., Romero-Campero, F.J., Sburlan, D. (eds.) Proceedings of the Third Brainstorming Week on Membrane Computing. pp. 249–262. Fénix Editora (2005), http://www.gcn.us.es/3BWMC/Volumen.htm
8. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
9. Zandron, C., Leporati, A., Ferretti, C., Mauri, G., Pérez-Jiménez, M.J.: On the computational efficiency of polarizationless recognizer P systems with strong division and dissolution. Fundamenta Informaticae **87**, 79–91 (2008), http://content.iospress.com/articles/fundamenta-informaticae/fi87-1-06

An adaptive optimization spiking neural P system to improve exploration and exploitation abilities for solving optimization problems

Ming Zhu[1], Jianping Dong[2], Qiang Yang[1], Haina Rong[2], Prithwineel Paul[2], and Gexiang Zhang[2,1] ⋆

[1] School of Control Engineering, Chengdu University of Information Technology, Chengdu 610225, China

[2] School of Electrical Engeneering, Southwest Jiaotong University, Chengdu, 611756, China

zhgxdylan@126.com

Abstract. Spiking neural P systems (SNPS) are distributed and parallel computing models that incorporate the idea of biological spiking neurons into membrane systems. Inspired by the adaptive learning ability of intelligent organisms and mutations of gene in nature, in this paper we propose an adaptive optimization spiking neural P system (AOSNPS) through introducing an adaptive learning rate and an adaptive mutation into the optimization spiking neural P system (OSNPS). With the adaptive learning rate, the efficiency of probability adjustment of the guider is greatly improved and the probability of the guider never goes beyond the upper or lower bound. With the adaptive mutation, the exploration and exploitation abilities for solving optimization problems are significantly improved and the much better balance between convergence and diversity is captured. Finally, extensive experiments on knapsack problems and an application to power system fault diagnosis have been reported to experimentally prove the viability, adaptability and effectiveness of the proposed adaptive neural system.

Keywords: Spiking neural system; adaptive optimization spiking neural P system; adaptive learning rate; adaptive mutation; power system fault diagnosis; combinatorial optimization; membrane computing.

⋆ Corresponding author.

1 Introduction

Biological neural network [1–3] generally refers to a network of biological brain neurons, cells, contacts, etc., used to generate biological awareness, and to help creatures think and act. Artificial neural networks (ANNs), also referred to as neural networks (NNs) or connection models, are mathematical computational models consisting of interconnected neurons [4, 5] that mimic the behavioral characteristics of biological neural networks and perform distributed parallel information processing. ANNs relies on the complexity of the system to adjust the relationship between a large number of internal nodes to achieve the purpose of self-adaptability, self-organization, machine learning, and information processing. ANNs have been extensively investigated and widely used in various fields, such as signal and image processing [6, 7], classification [8], pattern recognition [9, 10], earthquake prediction [11–13], epilepsy and seizure detection [14, 15], and optimization [16–20].

In the last decades, ANNs has passed three developmental generations with notable characteristics. The fundamental feature of the first generation is McCullochitts neurons [21] (perceptrons or threshold gates). Only Boolean functions [22] can be processed and output digital results. The salient feature of the second generation is activation function with weighted learning ability and processing of analog input and output. The introduction of time [23] (single action spiking of the neuron) concept when neurons encode information is a prominent feature of the third generation [22] of ANNs. As both computationally powerful and biologically more plausible models of neuronal processing, spiking neural networks (SNNs) are increasingly receiving attentions [24–27].

Membrane computing [28] (initiated by Păun) is a branch of natural computing which deals with abstract computing models (called P systems) inspired from the structure and the functioning of the biological cells, organs and colonies (of bacteria). The obtained computing models are distributed computing devices, working in parallel, processing multisets in the compartments defined by membranes. As a novel emerging research front (listed by Thompson Institute for Scientific Information in 2003), membrane computing models have been intensively investigated and used in various applications like in the areas of biology and biomedicine [29, 30], computer graphics [31], cryptography [32], robot control [33, 34], distributed evolutionary computing [32, 35, 36], power system fault diagnosis [37, 38] and other real-life complex problems [31, 32, 39, 40].

Membrane computing models can be divided into three categories depending on the membrane structure, i.e., cell-like, tissue-like and neural like. Where, spiking neural

P systems include many different types like Cell-like spiking neural P systems[41, 42], Spiking neural P systems with communication on request(SNQPS)[43, 44] and optimization spiking neural P systems(OSNPS)[45]. Unlike cell-like and tissue-like P systems [31, 32, 39], spiking neural P systems (SNPS) [32, 46] deal with a unique object. The spike is an electrical impulses of identical shape voltage, sent from a neuron to another one along the synapses. Among the various investigations on membrane computing, SNPS is one of the most promising and important research directions [32, 40, 47–50]. An SNPS consists of a set of neurons placed in the nodes of a graph and sending signals (spikes) along synapses (edges of the graph), under the control of firing and forgetting rules. One neuron is designated as the output neuron of the system and the spikes can exit into the environment, thus producing a spike train. With the spike train, SNPS can generate various semantics and solve an optimization problem. The first study of solving combinatorial optimization problem based on optimization spiking neural P system (OSNPS) was reported in [45]. Membrane-inspired evolutionary algorithms (MIEAs) [35, 40, 51–54] combine P system framework with meta-heuristic algorithms (evolutionary operators are integrated within P systems) to solve optimization problems, but OSNPS tackle combinatorial optimization problems only with SNPS.

OSNPS is the first optimization algorithm of SNPS in the sense of the pure membrane computing semantic framework. In [45], based on SNPS framework, an extended SNPS(ESNPS) was obtained by introducing two additional neurons (continuous supply of spikes to other neurons step by step), the probabilistic selection of evolution rules and the output (spike train) collection from multiple neurons, and a family of ESNPS, called optimization spiking neural P system(OSNPS), were further designed through introducing a guider to adjust probabilities (learning from through continuous learning generation by generation). The experimental result of OSNPS appears promising and competitive when compared with six other optimization algorithms by solving the same Knapsack problem. As indicated in [45], OSNPS starts a new research approach for solving optimization problems and other problems, such as the learning rate is constant, the probability of the guider may go beyond the upper or lower bound, and easy to be trapped in local optimum etc.

Charles Robert Darwin described in *the Origin of the Species*: It is not the strongest of the species that survive, but the one most responsive to change. The natural selection mechanism gives humans a rich source of ideas [55] to solve the challenges and problems we faced and the adaptability of nature is applied to various research fields by humans, such as adaptive resonance theory, adaptive control theory and adaptive opti-

mization technique etc. After a new algorithm emerged, we often introduce the adaptability to the algorithm to improve the ability and robustness of the algorithm to solve the practical problems, such as adaptive genetic algorithm [56, 57], adaptive particle swarm optimization [58, 59], adaptive ant colony algorithm [60, 61], adaptive neural network [62, 63] and adaptive spiking neural network [22, 64, 65]. and these adaptive optimization techniques have made tremendous contributions and value when they are applied to human activities. Therefore, based on the SNPS working framework, designing an adaptive OSNPS is interesting, meaningful and promising.

This paper extends the work in [45] and proposes an adaptive optimization spiking neural P system (AOSNPS) to solve combinatorial optimization problems. More specifically, inspired by the adaptive learning ability of intelligent organisms and mutations of gene in nature, an adaptive learning rate and adaptive mutation were introduced to OSNPS. The learning rate is not a constant, but a dynamical variable that can adapt suitably for each neuron at each generation. With the adaptive learning rate, the efficiency of probability adjustment of the guider is significantly improved and the probability of the guider never goes beyond the upper or lower bound. In addition, two dynamically changing indicator parameters are used to evaluate the evolutionary convergence of global optimal solutions and diversity of probability matrices of the algorithm running process. A triggering rule of mutation and a probability-based mutation mode constitute the adaptive mutation. With the adaptive mutation, the exploration and exploitation abilities of solving optimization problems are significantly improved and the much better balance between convergence and diversity is captured. Finally, experimental results and analysis illustrate the effectiveness of AOSNPS through analyzing the dynamic behavior of the process of solving the Knapsack problem with 1000 items, and comparing the final optimal solution and the total execution generation of solving the Knapsack problem with 5000, 6000, 7000,8000, 9000, and 10000 items using the same termination condition,respectively.

The remainder of this study is organized as follows: Section 2 briefly introduces OSNPS. Section 3 presents the proposed AOSNPS consisting adaptive learning rate, adaptive mutation and the novel guider. Experimental results and analysis are described in Sec. 4. Conclusion and future work are given in Sec.5 and Sec.6 respectively.

2 Preliminaries

In this section, we briefly review the spiking neural P system and the optimization spiking neural P system.

2.1 Spiking Neural P System

A SNPS of degree $m \geq 1$ is a tuple $\Pi = (O, \sigma_1, \cdots, \sigma_m, syn, i_o)$, where:

(1) $O = \{a\}$ is the singleton alphabet (a is called spike);

(2) $\sigma_1, \cdots, \sigma_m$ are neurons, identified by pairs

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m \tag{1}$$

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) $E/a^c \rightarrow a; d$ where E is a regular expression over O, and $c \geq 1, d \geq 0$;

(ii) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, we have $a^s \notin L(E)$;

(3) $syn \subseteq \{1, \ldots m\} \times \{1, \ldots m\}$ with $(i, i) \notin syn$ for $i \in \{1, \ldots m\}$ (synapses between neurons);

(4) $\sigma_{io} (i \in \{1, \ldots m\})$ is the output neuron.

The rules of type (i) are firing or spiking rules and are used in the following manner: if neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \rightarrow a; d$ can be applied. c spikes will be consumed and only $k - c$ spikes will remain in the neuron σ_i. After that the neuron is fired, sending a spike out along all outgoing synapses after d time units (in synchronous mode). If $d = 0$, then the spike is emitted immediately. If $d = 1$, then the spike will be emitted in the next step, etc. If the rule is used at step t and $d \geq 1$, then at steps $t, t+1, ..., t+d-1$ the neuron is closed, i.e., it cannot receive new spikes from the other neurons (if a neuron has a synapse to a closed neuron and tries to send a spike along it, the particular spike is lost). In the step $t + d$, the neuron becomes open again, and it can receive spikes in step $t + d + 1$ from the other neurons. In addition, if a rule $E/a^c \rightarrow a; d$ has $E = a^c$, it can be simplified as $a^c \rightarrow a; d$. If a rule $E/a^c \rightarrow a; d$ has $d = 0$, it can be written as $E/a^c \rightarrow a$.

The rules of type (ii) are forgetting rules and they are applied as follows: if neuron σ_i contains exactly s spikes, the rule $a^s \rightarrow \lambda$ in R_i can be applied, then s spikes are consumed by neuron σ_i.

A configuration of Π at any instant t is a tuple $(n_1, d_1), ..., (n_m, d_m)$, where n_i describes the number of spikes present in the neuron σ_i at the instant t and d_i represents the number of steps to count down until it becomes open. The initial configuration of Π is $(n_1, 0), ..., (n_m, 0)$, that is, all neurons are open initially. Using the rules of the

system in the way described above, a configuration C_* can be reached from another configuration C, such a step is called a transition step.

A computation of Π is a (finite or infinite) sequence of configurations such that:

* The first term of the sequence is the initial configuration of the system and each of the remaining configurations are obtained from the previous one by applying rules of the system in a maximally parallel manner with the restrictions previously mentioned;
* If the sequence is finite (called halting computation) then the last term of the sequence is a halting configuration, where no rule can be applied further.

With any computation we can associate a spike train. If the output neuron spikes, then we have 1 and otherwise we have 0. Hence, the spike train can be represented by the sequence of ones and zeros.

2.2 Optimization Spiking Neural P System

OSNPS was first proposed in [45]. The significant difference between OSNPS and many other membrane-inspired evolutionary algorithms (MIEAs) is that the OSNPS use the probabilistic selection of evolution rules to guide the algorithm to approximately solve the optimization problems but the MIEAs use evolutionary operators of heuristic approaches.

An extended spiking neural P system (ESNPS) is an important basic component of OSNPS, where two additional different neurons σ_{m+1} and σ_{m+2} are added in spiking neural P system. An ESNPS of degree $m \geq 1$ is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_{m+1}, \sigma_{m+2}, syn, I_0), \tag{2}$$

where:

(1) $O = \{a\}$ is the singleton alphabet (a is called *spike*);
(2) $\sigma_1, \ldots, \sigma_m$ are neurons of the form $\sigma_i = (1, R_i, P_i), 1 \leq i \leq m$.
   (a) Every neuron σ_i, has only 1 initial spike.
   (b) $R_i = \{r_i^1, r_i^2\}$ is a set of rules of firing or forgetting spike, where $r_i^1 = \{a \to a\}$ is firing rule and $r_i^2 = \{a \to \lambda\}$ 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, $\sigma_{m+1} = \sigma_{m+2} = (1, \{a \to a\})$, work as a step by step supplier of spikes to neurons $\sigma_1, \sigma_2, \ldots, \sigma_m$.

(4) $syn = \{(i, j) | (1 \leq i \leq m + 1 \wedge j = m + 2) \vee (i = m + 2 \wedge j = m + 1)\}$.

(5) $I_0 = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is a finite set of *output* neurons, and the output is a spike train formed by concatenating the outputs of $\sigma_1, \sigma_2, \ldots, \sigma_m$.

The structure of an ESNPS [45] is shown in Fig. 1.

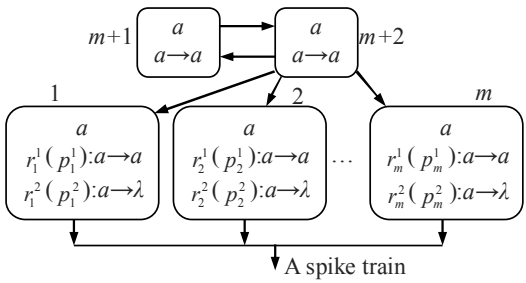


Fig. 1. An example of ESNPS structure

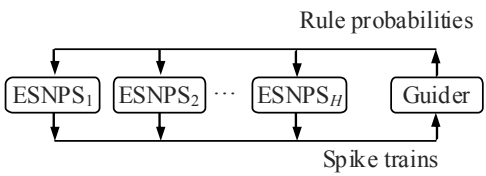


Fig. 2. An example of OSNPS structure

In the ESNPS, at each time unit, each neuron σ_1, σ_2, …, σ_m receives a spike from the cooperating neurons σ_{m+1} and σ_{m+2}. Every neuron $\sigma_1, \ldots, \sigma_m$ performs the firing rule r_i^1 with probability p_i^1 and the forgetting rule r_i^2 with probability p_i^2, $i = 1, 2, \ldots, m$. If the ith neuron spikes, we obtain the output 1, i.e., we obtain 1 with probability p_i^1. Otherwise, we obtain the output 0, i.e., we obtain 0 with probability p_i^2, $i = 1, 2, \ldots, m$. Thus, this system outputs a spike train consisting of 0 and 1 at each moment of time. If we can adjust the probabilities either $p_1^1, \ldots, p_m^1$ or $p_1^2, \ldots, p_m^2$, we

can control the output of the spike train. In both OSNPS and AOSNPS, we choose to adjust the probabilities $p_1^1, \ldots, p_m^1$.

Input: Spike train T_s, p_j^a, Δ, H and m
Rearrange T_s as matrix P_R
$i = 1$
while $(i \leq H)$ **do**
  j=1
  while $(j \leq m)$ **do**
    if $(rand < p_j^a)$ **then**
      $k_1, k_2 = ceil(rand * H), k_1 \neq k_2 \neq i$
      if $(f(C_{k_1}) > f(C_{k_2}))$ **then**
        $b_j = b_{k_1}$
      else
        $b_j = b_{k_2}$
      end if
      if $(b_j > 0.5)$ **then**
        $p_{ij}^1 = p_{ij}^1 + \Delta$
      else
        $p_{ij}^1 = p_{ij}^1 - \Delta$
      end if
    else
      if $(b_j^{max} > 0.5)$ **then**
        $p_{ij}^1 = p_{ij}^1 + \Delta$
      else
        $p_{ij}^1 = p_{ij}^1 - \Delta$
      end if
    end if
    if $(p_{ij}^1 > 1)$ **then**
      $p_{ij}^1 = p_{ij}^1 - \Delta$
    else
      if $(p_{ij}^1 < 0)$ **then**
        $p_{ij}^1 = p_{ij}^1 + \Delta$
      end if
    end if
    $j = j + 1$
  end while
  $i = i + 1$
end while
Output: Rule probability matrix P_R

Fig. 3. The Guider Algorithm of OSNPS

The optimization spiking neural P system is composed of a family of ESNPS. A guider was used to adjust the selection probabilities of rules inside each neuron of each ESNPS. The structure of OSNPS [45] is shown in Fig. 2.

The guider algorithm [45] in Fig. 3., was designed for solving a (specific) single objective and unconstrained combinatorial optimization problem. The detailed execution steps of the guider algorithm have been described in [45]. At the same time, the authors pointed out that the guider can also be modified in order to be suitable for different types of optimization problems.

All of ESNPS$_i$, $1 \leq i \leq H$, are the same as in Fig. 1. The guider is used to adjust the probabilities. The input of the guider is a spike train T_s (the sequence of 1 and 0) with $H \times m$ bits. The output of the guider is the rule probability matrix $P_R = [p_{ij}^1]_{H \times m}$, which is composed of the firing rule probabilities of H ESNPS [45], i.e.,

$$P_R = \begin{pmatrix} p_{11}^1 & p_{12}^1 & \cdots & p_{1m}^1 \\ p_{21}^1 & p_{22}^1 & \cdots & p_{2m}^1 \\ \vdots & \vdots & \ddots & \vdots \\ p_{H1}^1 & p_{H2}^1 & \cdots & p_{Hm}^1 \end{pmatrix} \tag{3}$$

The guider algorithm in [45] was designed for solving a (specific) single objective and unconstrained combinatorial optimization problems. At the same time, the authors pointed out that the guider can also be modified in order to be suitable for different types of optimization problems.

3 Adaptive Optimization Spiking Neural P System

Based on the work in [45] and inspired by the adaptive learning ability of intelligent organisms and mutations of gene, an adaptive optimization spiking neural P system was proposed in this paper. At first, an adaptive learning rate and an adaptive mutation rule are designed to organize a novel adaptive guider.

3.1 Adaptive Learning Rate

Learning rate defined as Δ in [45] is the step size of probability adjustment for p_{ij}^1, $1 \leq i \leq H$, $1 \leq j \leq m$ at each time unit in OSNPS. The probability adjustment method is $p_{ij}^1 = p_{ij}^1 + \Delta$ or $p_{ij}^1 = p_{ij}^1 - \Delta$. Moreover, Δ, a random number between 0.005 and 0.02, is one of the initialization parameters as shown in Fig. 3 and its value will not change during the execution of the algorithm. The efficiency of adjusting the probability with this learning rate is very low because of the fixed step size. A novel adaptive learning rate is proposed here and the probability adjustment efficiency also has significantly increased. Before demonstrating the adaptive learning rate, let us revisit the role of p_i^1 in ESNPS as defined in Eqs.2.

p_i^1 is the selection probability of rule r_i^1 and r_i^1 is the firing rule in each neuron. Then, if the value of p_i^1 is larger, the rule r_i^1 has a higher probability of execution and the rule r_i^2 has a lower probability of execution because $p_i^1 + p_i^2 = 1$. In the coding method: if the output neuron spikes, the output is 1, otherwise the output is 0. So, if we want to get 1, p_i^1 must be large (the ideal situation is $p_i^1 = 1$) and if we want to get 0, p_i^1 must be small (the ideal situation is $p_i^1 = 0$).

In this paper we ingeniously designed an adaptive probability adjustment step size for every neuron. At each time unit, the adaptive updating rule of probability is

$$p_{ij}^1 = p_{ij}^1 + \Delta_{ij}^{adaptive} \tag{4}$$

where $\Delta_{ij}^{adaptive}$ is the adjustment step size and it is defined as

$$\Delta_{ij}^{adaptive} = \frac{P_{\max or \min} - p_{ij}^1}{2} \tag{5}$$

$\Delta_{ij}^{adaptive}$ is designed to take the half of the distance between the current probability p_{ij}^1 and the ideal probability. $P_{\max or \min}$ is the upper or lower bound of the probability of p_{ij}^1. Without loss of generality, we take 1 as the upper bound and 0 as the lower bound. Now, whether to take the upper or lower bound value of $P_{\max or \min}$ is determined depending on the binary value (either 1 or 0) of the bit from we learn from. To be specific:

If we learn from 1 (the binary value of the learned bit), $P_{\max or \min}$ takes the upper bound value 1. The updating rule of probability is

$$p_{ij}^1 = p_{ij}^1 + \frac{1 - p_{ij}^1}{2} = 0.5 + 0.5p_{ij}^1 \tag{6}$$

If we learn from 0 (the binary value of the learned bit), $P_{\max or \min}$ takes the lower bound value 0. The updating rule of probability is

$$p_{ij}^1 = p_{ij}^1 + \frac{0 - p_{ij}^1}{2} = 0.5p_{ij}^1 \tag{7}$$

Comparing with the learning rate Δ defined in OSNPS, the adaptive learning rate $\Delta_{ij}^{adaptive}$ proposed in this paper have the following advantages:

* The novel learning rate $\Delta_{ij}^{adaptive}$ has a strong adaptive ability: Δ is a constant and $\Delta_{ij}^{adaptive}$ can change for every different neuron at every different time unit during the algorithm execution. If the distance between the current probability p_{ij}^1 to the ideal probability (either 0 or 1) is big, the size of the probability adjustment step is

big too. If the distance between the current probability p_{ij}^1 to the ideal probability (either 0 or 1) is small, the size of the probability adjustment step is small too.

* Learning efficiency is significantly improved. If we want to ge 1 from a neuron, the ideal probability of p_{ij}^1 is 1. From an initial probability (e.g $p_{ij}^1 = 0.1$), very few steps (after 4 steps, $0.9 < p_{ij}^1 < 1$) are needed for using the adaptive learning rate $\Delta_{ij}^{adaptive}$ and much more steps are needed in OSNPS using Δ (after 40 steps, $0.3 < p_{ij}^1 < 0.9$). Similarly, if we want to get 0 from a neuron, the ideal probability of p_{ij}^1 is 0. From an initial probability (e.g $p_{ij}^1 = 0.9$), very few steps (after 4 steps, $0 < p_{ij}^1 < 0.1$) are needed for using the adaptive learning rate $\Delta_{ij}^{adaptive}$ and much more steps needed in OSNPS using Δ (after 40 steps, $0.1 < p_{ij}^1 < 0.7$). In addition, from an arbitrary initial probability, we can obtain $0.5 \leq p_{ij}^1 \leq 1$ or $0 \leq p_{ij}^1 \leq 0.5$ after only one step.
* The probability of p_{ij}^1 does not overflow. In OSNPS, it will definitely appear $1 < p_{ij}^1$ or $p_{ij}^1 < 0$, which is not allowed and the additional amendments must be needed if this situation occur. But the probability overflow never occurs in the paper, if the probability was adjusted with the adaptive learning rate $\Delta_{ij}^{adaptive}$.

3.2 Adaptive Mutation

In order to further improve the performance, we introduce the idea of mutation from gene and establish an adaptive mutation rule suitable for AOSNPS. Two dynamically changing parameters P_{m1} and P_{m2} are defined to characterize the evolutionary state of AOSNPS running process. P_{m1} is used to evaluate the evolutionary convergence of global optimal solutions and P_{m2} is used to evaluate the evolutionary diversity of probability matrices.

The parameter P_{m1} is defined as follows:

If $G_{bf}(gen) > G_{bf}(gen-1)$

$$P_{m1} = 0 \tag{8}$$

If $G_{bf}(gen) = G_{bf}(gen-1)$

$$P_{m1} = P_{m1} + \frac{1}{N_{ni\max gen}} \tag{9}$$

where G_{bf} represents the global best fitness and is always replaced by new better fitness values during algorithm execution in generation by generation. It is a function that only increases without decreasing (here we consider a maximization problem). $G_{bf}(gen)$ is the global best fitness at current generation, $G_{bf}(gen-1)$ is the global

best fitness at previous generation and $G_{bf}(0)$ is the global best fitness at the initialization. $N_{ni\max gen} \geq 1$ and $N_{ni\max gen} \in \mathrm{N}$. If the global best fitness does not improve for consecutive $N_{ni\max gen}$ generations, the algorithm terminates. Then, we have $P_{m1} \in [0, 1]$.

The parameter P_{m2} is defined as follows:

$$P_{m2} = \frac{DP_a(gen)}{DP_a(0)} \tag{10}$$

where DP_a is the average probability distance of all individuals in P_R shown in Eqs.(3). $DP_a(gen)$ is the average probability distance of all individuals at the current generation and $DP_a(0)$ is the average probability distance of all individuals at initialization. DP_a is calculated as

$$DP_a = \frac{2}{(H-1)(H-2)} \sum_{i=1}^{H-1} \sum_{j=i+1}^{H} \frac{1}{m} \sum_{k=1}^{m} \left(\left|p_{ik}^1 - p_{jk}^1\right|\right) \tag{11}$$

From Eqs.10, we know that P_{m2} is the ratio of current population diversity to population diversity at initialization. The probability matrix P_R is random initialization when initializing the population. In general, the diversity of the probability matrix population $DP_a(0)$ is good. Therefore, the general case is $0 \leq P_{m2} \leq 1$, but $P_{m2} > 1$ may be possible. However, if P_{m1} and P_{m2} satisfy the condition then the mutation rule will trigger.

$$rand_1() < P_{m1} \quad and \quad rand_2() > P_{m2} \tag{12}$$

where $rand_1()$ and $rand_2()$ are two random number in [0, 1]. Only when both $rand_1() < P_{m1}$ and $rand_2() > P_{m2}$ are satisfied, the mutation can be triggered. The mutation mode of the probability matrix P_R is designed such that if $rand_3() < P_j^m$, for every $i = 1, 2, \cdots, H$, $i \neq R_{bestfit}$, and $j = 1, 2, \cdots, m$, then

$$p_{ij}^1 = rand() \tag{13}$$

where $rand_3()$ and $rand()$ are two random number in the range [0, 1]. P_j^m is the mutating probabilities in the range [0, 0.1]. $R_{bestfit} \in [1, H]$ is the row coordinate of the best chromosome found at current generation. It means that the best chromosome does not participate in the mutation. The purpose is to protect the current optimal solution obtained.

The mutation rule has a strong adaptive ability. From Eqs.8 to Eqs.13, we know that P_{m1} and P_{m2} are two dynamically changing parameters that characterize the evolution performance (convergence and diversity) of the probability matrix P_R. The triggering rule for mutation is explained in detail in the following.

* In the execution of the AOSNPS, if the global optimal solution is higher than the previous generation, i.e, $(G_{bf}(gen) > G_{bf}(gen-1))$, then the mutation is not triggered because $P_{m1} = 0$.
* If the global optimal solution is not improved compared to the previous generation, i.e, $(G_{bf}(gen) = G_{bf}(gen-1))$, then the probability of triggering the mutation increases by generation to generation $(P_{m1} = P_{m1} + \frac{1}{N_{ni\max gen}})$. If the population diversity $(DP_a(gen))$ is good at current generation (compared to the population diversity at initialization $DP_a(0)$), then the probability of mutation being triggered is still small because P_{m2} is still big.
* If the global optimal solution (G_{bf}) is not improved for many generations, and the population diversity $(DP_a(gen))$ of the probability matrix P_R is poor, then the probability of triggering mutation is greatly increased, because P_{m1} is big, P_{m2} is small, and Eqs.12 is easy to satisfy at this time.

3.3 The Novel Guider of Adaptive Optimization Spiking Neural P System

Based on the adaptive learning rate and the adaptive mutation rule, a novel guider algorithm (Fig. 4) is obtained for solving a (specific) single objective and unconstrained combinatorial optimization problems.

To clearly understand the guider algorithm, we explain the details step by step as follows:

Step 1: Input the learning probabilities p_j^a and the mutating probabilities P_j^m, $1 \leq j \leq m$. Rearrange the input spike train T_s in the rule probability matrix P_R, where each row comes from the identical ESNPS and can be used to represent a chromosome or an individual in an optimization application. gen is used to accumulate the generations executed by the algorithm ($gen = 0$ represents initialization). $G_{bf}(0)$ is the best fitness at the initialization and $DP_a(0)$ represents the population diversity of P_R at initialization.

Step 2: If the parameter P_{m1} is greater than 1, i.e., $P_{m1} > 1$, then the algorithm goes to Step 25.

Input: Spike train T_s, p_j^a, P_j^m, H, m, and $N_{ni\max gen}$

Rearrange T_s as matrix P_R, let $gen = 0$ and $P_{m1} = 0$, calculate $G_{bf}(0)$ and $DP_a(0)$
while $(P_{m1} \leq 1)$ **do**
  $gen = gen + 1$
  $i = 1$
  while $(i \leq H)$ **do**
    j=1
    while $(j \leq m)$ **do**
      if $(rand < p_j^a)$ **then**
        $k_1, k_2 = ceil(rand * H), k_1 \neq k_2 \neq i$
        if $(f(C_{k_1}) > f(C_{k_2}))$ **then**
          $b_j = b_{k_1}$
        else
          $b_j = b_{k_2}$
        end if
        if $(b_j > 0.5)$ **then**
          $p_{ij}^1 = 0.5 + 0.5p_{ij}^1$
        else
          $p_{ij}^1 = 0.5p_{ij}^1$
        end if
      else
        if $(b_j^{max} > 0.5)$ **then**
          $p_{ij}^1 = 0.5 + 0.5p_{ij}^1$
        else
          $p_{ij}^1 = 0.5p_{ij}^1$
        end if
      end if
      $j = j + 1$
    end while
    $i = i + 1$
  end while
  calculate $G_{bf}(gen)$, $DP_a(gen)$ and $R_{bestfit}$
  if $(G_{bf}(gen) > G_{bf}(gen-1))$ **then**
    $P_{m1} = 0$
  else
    $P_{m1} = P_{m1} + \frac{1}{N_{ni\max gen}}$
  end if
  $P_{m2} = \frac{DP_a(gen)}{DP_a(0)}$
  if $(rand_1() < P_{m1} \quad and \quad rand_2() > P_{m2})$ **then**
    $i = 1$
    while $(i \leq H)$ **do**
      if $i \neq R_{bestfit}$ **then**
        $j = 1$
        while $(j \leq m)$ **do**
          if $rand_3() < P_j^m$ **then**
            $p_{ij}^1 = rand()$
          end if
          $j = j + 1$
        end while
      end if
      $i = i + 1$
    end while
  end if
end while

Output: Rule probability matrix P_R

Fig. 4. The Novel Guided Algorithm of AOSNPS

Step 3: Algorithm execution generation increases by one generation, i.e., $gen = gen + 1$.

Step 4: Assign the row indicator the initial value $i = 1$.

Step 5: If the row indicator is greater than its maximum H, i.e., $i > H$, then the algorithm goes to Step 12.

Step 6: Assign the column indicator the initial value $j = 1$.

Step 7: If the column indicator is greater than its maximum m, i.e., $j > m$, then the algorithm goes to Step 11.

Step 8: If a random number $rand$ is less than the prescribed learning probability p_j^a, the guider performs the following two steps. Otherwise, it goes to Step 9.

(i) Choose two distinct chromosomes k_1 and k_2 that differs from the ith individual among the H chromosomes, i.e., $k_1 \neq k_2 \neq i$. If $f(C_{k_1}) > f(C_{k_2})$ ($f(\cdot)$ is an evaluation function to an optimization problem. C_{k_1} and C_{k_2} denote the k_1th and k_2th chromosomes, respectively), i.e., if the k_1th chromosome is better than the k_2th one in terms of their fitness values (here we consider a maximization problem), the current individual learns from the k_1th chromosome, i.e., b_j=b_{k_1}. Otherwise, the current individual learns from the k_2th chromosome, i.e., b_j=b_{k_2}, where b_j, b_{k_1} and b_{k_2} are intermediate variable, jth bit of the k_1th and k_2th chromosomes, respectively.

(ii) If $b_j > 0.5$, we increase the current rule probability p_{ij}^1 to $0.5 + 0.5p_{ij}^1$. Otherwise, we decrease p_{ij}^1 to $0.5p_{ij}^1$.

Step 9: If $b_j^{max} > 0.5$, then current rule probability p_{ij}^1 is increased to $0.5 + 0.5p_{ij}^1$. Otherwise, p_{ij}^1 is decreased to $0.5p_{ij}^1$, where b_j^{max} is the jth bit of the best chromosome found.

Step 10: The column indicator j increases by 1 and the guider goes to Step 7.

Step 11: The row indicator i increases by 1 and the guider goes to Step 5.

Step 12: Calculate the global best fitness at current generation ($G_{bf}(gen)$), the population diversity at the current generation ($DP_a(gen)$) and the row coordinate of the best chromosome found at current generation ($R_{bestfit}$).

Step 13: Compared to the last generation, if the global optimal fitness is improved, i.e., $(G_{bf}(gen) > G_{bf}(gen-1)$, then $P_{m1} = 0$, otherwise $P_{m1} = P_{m1} + \frac{1}{N_{ni\max gen}}$ (i.e.,$(G_{bf}(gen) = G_{bf}(gen-1))$.

Step 14: Calculate the ratio of current population diversity to population diversity at initialization P_{m2}.

Step 15: If the conditions for triggering a mutation are met, i.e., $rand_1() < P_{m1}$ and $rand_2() > P_{m2}$, the guider performs the following eight steps (Step 16 to Step 23). Otherwise, it goes to Step 24.

Step 16: Assign the row indicator the initial value $i = 1$.

Step 17: If the row indicator is greater than its maximum H, i.e., $i > H$, then the algorithm goes to Step 24.

Step 18: If $i \neq R_{bestfit}$, then the guider performs the following four steps (Step 19 to Step 22). Otherwise, it goes to Step 23.

Step 19: Assign the column indicator the initial value $j = 1$.

Step 20: If the column indicator is greater than its maximum m, i.e., $j > m$, then the algorithm goes to Step 23.

Step 21: If a random number $rand_3()$ is less than the prescribed mutating probabilities P_j^m, i.e., $rand_3() < P_j^m$, then $p_{ij}^1 = rand()$ (mutating).

Step 22: The column indicator j increases by 1 and the guider goes to Step 20.

Step 23: The row indicator i increases by 1 and the guider goes to Step 17.

Step 24: The guider goes to Step 2.

Step 25: The guider outputs the modified rule probability matrix P_R to adjust each probability value of each evolution rule inside each of neurons $1, \ldots, m$ in each ESNPS.

4 Experimentation and Analysis of Results

To illustrate the feasibility and effectiveness of AOSNPS for solving combinatorial optimization problems, and to compare with OSNPS, we chose the same knapsack problems as an application example to conduct experiments.

4.1 Knapsack problems

The Knapsack problem is an NP-complete problem with combinatorial optimization. The problem can be described as: given a group of items, each item has its own weight and price. So within a limited total capacity, how we choose that to make the total price of the items highest. The mathematical description of the knapsack problem is to select a subset from the given number of items to maximize the profit $f(x)$:

$$f(x) = \sum_{i=1}^{K} p_i x_i \tag{14}$$

subject to

$$\sum_{i=1}^{K} \omega_i x_i \leq C \tag{15}$$

where K is the number of items, p_i is the profit of the ith item, ω_i is the weight of the ith item, C is the capacity of the given knapsack, and x_i is either 0 or 1.

This study uses strongly correlated sets of unsorted data, i.e., the knapsack problem has a linear relationship between the weights and profit values of unsorted items. It was used in [35, 66–69] to test the algorithm performance.

$$\omega_i = uniformly\ random[1, \Omega] \tag{16}$$

$$p_i = \omega_i + \frac{1}{2}\Omega \tag{17}$$

where Ω is the upper bound of ω_i, $i = 1, \ldots, K$, and the average knapsack capacity C is

$$C = \frac{1}{2}\sum_{i=1}^{K} \omega_i \tag{18}$$

4.2 Comparative Analysis of Experimental Results of AOSNPS and OSNPS

In this subsection, we use AOSNPS and OSNPS to solve the same Knapsack problem (with the same items),respectively. In order to illustrate the difference between

AOSNPS and OSNPS in solving the optimization problem, we performed a dynamic behavior analysis of seven aspects (four of these indicators have also been discussed in [70]) of the experimental testing process, including:

(1) G_{bf}: global optimal fitness convergence trend. A larger value of G_{bf} gives a better solution with regard to a maximization optimization problem.

(2) D_{qbw}: binary-bit distance between the best and worst binary-bit individuals corresponding to the best and worst fitness values in a population, respectively. A larger value of D_{qbw} gives a hint of larger distance between the best and worst binary-bit individuals.

(3) D_{qa}: average binary-bit distance of all binary-bit individuals in a population. A larger value of D_{qa} suggests a larger distance between each pair of binary-bit 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.

(5) D_{hm}: 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.

(6) DP_{bw}: distance between the best and worst probability individuals in P_R corresponds to the best and worst fitness values in a population, respectively. A larger value of DP_{bw} gives a hint of larger distance between the best and worst probability individuals in P_R.

(7) DP_a: average probability distance of all probability individuals. A larger value of DP_a indicates more varieties between each pair of probability individuals in P_R.

The calculation method of G_{bf} is determined according to the specific optimization problem (for Knapsack problems, according to Eqs.(14) and taking the maximum). For the detailed description and calculation method of D_{qbw}, D_{qa}, D_{hbw} and D_{hm} should see [70]. DP_a was defined in Eqs.(11). DP_{bw} is defined as follows:

$$DP_{bw} = \frac{1}{m}\sum_{k=1}^{m}\left|p_{bk}^{1} - p_{wk}^{1}\right| \tag{19}$$

where p_{bk}^1 and p_{wk}^1 are the the best and worst probability individuals in P_R corresponding to the best and worst fitness values in a population, respectively.

At first, we use OSNPS and AOSNPS to solve the the same knapsack problem with 1000 items ($\Omega = 50$), respectively. Both OSNPS and AOSNPS consist of $H = 50$ ESNPS, and each of which has 1002 neurons ($m = 1000$).

In OSNPS, the learning probability p_j^a ($j = 1, \ldots, m$) and the learning rate Δ are prescribed as a random number in the range [0.05, 0.20] and [0.005, 0.02], respectively (the same initialization conditions were used in [45]).

In AOSNPS, the learning probability p_j^a ($j = 1, \ldots, m$) uses the same value used in OSNPS (a random number in [0.05, 0.20]), the mutating probabilities $p_j^m = 0.01$ ($j = 1, \ldots, m$) and $N_{ni\max gen} = 500$.

It is worth to pointing out that, since algorithm OSNPS does not specify the conditions for calculating how many generations to stop, we artificially set the total execution generations to 8000 (the average generations over 30 independent runs were 7123 as shown in [45]). In order to better compare the dynamic behavior of OSNPS and AOSNPS, the total execution generations of AOSNPS are also set to 8000 (e.g., in Step 2, ($P_{m1} \leq 1$) is replaced by ($gen \leq 8000$) in Fig. 4).

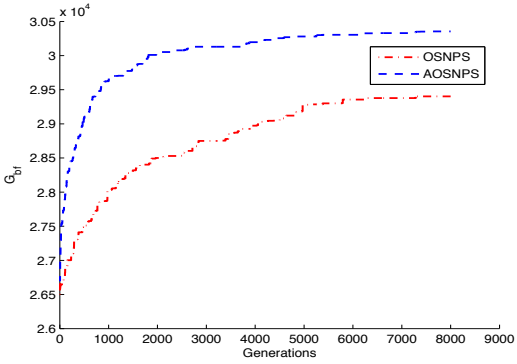


Fig. 5. Experimental result G_{bf} of OSNPS and AOSNPS

Based on the same random initial probability matrix $P_{R50\times1000}$, we obtained the comparisons shown in Fig.5 - 11.

From Fig.5, we can see that after performing the same evolutionary optimization generations, AOSNPS (30351) captures a better solution than OSNPS (29402). In ad-

dition, AOSNPS exhibits better transient convergence response performance due to the contributions of adaptive learning rate.

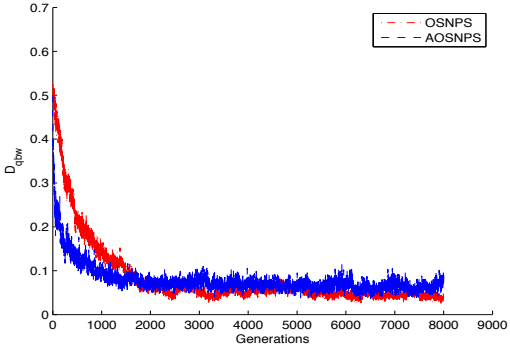


Fig. 6. Experimental result D_{qbw} of OSNPS and AOSNPS

In binary-bit space, D_{qbw} and D_{qa} can be regarded as genotypic diversity measures. D_{hbw} and D_{hm} can be regarded as phenotypic diversity measures indicated in [45, 71]. From Fig.6 - 9, we can see that, in the early stage, the four diversity evaluation indicators D_{qbw}, D_{qa}, D_{hbw} and D_{hm} of AOSNPS drop more rapidly than that of OSNPS. But in the middle and late stages, AOSNPS maintains a better diversity than OSNPS. This is the result of the joint contributions of the adaptive learning rate (the rapid decline of diversity in early stage is mainly due to the improved learning efficiency using $\Delta_{ij}^{adaptive}$) and the adaptive mutation (better diversity obtained in the middle and late stages is due to the frequent triggering of mutation rule).

In probability individuals space in P_R, the experimental results of DP_{bw} and DP_a are shown in Fig.10 - 11. They also illustrate the better ability of AOSNPS to maintain the diversity of probability individuals in comparison with OSNPS. This is due to the contribution of the adaptive mutation. During the entire execution of the AOSNPS (8000 generations), the adaptive mutation is triggered 1971 times. The first time to trigger mutation is at the 572th generation, the second time to trigger mutation is at the 629th generation, etc. The histogram of the distribution of mutation triggered times during running AOSNPS 8000 generations is shown in Fig.12.

At the beginning of the algorithm execution, the mutation was not enabled because the optimal solution improved quickly. In the middle and later stages of algorithm ex-

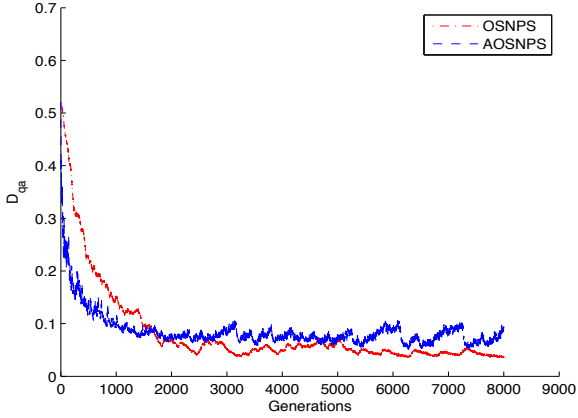


Fig. 7. Experimental result D_{qa} of OSNPS and AOSNPS

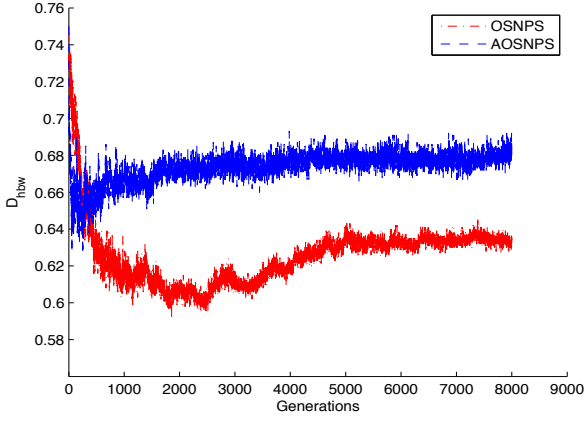


Fig. 8. Experimental result D_{hbw} of OSNPS and AOSNPS

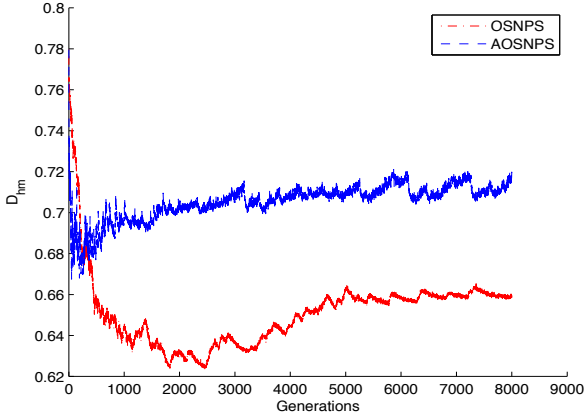


Fig. 9. Experimental result D_{hm} of OSNPS and AOSNPS

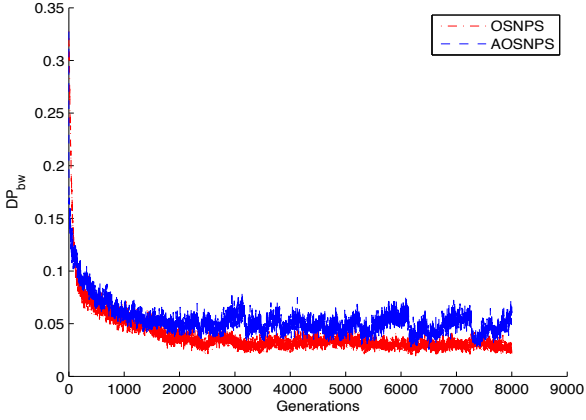


Fig. 10. Experimental result DP_{bw} of OSNPS and AOSNPS

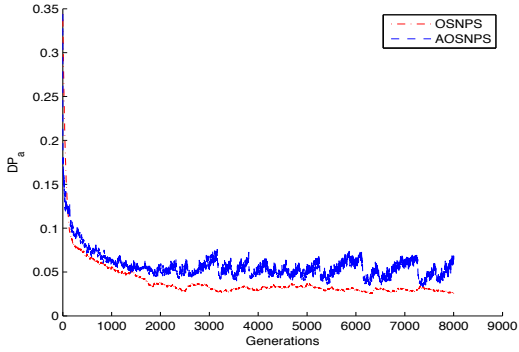


Fig. 11. Experimental result DP_a of OSNPS and AOSNPS

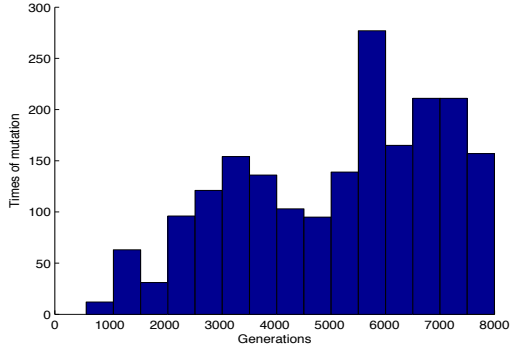


Fig. 12. The histogram of the distribution of mutation triggered times

ecution, the global optimal solution grows weakly, and the mutation is often enabled to explore possible better solutions. In order to better understand the dynamic working behavior of adaptive mutation, we count the interval generations between two adjacent mutations triggered as shown in Fig.13.

Statistical results shown in Fig.13 well validated the adaptive working mechanism of mutation. More specifically, when the interval generation of mutation is one generation, it indicates that mutation has occurred in two consecutive generations. At this time, in AOSNPS, the global optimal solution has not been improved for many generations (P_{m1} is large), and the difference between the populations is small (P_{m2} is small). So it is trapped in the local optimum. Therefore, mutations are needed to explore possible

new excellent solution. If the interval generation of mutation is large, AOSNPS can obtain new global optimal solution (P_{m1} is reset to zero) or the difference between the populations is large (P_{m2} is large).

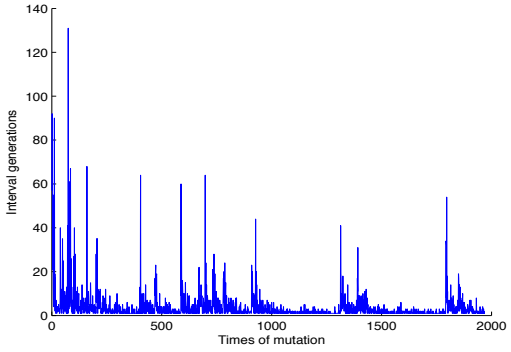


Fig. 13. The interval generations between two adjacent mutations triggered

To further test the optimization capability of AOSNPS, we solved the Knapsack problem with 5000, 6000, 7000, 8000, 9000 and 10000 items separately. For Knapsack problem with different items, the solution set space is different, and it is unreasonable to solve them with a certain execution generations. Therefore, both algorithms use the same termination criteria: when the global optimal solution has not improved for 500 consecutive generations, the algorithm terminates. The total execution generations (TG) of AOSNPS and OSNPS and the final optimal solution (OS) explored are shown in Table 1.

From Table 1, we can see that comparison with the OSNPS, AOSNPS got a better optimal solution when solving the Knapsack problem with 5000, 6000, 7000, 8000, 9000 and 10000 items respectively. Furthermore, under the same algorithm termination condition (the global optimal solution has not improved for 500 consecutive generations, the algorithm terminates), the total execution generations of AOSNPS are much more than OSNPS which indicated the much stronger exploration and exploitation abilities of solving combinatorial optimization problem such as Knapsack problem.

Table 1. The total execution generations of AOSNPS and OSNPS and the final optimal solution

K-problems	OSNPS		AOSNPS	
	OS	TG	OS	TG
5000	138011	10373	142716	8841
6000	165366	14924	172901	29606
7000	190531	9248	199617	24002
8000	217610	11385	225838	18272
9000	244211	11897	256390	40041
10000	271191	13782	283167	38280

4.3 An Application to Power System Fault Diagnosis

In this subsection, AOSNPS is applied to solve the problem of power system fault diagnosis with 39 nodes. In Fig.14, a typical IEEE 39 node electric power system, is used to carry out the simulation. It contains 39 buses, 45 lines and 99 Circuit breakers. 39 buses and 45 lines are $B_1, ..., B_{39}$, and, $L_{1-7}, ..., L_{29-39}$, 99 Circuit breakers are $CB_{(1)-7}, ..., L_{(39)-29}$, where, $CB_{(1)-7}$ represents the breaker that L_{1-7} is near the B_1 side. For instance, line L_{3-4} has three types of protective relays including main protective relay $L_{(3)-4m}$ and $L_{3-(4)m}$, first backup protective relay $L_{(3)-4p}$ and $L_{3-(4)p}$,and second backup protective relay $L_{(3)-4s}, L_{3-(4)s}, L_{4-(5)s}$ and $L_{4-(8)s}$. The operational rules of the protective relays of buses and lines are described in the following manner [38].

(1) Protective relays of buses

If the main protective relays of a bus operate, all breakers connected to the bus will be tripped. For example, if bus B_{11} fails, the main protective relays B_{11m} of the bus B_{11} operates to trip $CB_{(11)-18}, CB_{7-(11)}$ and $CB_{(11)-12}$. Similarly, if bus B_{31} fails, the main protective relay B_{11m} of the bus B_{31} operates to trip $CB_{(31)-33}, CB_{26-(31)}$ and $CB_{27-(31)}$.

(2) Protective relays of lines

If the main protective relays of a line operate, all breakers connected to the line will be tripped. For instance, if line L_{18-19} fails, the main protective relays $L_{(18)-19m}$ and $L_{18-(19)m}$ of the line L_{18-19} operate to trip $CB_{(18)-19}$ and $CB_{18-(19)}$, respectively. Likewise, when the main protective relays of a bus fail to operate, the first backup protective relays operate to trip all breaker connected to the line. For example, when line L_{18-19} fails and the main protective relay $L_{(18)-19m}$ fails to

operate, the first backup relay $L_{(18)-19p}$ operate to trip breaker $CB_{(18)-19}$. If adjacent regions of a line fail and their main protective relay and first backup relay fail to operate, then the second backup relay of a line operate. For instance, if B_{20} fails and breaker $CB_{18-(19)}$ fails to operate, the second protective relay $CB_{(18)-19s}$ of line $CB_{(18)-19}$ operate to trip $CB_{(18)-19}$.

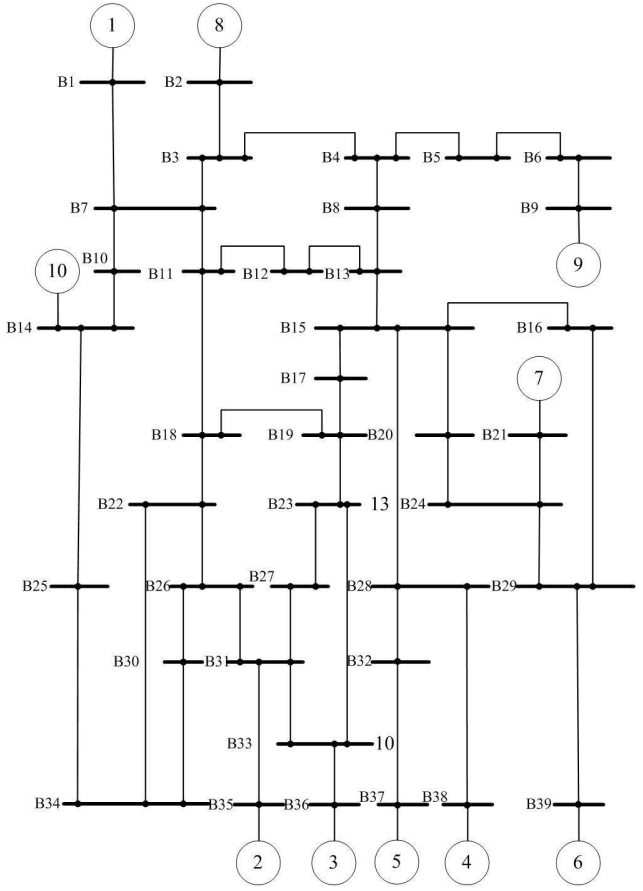


Fig. 14. IEEE 39 node electric power system

Four cases in typical IEEE 39 node electric power system is considered. Some status information about four cases in Fig.14 is shown in Table 2, where, case 1 is a single fault, case 2 is multiple faults, case 3 and case 4 are multiple faults with incompleteness

and uncertainty. Four cases are estimated by the adaptive optimization spiking neural P systems methods. The estimation results are shown in Table 2 with comparison with other fault diagnosis methods.

Table 2. Status information about four cases

case	protective relays	breakers
1	$L_{(18)-19m}$operated,$L_{18-(19)p}$ operated $L_{18-(19)m}$ refused to operated	$CB_{(18)-19}$ operated,$CB_{18-(19)}$ operated
2	$L_{(18)-19m}$ operated, $L_{18-(19)m}$ operated $L_{(17)-19m}$ operated, $L_{19-(23)s}$ operated $L_{(17)-19m}$ operated, $L_{19-(23)s}$ operated	$CB_{(18)-19}$ operated,$CB_{17-(19)}$ operated $CB_{19-(23)}$ operated $CB_{18-(19)}$ refused to operate
3	$L_{(18)-19m}$ operated, $L_{18-(19)m}$ operated B_{19m} operated, $L_{(17)-19s}$ operated $L_{19-(23)m}$ misoperation	$CB_{(18)-19}$operated, $CB_{18-(19)}$ operated $CB_{(17)-19}$ operated, $CB_{19-(23)}$ operated $CB_{17-(19)}$ refused to operated
4	$L_{(11)-12m}$ operated, $L_{11-(12)m}$ operated $L_{12-(13)m}$ operated, $L_{(18)-19p}$ operated $L_{18-(19)p}$ operated,$L_{(12)-13m}$ misoperation $L_{(18)-19m}$ refused to operated $L_{18-(19)m}$ refused to operated	$CB_{(11)-12}$ operated, $CB_{12-(13)}$ operated $CB_{(17)-19}$ operated, $CB_{17-(19)}$ operated $CB_{11-(12)}$ refused to operated

In Table 3, the result of the estimation of AOSNPS is the same as [72] and [73] in case 1. The estimation result of AOSNPS is the same as [73], but is different with [73] in case 2. From case 3 and case 4, we known that the estimation results of AOSNPS are different from those in [72] and [73]. From [74], we learn that the results of AOSNPS are correct. Therefore, in the previous examples, AOSNPS is more effective and precise than other methods in [72] and [73] in fault section estimation of power systems.

The fault section estimation process of case 3 is described in detail as follows:

(1) According to SCADA data, operated protective relays are $L_{(18)-19m}$, $L_{18-(19)m}$, B_{19m} and $L_{(17)-19s}$, tripped breakers are $CB_{(18)-19}$, $CB_{18-(19)}$, $CB_{(17)-19}$, $CB_{19-(23)}$ and $CB_{(19)-23}$. We use network topology analysis method which is descried in detailed in [75] to identify passive networks, which is shown in Fig.15, where, B_{19m}, L_{17-19}, L_{18-19} and L_{19-23} are candidate faulty sections and their corresponding status vector is $S = [s_1, s_2, s_3, s_4]$.

(2) From the Fig.15, the real status vector of the protective relays is $R = [r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}, r_{17}, r_{18}, r_{19}] = [1, 0, 0, 1, 1, 0, 1, 0,$

$0,0,0,0,0,1,0,0,0,0,0]$, where, r_1∼r_{19} represent $B_{19m}, L_{(17)-19m}, L_{17-(19)m}$, $L_{(18)-19m}, L_{18-(19)m}, L_{(19)-23m}, L_{19-(23)m}, L_{(17)-19p}, L_{17-(19)p}, L_{(18)-19p}$, $L_{18-(19)p}$,$L_{(19)-23p}, L_{19-(23)p}, L_{(17)-19s}, L_{17-(19)s}, L_{(18)-19s}, L_{18-(19)s}$, $L_{(19)-23s}, L_{19-(23)s}$, respectively. The real status vector of the breakers is $C = [c_1, c_2, c_3, c_4, c_5, c_6]$, where, c_1∼c_6 represent $CB_{(17)-19}, CB_{17-(19)}, CB_{(18)-19}$, $CB_{18-(19)}, CB_{(19)-23}, CB_{19-(23)}$, respectively.

Table 3. Comparisons between AOSNPS and two methods

case	candidate sections	The results in [72]	The results in [73]	The results in this study
1	L_{18-19}	L_{18-19}	L_{18-19}	L_{18-19}
2	B_{19}, L_{17-19}, L_{18-19}, L_{19-23}	L_{17-19},L_{18-19}	L_{18-19}	L_{18-19}
3	B_{19}, L_{17-19}, L_{18-19}, L_{19-23}	B_{19}, L_{17-19}, L_{18-19}	B_{19}, L_{17-19}, L_{18-19}	B_{19}, L_{18-19}
4	B_{12}, L_{11-12}, L_{12-13}, L_{17-19}	B_{12}, L_{12-13}, L_{17-19}	B_{12}, L_{11-12}, L_{17-19}	L_{11-12}, L_{17-19}

(3) According to the logical relationship among section fault, protective relays and the corresponding circuit breakers. The expected values of the protective relays and circuit breakers are computed in [76]. We will utilize a 0-1 integer programming model with an objective function, as shown in Eqs.(20), which is obtained according to the relation between a fault and the status of protective relays and circuit breakers in [77].

$$E(S) = \sum_{i=1}^{N_r} |r_i - r_{ei}(S)| + \sum_{j=1}^{N_c} |c_j - c_{ej}(S,R)| \tag{20}$$

Where:

(a) N_r and N_c represent the numbers of the protective relays and circuit breakers, respectively;

(b) E(S) represents a status function of all the sections in power system;

(c) S is an n dimension vector, where, s_i represents the status of section, $s_i = 1$ and $s_i = 0$ represent the fault status and normal status of section i, respectively;

(d) r_i $(1 \leqslant i \leqslant N_r)$ represents the real status of the protective relay i. $r_i = 1$ and $r_i = 0$ represent the operation and non-operation of the protective relay i,

respectively. $r_{ei}(1 \leqslant i \leqslant N_r)$ represents the expected status of the protective relay i. If the ith protective relay is expected to operate,then $r_{ei} = 1$, otherwise, $r_{ei} = 0$;

(e) $c_j(1 \leqslant j \leqslant N_c)$ represents the real status of the circuit breaker j. If the jth circuit breaker trip, then $c_j = 1$, otherwise, $c_j = 0$. $c_{ej}(S, R)(1 \leqslant j \leqslant N_c)$ represents the expected status of the circuit breaker j. If the jth circuit breaker trip, then $c_{ej}(S, R) = 1$, otherwise,$c_{ej}(S, R) = 0$. Therefore, in case 3, we obtain the expected value vector $r_e(S)$ of the protective relays and the expected value vector $c_e(S, R)$. The circuit breakers,where,$r_e = [r_{e1}, r_{e2}, r_{e3}, r_{e4}, r_{e5}, r_{e6}, r_{e7}, r_{e8}, r_{e9}, r_{e10}, r_{e11}, r_{e12}, r_{e13}, r_{e14}, r_{e15}, r_{e16}, r_{e17}, r_{e18}, r_{e19}] = [s_1, s_2, s_2, s_3, s_3, s_4, s_4, s_2, s_2, 0, 0, s_4, s_4, s_1, 0, 0, 0, 0, 0]$, $c_e = [c_{e1}, c_{e2}, c_{e3}, c_{e4}, c_{e5}, c_{e6}] = [s_1, s_1, 0, s_1, s_1, s_4]$.

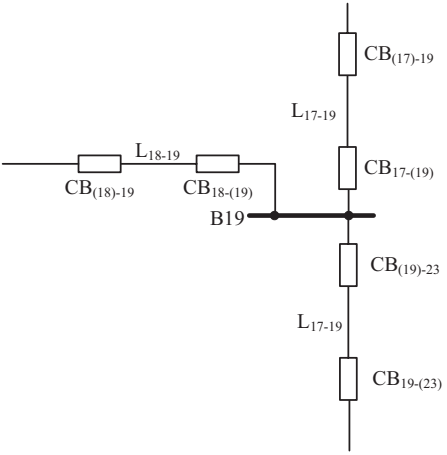


Fig. 15. Passive network

(4) According to the real status vector r and the expected status vector $r_e(S)$ of the protective relays and the real status vector c and the expected status vector $c_e(S, R)$ of the circuit breaker, we get that $E(S) = 11 - 4s_1 + 4s_2 - 2s_3 + s_4$.Therefore, we can obtain the minimum value of the objective function by performing AOSNPS algorithm. First of all, there are some parameters which need to be set in the AOSNPS algorithm. For example, in this study, the number of ESNPS is 100(H=50), the value of the learning probability p_j^a $(j = 1, \ldots, m)$ is set as a random number in

[0.05, 0.20], which is the same with the learning probality in OSNPS, the mutating probabilities $p_j^m = 0.01$ $(j = 1, \ldots, m)$ and $N_{ni\max gen} = 500$. Then perform AOSNPS algorithm. If s_i=1, there are faults in the ith section. Otherwise, the ith section is normal. Finally, B_{19} and L_{18-19} are diagnosed as the fault section after performing AOSNPS algorithm.

In the case simulations above, the results of case simulation manifest that AOSNPS algorithm is viable and effective to power system fault diagnosis. It is practicable for AOSNPS algorithm to deal with single fault, mutiple fault and complex fault. From table 6, the results of the study are more correct than the method in [72].

5 Conclusions

As an extension of the work in [45], an effective AOSNPS is proposed to solve combinatorial optimization problems. In this study, an adaptive learning rate $\Delta_{ij}^{adaptive}$ is ingeniously designed to adjust the probability of P_R. Two dynamically changing indicator parameters P_{m1} and P_{m2} are used to evaluate the evolutionary convergence of global optimal solutions and diversity of probability matrices of the algorithm running process, a triggering rule of mutation and a probability-based mutation mode constitute an adaptive mutation. With the ingenious and reasonable adaptive learning rate and adaptive mutation, the efficiency of probability adjustment of the guider is greatly improved and the probability of the guider never goes beyond the upper or lower bound. The exploration and exploitation abilities of solving combinatorial optimization problems are significantly improved by the AOSNPS and the better balance between convergence and diversity is captured. Finally, experimental results and analysis illustrate the effectiveness of AOSNPS. More diverse probabilistic learning mechanisms, mutation modes and more applied study are interesting subjects for further research.

Acknowledgment

This work was supported by Scientific Research Fund of Sichuan Provincial Science and Technology Department under Grant (2015JY0257, 2017FZ0010), the National Natural Science Foundation of China (61672437, 61702428), the Sichuan Science and Technology Program (2018GZ0185, 2018GZ0086), New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0044) and the Fundamental Research Funds for the Central Universities (2682016CX038).

References

1. Lettvin, J., Maturana, H., McCulloch, W., Pitts, W.: What the Frog's Eye Tells the Frog's Brain. Proceedings of the IRE, 11(47), 1940-1951(1959).
2. Sussillo, D., Churchland, M., Kaufman, M., Shenoy, K.: A neural network that finds a naturalistic solution for the production of muscle activity. Nature Neuroscience, 18(7), 1025-1033(2015).
3. S*ü*dhof, T.: Synaptic Neurexin Complexes: A Molecular Code for the Logic of Neural Circuits. Cell, 171(4), 745-769(2017).
4. Cen, Z., Wei, J., Jiang, R.: A gray-box neural network-based model identification and fault estimation scheme for nonlinear dynamic systems. International Journal of Neural Systems, 23(6), 497-383(2013).
5. Ding, S., Li, H., Su, C., Yu , J., Jin, F.: Evolutionary artificial neural networks: a review. Artificial Intellience Review, 39(3), 251-260(2013).
6. Cichocki, A., Unbehauen, R.: Neural networks for optimization and signal processing. John Wiley and Sons, 74(1), 245-250(1992).
7. Ciresan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. Computer Vision and Pattern Recognition, 175(10), 3642-3649(2012).
8. Ahmadlou, M., Adeli, H.: Enhanced probabilistic neural network with local decision circles: A Robust Classifier. Integrated Computer-Aided Engineering, 17(3), 197-210(2010).
9. Bishop, C.: Neural networks for Pattern Recognition. Oxford, U.K.:Oxford University, (1995).
10. Kasabov, N., Dhoble, K., Nuntalid, N., Indiveri, G.: Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition. Neural Networks, 41(5), 188-201(2013).
11. Adeli, H., Panakkat, A.: A probabilistic neural network for earthquake magnitude prediction. Neural Networks, 22(7), 1018-1024(2009).
12. Panakkat, A., Adeli, H.: Neural network models for earthquake magnitude prediction using multiple seismicity indicators. International Journal of Neural Systems, 17(1), 13-33(2007).
13. Panakkat, A., Adeli, H.: Recurrent neural network for approximate earthquake time and location prediction using multiple seismicity indicators. Computer-Aided Civil and Infrastructure Engineering, 24(4), 280-292(2009).
14. Ghosh-Dastidar, S., Adeli, H., Dadmehr, N.: Mixed-band wavelet-chaos-neural network methodology for epilepsy and epileptic seizure detection. IEEE Transactions on Biomedical Engineering, 54(9), 1545-1551(2007).
15. Ghosh-Dastidar, S., Adeli, H., Dadmehr, N.: Principal component analysis-enhanced cosine radial basis function neural network for robust epilepsy and seizure detection. IEEE Transactions on Biomedical Engineering, 55(2), 512-518(2008).

16. Adeli, H, Park, H.: Optimization of space structures by neural dynamics. Neural Networks, 8(5), 769-781(1995).
17. Ahmadkhanlou, F., Adeli, H.: Optimum cost design of reinforced concrete slabs using neural dynamics model. Engineering Applications of Artificial Intelligence, 1(1), 65-72(2005).
18. Atencia, M., Joya, G., Sandoval, F.: Dynamical analysis of continuous higher-order Hopfield networks for combinatorial optimization. Neural Computation, 17(8), 1802-1819(2005).
19. Cheng, L., Hou, Z., Lin, Y., Tan, M., Zhang, W., Wu, F.: Recurrent neural network for non-smooth convex optimization problems with application to the identification of genetic regulatory networks. IEEE Transactions on Neural Networks, 25(2), 714-726(2011).
20. Chau, T., Yu S., Fernando T., Iu, H., Small, M.: A load-fforecasting-based adaptive parameter optimization strategy of STATCOM using ANNs for enhancement of LFOD in power systems. IEEE Transactions on Industrial Informatics, 14(6), 2463-2472(2018).
21. Pitts, w., McCulloch, W.: A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys, 5, 115-133(1943).
22. Maass W.: Networks of Spiking Neurons: the Third Generation of Neural Network Models. Neural Networks, 10(9), 1659-1671(1997).
23. Fiete, I., Senn, W., Wang, C., Hahnloser, R.: Spike-time-dependent plasticity and heterosynaptic competition organize networks to produce long scale-free sequences of neural activity. Neuron, 65(4), 563-576(2010).
24. Maass W.: Lower bounds for the computational power of networks of spiking neurons. Neural Computation, 8(1), 1-40(1996).
25. Ghosh-Dastidar, S., Adeli, H.: A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. Neural Networks, 22(10), 1419-1431(2009).
26. Ponulak, F., Kasinski, A.: Introduction to spiking neural networks: Information processing, learning and applications. Acta Neurobiologiae Experimentalis, 71(4), 409-433(2011).
27. Schliebs, S., Kasabov, N., Defoin-Platel, M.: On the probabilistic optimization of spiking neural networks. International Journal of Neural Systems, 20(6), 481-500(2010).
28. Gheorghe P.: Computing with Membranes. Journal of Computer and System Sciences, 61(1), 108-143(2000).
29. Manca, V., Bianco, L.: Biological networks in metabolic P systems. Bio Systems, 91(3), 489-498(2008).
30. Frisco, P., Gheorghe, M., Pérez-Jiméne, M.: Applications of membrane computing in cystems and cynthetic biology. Heidelberg: Springer, 7(9), S624(2014).
31. Frisco, P., Gheorghe, M., Pérez-Jiménez M.: Applications of Membrane Computing in Systems and Synthetic Biology. Emergence, Complexity and Computation, (2014).
32. Zhang, G., Pérez-Jiménez, M., Gheorghe, M.: Real-life Applications with Membrane Computing. Emergence, Complexity and Computation, (2017).

33. Buiu, C., Vasile, C., Arsene, O.: Development of membrane controllers for mobile robots. Information Sciences, 187(1), 33-51(2012).
34. 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(1), 15-30(2015).
35. Zhang, G., Gheorghe, M., Wu, C.: A quantum-inspired elutionary algorithm based on P systems for Knapsack Problem. Fundamenta Informaticae, 87(1), (2008).
36. Zhu, M., Zhang, G., Yang, Q., Rong, H., Yuan, W., Pérez-Jiménez, M.: P Systems-Based Computing Polynomials With Integer Coefficients Design and Formal Verification. IEEE Transactions on Nanobioscience, 17(3), 272-280(2018).
37. 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).
38. 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).
39. Ciobanu, G., Pérez-Jiménez, M., P?un, G.: Applications of membrane computing. Berlin: Springer, (2006).
40. Zhang G., Gheorghe, M., Pan, L., Mario J. Pérez-Jiménez: Evolutionary membrane computing: A comprehensive survey and new results. Information Sciences, 279, 528-551(2014).
41. Wu, T., Zhang, Z., Paun, G., Pan, L.: Cell-like spiking neural P systems, Theoretical Computer Science, 623: 180-189(2016).
42. 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).
43. Pan, L., P?un, G., Zhang, G., Neri, F.: Spiking neural P systems with communication on request. International Journal of neural systems, 27(08), 2017.
44. Wu, T., Bilbie, F.-D., Paun, A., Pan, L., Neri, F.: Simplified and yet Turing universal spiking neural P systems with communication on request, International Journal of Neural Systems, 28(8), (2018).
45. 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), 01-16(2014).
46. Ionescu, M., Păun, G., Yokomori, T.: Spiking Neural P Systems. Fundamenta Informaticae, 71(2, 3), 279-308(2006).
47. Wu, T., Wang, Y., Jiang, S., Su, Y., Shi, X.: Spiking neural P systems with rules on synapses and anti-spikes. Theoretical Computer Science, 16(8), 888-895(2017).
48. Song, T., Rodríguez-Patón, A., Zheng, P., Zeng, X.: Spiking Neural P systems with Colored Spikes. IEEE Transactions on Cognitive and Development Systems, (2018).
49. Francis George C. Cabarle, Henry N. Adorna, Min Jiang, Zeng X.: Spiking Neural P systems with Scheduled Synapses. IEEE Transactions on Nanobioscience, 16(8), 792-801(2017).

50. Wu, T., Păun, A., Zhang, Z., Pan, L.: Spiking Neural P systems With Polarizations. IEEE Transactions on Neural Networks and Learning Systems, 8(29), 3349-3360(2018).
51. Kociecki, M., Adeli, H.: Two-phase genetic algorithm for size optimization of free form steel space-frame roof structures. Journal of Constructional Steel Research, 90(9), 283-296(2013).
52. Zhang, G., Zhou, F., Huang, X., Cheng, J., Gheorghe, M., Ipate, F., Lefticaru, R.: A novel membrane algorithm based on particle swarm optimization for solving broadcasting problems. Journal of Universal Computer Science, 13(18), 1821-1841(2012).
53. Huang, L., He, X., Wang, N., Xie, Y.: P Systems Based Multi-Objective Optimization Algorithm. Progress in Natural Science, 17(4), 458-465(2007).
54. 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(3), 1528–1542(2013).
55. Takase, T., Oyama, S., Kurihara, M.: Effective neural network training with adaptive learning rate based on training loss. Neural Networks, 101, 68-78(2018).
56. Srinivas, M., Patnaik, L.: Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Transactions on System, Man and Cybernetics, 24(4), 656-667(1994).
57. Zhang, J., Chung, S., Lo, W.: Clustering-based adaptive crossover and mutation probabilities for genetic algorithms. IEEE Transactions on Evolutionary Computation, 11(3), 326-335(2007).
58. Zhan, Z., Zhang, J., Li, Y., Chung, H.: Adaptive particle swarm optimization. IEEE Transactions on System, Man and Cybernetics, 39(6), 1362-1380(2009).
59. Wang, H., Yen, G.: Adaptive multiobjective particle swarm optimization based on parallel cell coordinate system. IEEE Transactions on Evolutionary Computation, 19(1), 1-18(2015).
60. Dorigo, M., Gambardella, L.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1(1), 53-66(1997).
61. Hsin, H., Chang, E., Su, K., Wu, A.: Ant colony optimization-based adaptive network-on-chip routing framework using network information region. IEEE Transactions on Computers, 64(8), 2119-2131(2015).
62. Ge, S., Hang, C., Woon, L.: Adaptive neural network control of robot manipulators in task space. IEEE Transactions on Industrial Electronics, 44(6), 746-752(2002).
63. Denève, S., Alemi, A., Bourdoukan, R.: The Brain as an Efficient and Robust Adaptive Learner. Neuron, 94(5), 969-977(2017).
64. Wang, J., Belatreche, A., Maguire, L., Mcginnity, T.: An online supervised learning method for spiking neural networks with adaptive structure. Neurocomputing, 14(20), 526-536(2014).
65. Wang, J., Belatreche, A., Maguire, L., Mcginnity, T.: Spike Temp: An enhanced rank-order-based learning approach for spiking neural networks with adaptive structure. IEEE Transactions on Neural Networks and Learning Systems, 28(1), 30-43(2017).

66. Han, K., Kim, J.: Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE Transactions on Evolutionary Computation, 6(6), 580-593(2002).
67. Han, K., Kim, J.: Quantum-inspired evolutionary algorithms with a new termination criterion, Hepsilon gate, and two-phase scheme. IEEE Transactions on Evolutionary Computation, 8(2), 156-169(2004).
68. Zhang, G.: Quantum-inspired evolutionary algorithms: a survey and empirical study. Journal of Heuristics, 17(3), 303-351(2011).
69. 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).
70. Zhang, G., Cheng, J., Gheorghe, M.: Dynamic behavior analysis of membrane-inspired evolutionary algorithms. International Journal of Computers, Communications and Control, 9(2), 227-242(2014).
71. Burke, E., Gustafson, S., Kendall, G.: Diversity in genetic programming: an analysis of measures and correlation with fitness. IEEE Transactions on Evolutionary Computation, 8(1), 47-62(2004).
72. Sun, J., Qin, S., Song, Y.: Fault diagnosis of electric power systems based on fuzzy Petri nets. IEEE Transactions on Power Systems, 19(4), 2053-2059(2004).
73. Yang, J., He, Z.,: Ower System Fault diagnosis approach based on time sequence fuzzy petrinet. Automation of Electric Power Systems, 35(15), 46-51(2011).
74. Wu, K., Wen, F., Xue, Y., Zhou, H., Li, X.: Fault diagnosis model of time-delay constrained weighted fuzzy Petri nets based on multi-source information. Automation of electric power system, 37(24), 43-53(2013).
75. Tang, L., Sun, H., Zhang, B., Gao, F.: Online fault diagnosis for power system based on information theory. Proceedings of the Csee, 23(7), 5-11(2003).
76. Wen, F., Qian, Y., Han, Z., Tian, L., Shi, J., Zhang, H.: A tabu search based approach to fault section estimation and state identification of unobserved protective relays in power systems using information from protective relays and circuit breakers. Proceedings of the CSEE, 13(5), 1000-6753(1998).
77. Wang, T., Zeng, S., Zhang, G., Pérez-Jiménez, M., Wang, J.: Fault section estimation of power systems with optimization spiking neural P systems. Romanian Journal of Information Science and Technology, 9(6), 786-799(2014).

Multi-behaviors coordination controller design with enzymatic numerical P systems for autonomous mobile robots in unknown environments

Xueyuan Wang[1,2], Gexiang Zhang[2] ⋆, Haina Rong[2], Prithwineel Paul[2], and Hua Zhang[1]

[1] School of Information Engineering, Southwest University of Science and Technology, MianYang 621010, P.R.China
121053406@qq.com
[2] School of Electrical Engeneering, Southwest Jiaotong University, Chengdu, 610031, China
zhgxdylan@126.com

Abstract. In this paper we propose a novel multi-behaviors co-ordination controller model using enzymatic numerical P systems for autonomous mobile robots navigation in unknown environments. An environment classifier is constructed to identify different environment patterns in the maze-like environment and the multi-behavior co-ordination controller is constructed to coordinate the behaviors of the robots in different environments. A 11 sensory prototype of local environments is presented to design the environment classifier, which needs to memorize only the rough information, for solving the problems of poor obstacle clearance and sensor noise. A switching control strategy and multi-behaviors coordinator are developed without detailed environmental knowledge and heavy computation burden, for avoiding the local minimum traps or oscillation problems and adapt to the unknown environments. Also, a serial behaviors control law is constructed on the basis of Lyapunov stability theory aiming at the specialized environment, for realizing stable navigation and avoiding actuator saturation. Moreover, both environment classifier and multi-behavior coordination controller are amenable to addition of new environment models or new behaviors due to the modularity of the hierarchical architecture of P systems. Extensive experiments conducted on the simulated wheeled mobile robots show the effectiveness of this approach.

Keywords: Membrane computing, reactive navigation, autonomous mobile robot, behaviors coordination.

1 Introduction

P systems (PS) are bio-inspired parallel distributed computing models. Since the introduction in 1998 by Păun, these models have gained significant attention from the formal language theorists, mathematicians, biologists and computer scientists. Many variants of P systems have been introduced, inspired from the functioning and intercellular communication of cells, neurons etc [1, 80, 83]. The computation power and

⋆ Corresponding author.

complexity aspects of these models have been studied extensively [1, 79, 83]. Moreover, these variants have been used to solve the computationally hard problems, i.e., the **NP**-complete, **PSPACE**-complete problems in polynomial time or even on linear time [73–77]. In recent years the use of the membrane computing to solve many real-life problems has also gained interest, especially to solve engineering problems. Some variants such as spiking neural P systems have been used for fault diagnosis of the power systems [25], image processing [26], approximation modelling [28]. The variants numerical P systems, tissue P systems have been used in robotics, image segmentation [81, 82] respectively. Also, the complementary methods of meta heuristics and different types of PS have been developed to solve various optimization problems, such as parameter optimization problems in manufacturing [24], optimal approximation models [28], etc. P systems are also used for some real-life applications such as modeling problem in ecological systems with probabilistic P systems [27].

Numerical P Systems (NPS) were introduced by Păun in [33] with an inspiration to model economic and business processes. Many variants of NPS are discussed in [7–10]. Enzymatic Numerical P Systems (ENPS) are a variant of NPS introduced in [30]. Both NPS and ENPS have been proved to be Turing universal [31]. Moreover, mobile robot's behavior controllers in the framework of NPS and ENPS have been developed and shown that membrane controllers have an excellent performance [29, 30, 32, 34]. But most of the controllers considered in these works are relatively simple or single tasks, such as obstacle avoidance [29], location [30], trajectory tracking [34], etc. Because of the inherent parallel and distributed nature along with powerful numerical computation power, NPS and ENPS are a good choice for modeling parallel and distributed control systems, especially for modular and complex tasks of autonomous mobile robotics (AMR).

One of the most fundamental problem in robotics is obtaining a path for the robot from the starting point to the goal. When a robot moves in the complex and unknown environment, it faces many situations. To reach the goal by overcoming these situations is the main challenge. These problems can be solved by recognizing the environment patterns, planning a path and executing the navigation safely and efficiently [46–69]. The concept of membrane controllers based on numerical P systems was introduced in [29] and has been further discussed in [34] to design the control of autonomous mobile robots using enzymatic numerical P systems. Still now the controllers based on P systems, considered in the research of robotics are simple in nature. In this work we investigate the navigation of the robots with more complex controllers based on ENPS which can identify the 11 environment prototypes and coordinate the behaviors of the robots using multi-behavior coordination controller. Although the problems in robotics have been solved by using these computational paradigms, the domain of application of membrane computing, specifically numerical P systems and enzymatic numerical P systems can be extended further by investigating the navigation of the robots with more complex controllers.

In this study, an environment classifier and a novel multi-behaviors control approach based on ENPS are proposed to enhance the reactive navigation performance of the AMR. The novelty of this approach is mainly in three aspects: (1) 11 prototypes of comprehensive topological maps describing the local environments are con-

sidered together to design the classifier for environment identification module; (2) A multi-behavior coordination membrane controller (MBCMC) is presented for behavior coordinator module; (3) A serial control algorithm is developed to guide AMR to avoid obstacle, tend to target and follow a wall, etc. In order to reduce the error impact of sensor noise and poor obstacle clearance, the membrane classifier is designed based on the "binarized rough model" to produce the precisely desired environment pattern, which is used as the input of the behavior coordinator module. Behavior coordinator uses an enzymatic numerical P system to integrate specialize behaviors by a well-thought out local path planning strategy, without large memory size and heavy computation burden. The specialized behavior control algorithm is designed based on the Lyapunov stability theory to produce the precisely desired velocity. Furthermore, the effectiveness of the introduced control approach is verified by applying into the simulated AMR.

The paper is organized as follows. Section 2 describes MBDSP. In Section 3, we depict the proposed behavior based membrane controller in detail and discuss hybrid control architecture for solving MBDSP of AMR reactive navigation. Section 4 presents experimental results. Conclusions are drawn in Section 5.

2 Multi-Behaviors Dynamic Selection Problems and ENPS

The autonomous robots are capable of self-judgment and independent navigation in an unknown environment. We describe the AMR mechanical system, and MBDSP in the following section.

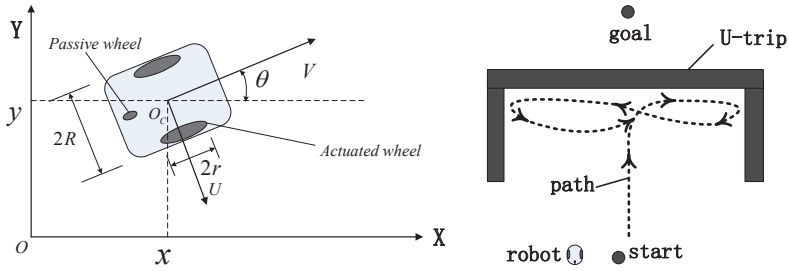
2.1 AMR description and Problem Statement

In this study, the AMR mechanical system schematic is shown in Fig.1(a) which consists of two actuated wheels and a back unpowered universal wheel. The passive wheel does not affect the degree of freedom of the kinematic model, and can work with the nonholonomic constraints as follows:

$$\dot{y} \cdot \cos\theta + \dot{x} \cdot \sin\theta = 0 \tag{1}$$

The posture of AMR in global coordinates frame XOY is represented by using the Cartesian coordinate vectors with three degrees of freedom $p = \{x, y, \theta\}^T$. The positive direction of θ is anti-clockwise, which is used to guide the angle of a robot. The motion posture of AMR is determined by linear velocity v and angular velocity ω, which is denoted by vector $V = (v, \omega)^T$. Note that, the two wheels are driven by independent torques from two DC motors, where the radius of two wheels are represented by r, while the distance between two driving wheels is denoted by $2R$. It is assumed that the AMR mass center is located at O_c and mounted with non-deformable wheels. The kinematic model for AMR can be represented as (2) in [43], where v_r and v_l are the linear velocities of the left and right wheel, respectively.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} (v_r + v_l) * \cos\theta / 2 \\ (v_r + v_l) * \sin\theta / 2 \\ (v_r - v_l) / 2R \end{bmatrix} \tag{2}$$



(a) mobile robot schematic

(b) AMR getting trapped in a U-shape trip

Fig. 1. AMR description and getting trapped in U-shape trip

Now we discuss what is multi-behaviors dynamic selection problems (MBDSP). Let us imagine that a robot wants to reach some destination in an unknown environment. At first, the robot follows the planned path and will avoid if some obstacle is blocking the path. If the obstacle is very large, it may decide to walk along the periphery of the obstacle. The robot should stop immediately if some motion obstacle suddenly appears in front, etc. So there can be many unknown situations in front of the robot and it must have the ability to handle the movements safely and effectively. Hence a group of distinct behavior modes is supposed to help the robot to co-ordinate at each time instant. This is the so-called *Action Selection Problem* in robotic reactive navigation [44], which we have referred to as the MBDSP. The reactive navigation is one of the most challenging problems in AMR. The behavior-based systems are proved to be very responsive to an unknown environment, and the performance of reactive navigation greatly relies on its behavior selection mechanism module. Moreover, there are several aspects about MBDSP which should be paid more attention to

(1) **Behavior control law model:** current controllers usually implement processing of sense-plan-action separately, and do not consider the unity kinematic control law model of different behaviors deliberately, while robots need to wander free not only in maze but also in outdoor and indoor unknown environment;

(2) **Control architecture mode:** current action based architecture is not clear about designing an architecture which allows the dynamic switching among different types of behavior (such as reactive or reflective behavior) selection strategies;

(3) **Multi-behaviors coordination mode:** AMR can very easily fall into the local minima trap when reactive navigation has no prior knowledge of the complex environment. It is also likely to be caused by the first two factors. But an excellent coordinator can prevent from these faults. Hence, the dynamic switching strategy, subdivision of different types of behaviors and designing of the corresponding control law are introduced. Furthermore, the behaviors that are usually needed for AMR to wander free in an unknown environment (including outdoor, indoor and maze) are defined clearly in the following:

* Environment classification;

* Path tracking;
* Goal reaching;
* Obstacle avoidance;
* Wall following;
* Corridor walking;
* Emergency U-turn;
* Self rotation;
* ...;

2.2 Related Work

The study on the control for mobile robots was introduced by Brooks [46] and Lumelsky [47]. Following these methods, more and more advanced modern control approaches have been proposed and successfully applied to AMR in industrial contexts, see also [13–22, 34, 36, 37, 39–41]. These control approaches can be classified into the following categories according to different control theories: artificial potential field (APF) [48], vector field histogram methods [49], virtual target approaches [50], dynamic window approaches [51], fuzzy logic control (FLC) [52], neural network methods [53], bug methods [54], and many others. Among the various local or reactive navigation methods, some problems continue to bother them, such as local minimum trap, complex scenarios, lack of prior knowledge, etc. The well-known traditional APF [48] and its extended methods [52, 55, 56] are suitable for underlying on-line control in dynamic environments and low processing needs, but it has a problem of local minima [55], which needs to resort global knowledge of the environment at a higher layer. The Bug family methods [54, 57–59] are inspired by bug's behavior on crawling along the obstacle. These approaches are well known for local navigation with minimum sensor, and also for shorter timing, shorter path planning, a simpler algorithm and better performance. But the performance of these approaches depends on the shape of the obstacles in the environment and need some global visual information. Moreover, the Bug family algorithm usually ignores robotic's practical setting (e.g., for kinematic or dynamic constraints). FLC is indeed one of the most fundamental methods and widely used to coordinate numerous basic tasks involved in path planning of behavior-based robots. Many FLC approaches with other complementary techniques were developed to solve some of mobile robot navigation problems in obstacle avoidance [52, 61], path tracking [60] and behavior coordination [62, 63]. Although FLC rules offer possible implementations of human knowledge and experience which do not require a precise analytical model of the environment, they cannot obtain the optimal solution and mostly fail while dealing with trap situations and complex scenarios [66].

AMR behavior based reactive navigation usually involves many aspects such as environment identification, control structure, dynamic behavior selection strategies, robot physical setting, etc. The study of MBDSP [42, 43, 45, 64–66] usually emphasizes on one or two aspects and the other properties are simplified or ignored. In this study, most of them are carefully considered to obtain the desired behaviors of the corresponding environment models and reduce the influence of the local minimum traps of complex unknown environments. Unlike APF and bug family methods [55, 56, 58, 59], which do not care about the robotic physical characteristics completely. But in this paper, the

kinematic behaviors are considered to be designed by Lyapunov theory in accordance to robotic characteristics which are suitable for indoor and outdoor environments. Design of the specialized behaviors control law is beneficial for multi behaviors co-ordination. This study also uses an enzymatic numerical P system to improve the parallel computation performance of the environment classifier and behavior coordinator. In order to coordinate these behavior controller for different tasks, a novel membrane hybrid control architecture is proposed to manage those reactive and reflective controllers. Also, the computations can be performed in parallel. Thus, the computations are flexible and are in accordance with reactive navigation.

2.3 Enzymatic numerical P systems:

ENPS are naturally distributed and parallel computing models, in which numerical variables store information. Also a set of evolving rules in each membrane region can iterate simultaneously according to the activation conditions, and transmit information between the nodes (membranes). A standard ENPS is as follows [31]:

$$\Pi = (m, H, \mu, (Var_1, E_1, Pr_1, Var_1(0)), \quad ..., (Var_m, E_m, Pr_m, Var_m(0))) \tag{3}$$

where

1. m is the number of membranes, $m \geq 1$;
2. H is an alphabet that contains m symbols;
3. μ is a membrane structure;
4. Var_i is the set of variables from membrane i, and $Var_i(0)$ is the initial values for these variables;
5. E_i is a set of enzyme variables from membrane i, i.e., $E_i \subset Var_i$;
6. Pr_i is the set of programs (rules) in membrane i, composed of a production function and a repartition protocol.

ENPS have flexible computing feature. Because of the hierarchical membrane structure with multiple rules in one region characteristics, enzyme variables can be used for conditional transmembrane transport and decide on the rules of evolution direction. The active rules are performed simultaneously inside their membranes, but unnecessary rules are not carried out and the results are distributed in globally uniform way. The computing power of the ENPS, and efficiency of the membrane structure representation for designing robotic behaviors have been investigated in [31] and [29], respectively.

3 Design of Environment Classifier and Behavior Coordination Controller

In order to realize the AMR reactive navigation in unknown and complex environments, at first we need a reasonable control structure to organize environment classifier and behavior coordination controller modules. Fig.2 shows the hybrid control architecture proposed in this paper based on P system. A three-layer (organization management, deliberative, reactive) architecture is adopted and the structure of the reactive layer is

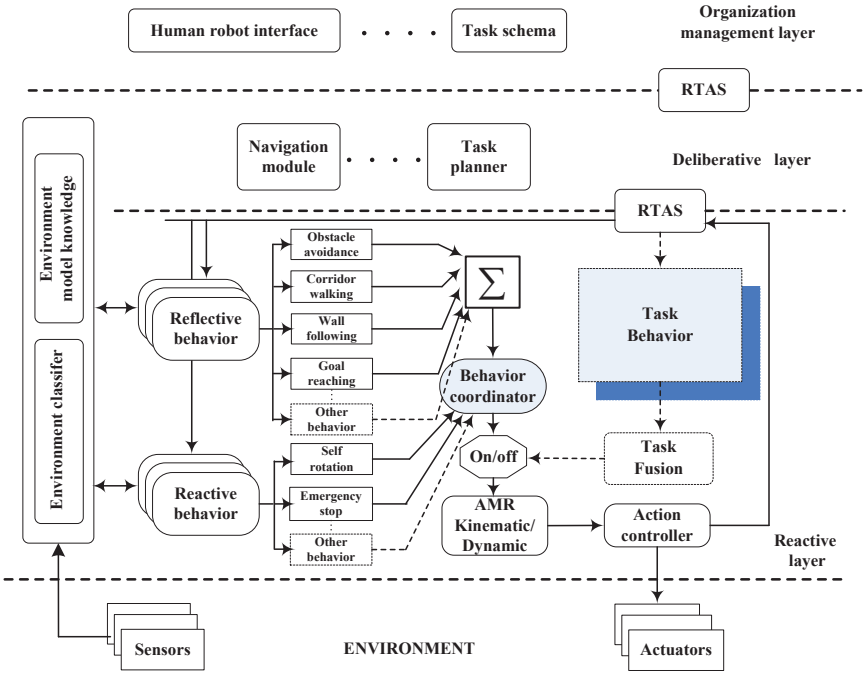


Fig. 2. membrane hybrid control architecture

discussed. The focus of this paper is mainly on the execution level (Reactive layer in Fig.2) of the proposed hybrid control architecture. Exact geometrical modeling is both time-consuming and unnecessary. But environment model knowledge module only has the "binarized rough model", and the membrane classifier module is designed based on the ENPS to produce the precisely desired environment pattern as the input of the behavior coordination module. In Behavior Coordination module, we give motivation for a mechanism for dynamic behavior selection which can switch between reflective and reactive behaviors (see in Fig.2). From the behavioral consciousness of the human brain we know that some basic behaviors are conditional reflective ones and can be executed without going through the brain. But some behavior's action is well thought-out. So, the reflective behaviors in this paper (such as obstacle avoidance, wall following, corridor walking. etc) have been carefully designed by the similar control law and "virtual target", and the purpose is to conveniently coordinate these co-operative behaviors. But those behaviors (such as emergency stop, U-turn, self-rotation, etc) need not precisely control law design, which are more likely a conditioned reflex action and is classified as reactive behavior and can be executed individually. Unlike reflective behaviors which are usually active as cooperative method, the reactive behaviors are usually active as competitive method. Once a status of the robot is selected by using the outputs of environment classifier, the behavior coordination controller can produce an appro-

priate behavior according to the relationship among robot, goal and environment. This behavior-based pattern is similar to human information processing, which has a low level of conditional reflective behavior and is executed according to the behavior memory or pre-taught action. Also, it can execute a high level complex behavior according to the information from the sensory organs.

3.1 Design of Environment Classifier

In order to respond according to the appropriate behavior, AMR should know the relationship between its current status and the local environment at first. The output of the environment prototype will work as the features of the essential environment for navigation, and need not store or deal with unnecessary details.

Local environment prototype: Based on our understanding of the outdoor or indoor navigation, there are ten cases for a robot, such as: following a left-side wall, wandering in open area, crossing a corridor or meeting a right-side obstacle, etc. At the first row of Fig.3 five following cases have been shown: left wall (LW), right wall (RW), hallway wall (HW), left corner wall (LC) and right corner wall (RC). The five cases of meeting an obstacle are defined at the second row of Fig.3, i.e., front wall (FW), left side (LS), right side (RS), two side (TS) and dead end (DE).

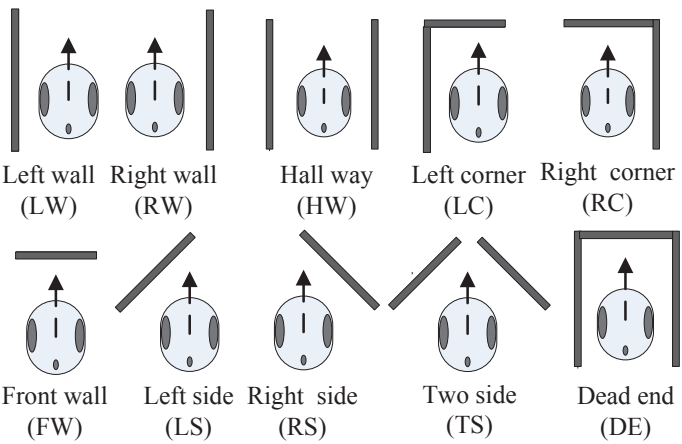


Fig. 3. Ten prototypes of local environment robots may meet

Before classification of the various local environments by sensor, the robot's sensor feature must be defined. In order to reduce the cost of a sensor device, e-puck has only eight 8 Infra-red(IR) distance sensor around the body in Fig.4(a). The Fig.4(b) shows the sensors $IR_{1...8}$ layout and the probing direction from the top of the robot. The values from the 8 IR are grouped into $(G_0, G_1, \ldots, G_7)$ as they meet some obstacle

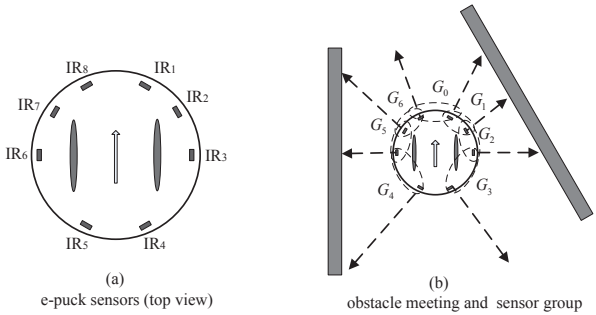


Fig. 4. IR sensors placement for the e-puck

or follow some wall. For instance in Fig.4(b), the values for the groups (G_4, G_5) and (G_1, G_2) will be bigger than the other groups when they meet the left wall and right side obstacle conditions (bigger value means smaller distance to obstacle), respectively. Fig.5 shows the 11 sensory patterns registered for the entire prototype environment which correspond to the 10 cases of maximum possibility according to the assumptions in Fig.3 and *NO* represents there is no obstacle in the environment.

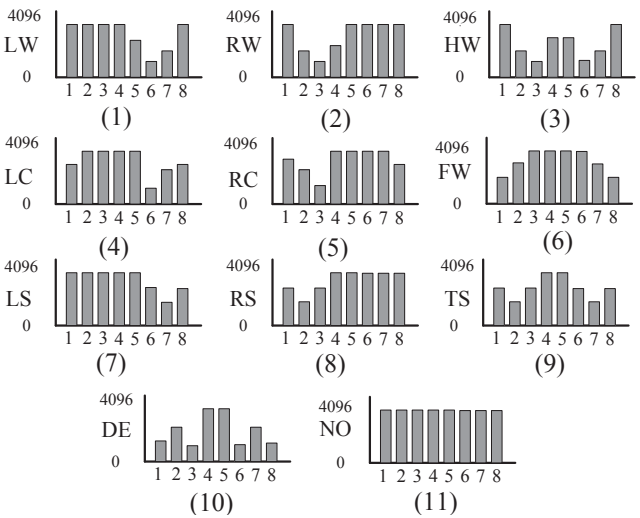


Fig. 5. Sensory patterns for 11 cases

Environment classifier design based on ENPS: In this paper, we propose a local environment classifier based on ENPS to quickly identify the sensory patterns when AMR is surrounded by obstacles. Fast and accurate environment classification is beneficial for the response to the appropriate behavior.

As shown in Fig.6, the environment classifier is designed by using a membrane system with a hierarchical membrane structure containing four membranes. The inner membrane *Compute Environment Model* is used to match the 11 case environment model. According to the sensor data, it has 11 variables, where $s_j\left[\left(sensor_j - p_j^i\right)^2\right]$, $(j = 1 \cdots 8)$ represent the 8 infrared sensor match errors. The p_j^i $(i = 1 \cdots 11)$ represent the 11 cases of environment patterns in Fig.5. The enzyme $E_c[v_{in}]$ has the threshold input value v_{in} as the initial value, and it is used to decide whether the rules $\mathrm{Pr}_{i\_j}, sensor_j$ should be executed according to the values of the variable of $s_{1\cdots8}$. Rule $\mathrm{Pr}_{i\_j}, sensor_j$ is executed when $sensor_j$ is matched with i-th environment pattern $patt_{i\_j}$ successfully and the variables $sum_i[0]$ and $Sum_{all}[0]$ are assigned with value 1 simultaneously. Again, sum_i is used to store the number of successful match of the i_{th} pattern $(i = 1 \cdots 11)$, where larger value represents higher match degree. Then, the numbers are sorted from big to small. The Variable Sum_{all} is proposed to store the total number of successful matches in 11 sensory patterns, which is further used to understand the accelerated sorting instead of traditional sorting method (such as Bubble Sorting, Hill sorting, etc).

The inner membrane *Find Out Several Possible Pattern* is designed to find out several more likely patterns with nine variables, where $S_{aver}[0]$ is the set of average time of total successful match through rule $\mathrm{Pr}_1, pattern_i$. The variable $pat_i[0]$ represents the distance difference between $pat_i[0]$ and $S_{aver}[0]$ though the rule $\mathrm{Pr}_2, pattern_i$. The Enzyme $E_{aver}[0]$ is combined with pat_i in $\mathrm{Pr}_3, pattern_i$ to verify whether this rule is applicable or not. The execution of this rule means, this sensory pattern is a matching environment prototype and the next rules are applicable. The Enzyme $E_{max}[0]$ is set to 9 in $\mathrm{Pr}_3, pattern_i$. Since the pattern variable sum_i must be less than 9, the rule $\mathrm{Pr}_4, pattern_i$ can be applied, and the enzyme $E_{pat\_i}[0]$, $E_{patt\_i}[0]$ are set to pattern value sum_i. The pattern sum variable $M_{sum}[0]$ also accumulate one copy of sum_i. Then the rule $\mathrm{Pr}_5, pattern_i$ is executed and the initial value 1 of variable $num_i[1]$ accumulate to the sum variable $Num_{sum}[0]$ which works as a counter.

The innermost membrane *Find Out Optimal Model* has two variables. The average variable $S_{pat\_i}[0]$ is assigned to the number of the group pattern whose values are bigger than S_{aver} in membrane *Find Out Several Possible Pattern*. So, S_{pat_i} must be larger than S_{aver}, and it can decide whether rule Pr_2 can be activated while combined with the enzyme E_{patt_i}. It can also find out the optimal pattern and the output of the most possible result in the i-th pattern is stored into No. Note that, the enzyme EH in skin membrane *Output Environment Model No* must be assigned to double value of i_{th}. For instance, if the most possible pattern happens at $i = 1$ and EH only get one part value of i_{th}, then $\mathrm{Pr}_2, main$ in skin membrane cannot be activated because of the initial value of variable C_T being 1, and the computing cannot be finished. It is used to ensure that the rule $\mathrm{Pr}_2, main$ in the skin membrane must be executed and the computing is terminated. Meanwhile, the variable Out_{no} in the skin membrane collects the output result of the computation.

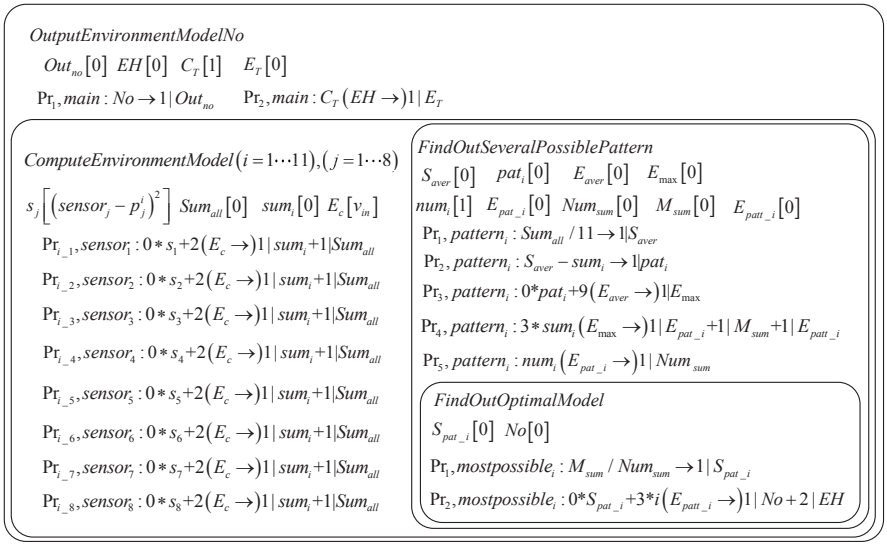


Fig. 6. Membrane classifier for 11 environment patterns

3.2 Multi Behavior Design

In order to adapt to the local environment, AMR reflective behavior and reactive behavior are properly designed according to the physical characteristics of the robot.

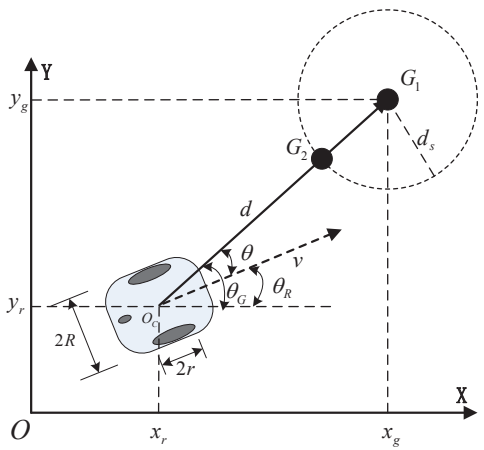


Fig. 7. Defining Path tracking and Goal reaching

Goal reaching: Goal reaching is a behavior that orders the robot to move from the current position to a destination by receiving the desired goal position from the top of the deliberative layer. The description for goal reaching is shown in Fig.7. The current position of AMR with respect to the goal position is expressed in form of the polar coordinates where d represents the distance between goal point G_1 and AMR current point O_c. Again, d_s specifies the safety distance of goal reaching, and θ is the angle error between the current robot heading vector θ_R and goal vector θ_G, $\theta = \theta_R - \theta_G$. Let the robot safety distance and AMR speed limited are as depicted in Fig.7, then the kinematic equations of AMR can be described by the following formulas

$$\begin{aligned} \dot{d} &= -v * \cos\theta \\ \dot{\theta} &= -\omega + \tfrac{v * \sin\theta}{d} \end{aligned} \tag{4}$$

where v and ω represent the robot's linear and angular velocities, respectively. First, select the Lyapunov candidate function

$$V = \theta^2/2 + \int_0^d S(\tau)\tau d\tau \tag{5}$$

which is a positive definite function and the time derivative of Formula (5) is

$$\dot{V} = \theta * \dot{\theta} + S(d) * d * \dot{d} \tag{6}$$

From formula (4), one gets

$$\dot{V} = \theta * \left(-\omega + \tfrac{v * \sin\theta}{d}\right) + S(d) * d * (-v * \cos\theta) = \dot{V}_1 + \dot{V}_2 \tag{7}$$

The control law of linear velocity is

$$v = v_{\max} * S(d) * \cos\theta \tag{8}$$

where $v_{\max}$ is the maximum of linear speed and S function is defined below

$$S(d) = \begin{cases} 1, & d > d_s \\ 1 - \left(\frac{d_s - d}{d_s}\right)^2, & 0 < d \leq d_s \end{cases} \tag{9}$$

The variable d_s in (9) decides the deceleration distance of AMR while reaching the goal. When the robot is far from the goal $(i.e., d > d_s)$, AMR approaches the target as fast as possible, and begins to slow down while reaching the desired target. Substituting (8) into the second partial $\dot{V}_2$ of (7), since $v_{\max} > 0$, $d > 0$, one gets the semidefinite negative function

$$\dot{V}_2 = -v_{\max} * d * S^2(d) * \cos^2\theta \leq 0 \tag{10}$$

Then, Substituting (8) into the first partial of (7), $\dot{V}_1$ rewriting as

$$\dot{V}_1 = \theta * \left(-\omega + \tfrac{v_{\max} * S(d) * \cos\theta * \sin\theta}{d}\right) \tag{11}$$

So, the control law of angular velocity is proposed as

$$\omega = k * \theta + \frac{v_{\max} * S(d) * \cos\theta * \sin\theta}{d} \tag{12}$$

where k is a proportional constant and $k > 0$. Substituting (12) into (11) which results in another semidefinite negative function $\dot{V}_1 = -k * \theta^2 \leq 0$. Hence, one can conclude that the first derivative of the Lyapunov function (13) is the semidefinite negative function, $d = 0$ and $\theta = 0$, which results in $\dot{V} = 0$, i.e.,

$$\dot{V} = \dot{V}_1 + \dot{V}_2 \leq 0 \tag{13}$$

The proposed controller guarantees that $v \leq v_{\max}$ for all $t \geq 0$, and drives the states $d(t)$ and $\theta(t)$ asymptotically to zero. In addition, the goal reaching strategy is based on the Lyapunov function which utilizes the target distance information, has good portability. Moreover, it is inclusive and is applied to design control laws of obstacle avoidance, wall following, corridor walking with a unified "virtual target". In addition, the control laws take into account the safety distance close to the target and the maximum speed of AMR, for which it has better performance of efficiency and safety.

Obstacle avoidance: This controller is responsible for avoiding the obstacles that may appear randomly when AMR is moving towards the target or following a wall. It is a reflective action and is designed by the goal reaching method described above. When an obstacle is detected, we suppose that a dynamic goal will appear ahead of the robot motion direction to lead it to walk around obstacle smoothly. In Fig.8, an obstacle is around the robot and the environment classifier accurately judge it to locate it on the right side of the robot. Thus, one should set a new "virtual target" on the left side of the robot to make ARM turning left in order to prevent the collision. First, it must select the appropriate sensor IR_i (since several sensors detect the obstacle simultaneously, we must select the suitable ones to define the reference direction of "virtual target"). Since the obstacle is located at the right side of the robot, first find the non-zero value sensors (represent obstacle detection) from the front of left side layout sensor IR_6 to the front of right side ones IR_3 ($IR_6 \rightarrow IR_7 \rightarrow IR_8 \rightarrow IR_1 \rightarrow IR_2 \rightarrow IR_3$). If the obstacle is located at left side of the robot, then find the first non zero sensors from IR_3 to IR_6 ($IR_3 \rightarrow IR_2 \rightarrow IR_1 \rightarrow IR_8 \rightarrow IR_7 \rightarrow IR_6$).

In this case, IR_1 is found as the candidate sensor, where d_{max} is the maximum measurable distance of sensor, d_{safety} is the safety turning distance of the robot, d_{IR_1} is the distance reading of IR_1. The virtual pressure radius is denoted by R_v and $R_v = d_{\max} - d_{IR\_1}$, d_{os} is the distance between O_o and O_s, where O_o is the contact point from the ray of sensor IR_1 to the surface of the obstacle. In fact, $d_{os} = d_{IR\_1} - d_{safety}$. So, the virtual turn angle of robot θ_T is

$$\theta_T = \arctan\left(\frac{R_v}{d_{os}}\right) = \arctan\left(\frac{d_{\max} - d_{IR\_1}}{d_{IR\_1} - d_{safety}}\right) \tag{14}$$

The direction of the new goal O_{new} is parallel to the line segment $O_r O_s$. Thus, the desired new goal position can be given by

$$O_{new} = \begin{bmatrix} x_g \\ y_g \end{bmatrix} = \begin{bmatrix} x_r + k * d_w * \cos\theta_{New} \\ y_r + k * d_w * \sin\theta_{New} \end{bmatrix} \tag{15}$$

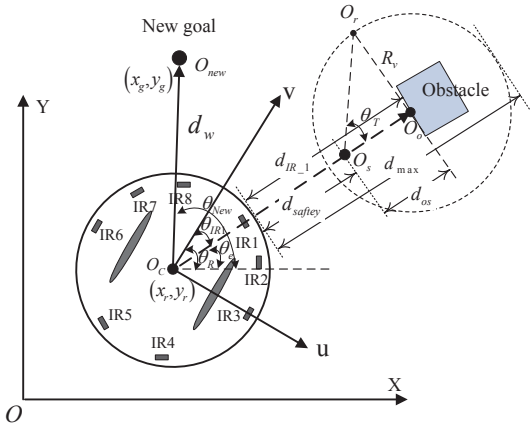


Fig. 8. State describe for obstacle avoidance

where d_w is the desired distance of the new goal ("virtual target"), k is the proportional coefficient and the variable θ_{New} is the orientation of new goal O_{new} attached on the robot platform centroid O_c measured from the horizontal axis (OX). Moreover, θ_{New} is reduced to

$$\begin{aligned} \theta_{New} &= \theta_e + \theta_T \\ \theta_e &= \theta_R - \theta_{IR1} \end{aligned} \tag{16}$$

where θ_R is the orientation of the robot measured from the horizontal axis (OX), and θ_{IR1} is the mounted angle of the sensor IR_1 attached to the robot local coordinate frame (u, O_c, v). Hence, θ_e is equivalent to the angle of the sensor vector (IR_1) relative to the horizontal axis (OX).

Formula (15) defines a local new goal ahead of the robot motion direction, precisely it converts the repulsive force field of the obstacle into a gravitational field of "virtual target". Obviously, as the robot-obstacle distance starts decreasing, θ_T will become bigger and the new goal of the robot is shifted to the opposite direction rapidly. Then, a deviation control signal of angular velocity ω_{oa} and line velocity v_{oa} are generated by goal reaching methods

$$\begin{aligned} v_{oa} &= k_r * d_w * \cos\theta_{oa} \\ \omega_{oa} &= k_{oa} * \theta_{oa} + \kappa_{oa} * \sin\theta_{oa} * \cos\theta_{oa} \\ \theta_{oa} &= \theta_{New} - \theta_R = \theta_T - \theta_{IR1} \end{aligned} \tag{17}$$

where k_r, k_{oa} and κ_{oa} are the proportional coefficients. The reaction capability of the controller is regulated by a proper pre-definition of those constants. In this way, the proposed obstacle avoidance controller can merge with the goal of reaching navigation capability.

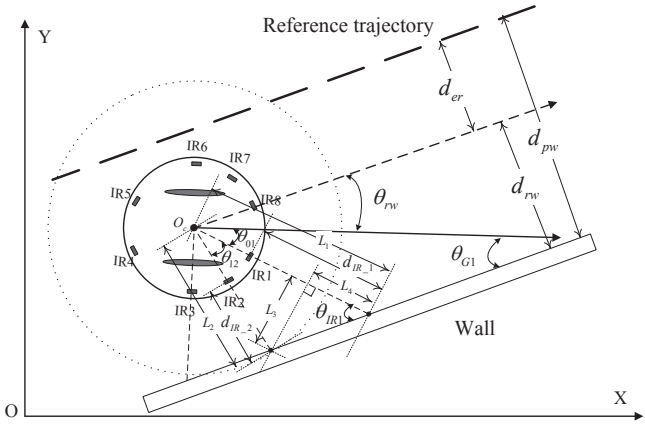


Fig. 9. State describe for wall following

Wall following: Wall following is the robot's ability to follow a wall. These abilities are intended to complement other reflective behaviors in narrow spaces and corridors. The basic objective of the wall following is the generation of a reference trajectory, parallel to a wall (even to a bigger obstacle surrounding). If the distance and angular information between the robot and wall are considered, it can use the goal reaching method to design the wall following controller. In [68, 69], the wall of the surface followed by the robot is supposed to be parallel or perpendicular and the coordinates (X,O,Y) are used in order to simplify the design. Consider a more practical situation as in the wall in Fig.9 which is an arbitrary arrangement. The angle θ_{rw} between the direction of robot motion and the parallel line of the wall can be computed by the angles between the infrared sensors ray and wall surface. In Fig.9, environment classifiers judge the wall located at the right side of the robot. So, several right side group of sensors (G_1,G_2,G_3) in Fig.4(b) are used to compute θ_{rw}. In this case, G_1 is used to get θ_{G1}. For instance, G_1 has two sensors IR_1 and IR_2. The θ_{IR1} is the angle between the vector of IR_1 and wall surface. Moreover, L_1 and L_2 are the line length from the centroid O_c to the wall surface through sensors IR_1 and IR_2, respectively. Again, $L_1 = d_{IR\_1} + R$, $L_2 = d_{IR\_2} + R$, where d_{IR_1} and d_{IR_2} are the reading data of IR_1 and IR_2 and R is the robot's radius. Hence, θ_{IR1} can be given by

$$\begin{aligned} \theta_{IR1} &= \arctan\left(\frac{L_3}{L_4}\right) \\ L_3 &= L_2 * \sin\theta_{12} \\ L_4 &= L_1 - L_2 * \cos\theta_{12} \end{aligned} \tag{18}$$

where, θ_{12} is the angle between sensor IR_1 and IR_2, θ_{01} is the angle between the sensor IR_1 and the robot motion direction. So, θ_{G1} is represented by

$$\theta_{G1} = \theta_{IR1} - \theta_{01} \tag{19}$$

The error angle between the robot motion direction and the wall surface is θ_{rw}, and hence we can get

$$\theta_{rw} = \frac{\sum_{i=1}^{n} \theta_{Gi}}{n} \tag{20}$$

where n is the number of effective group sensors, and by effective group it means that both the sensors can be obtained from the distance information, θ_{Gi}, also can be obtained by Formula (18) and (19).

In this case, if the assumed AMR walk along the wall at uniform speed v_{con}, then the kinematic formula is reduced to

$$\begin{aligned} \dot{d}_{er} &= v * \sin\theta_{rw} \\ \dot{\theta}_{rw} &= \omega \end{aligned} \tag{21}$$

where d_{er} is the error distance between the robot centroid O_c and reference trajectory. If the Lyapunov candidate function is as in the Formula (5), then the control law of the angular velocity ω_{wf} is obtained in the following manner

$$\omega_{wf} = -k_{wf} * \theta_{rw} - \kappa_{wf} * v_{con} * d_{er} * \frac{\sin\theta_{rw}}{\theta_{rw}} \tag{22}$$

where k_{wf} and κ_{wf} are the positive proportional coefficients. Again, $d_{er} = d_{pw} - d_{rw}$, d_{pw} is the distance of the given reference trajectory to wall, d_{rw} is the distance from the robot centroid O_c to the right side wall and can be represent by

$$d_{rw} = \frac{\sum_{i=1}^{n} d_{Gi}}{n} + R \tag{23}$$

$$d_{Gi} = \frac{d_{IR\_i1} + d_{IR\_i2}}{2} \tag{24}$$

where d_{IR_i1} and d_{IR_i1} are the distance datas of two sensors in right side effective group G_i, respectively.

Corridor walking: The control law can be extended to solve the corridor walking as considered in Fig.3. For this case (i.e., HW), both the left and right side group sensors get the distance data. If $\sum_{i=1}^{4} d_{IR\_i} > \sum_{i=5}^{8} d_{IR\_i}$, it means that the robot is closer to the right side wall and hence it should follow the right side wall. Otherwise, follow the left side wall. In order to let the robot running in the middle of the hallway as far as possible, d_{pw} is reset to

$$d_{pw} = \frac{d_{rw} + d_{lw}}{2} \tag{25}$$

where d_{lw} is the distance from the robot centroid O_c to the left side wall and can be obtained by Formula (23) and (24). The Lyapunov candidate function is also treated as Formula (5) and the d_{er} can be replaced by d_{pw}.

Self rotation: When the robot meets with one of the cases such as corner of the wall, ending of the corridor, local trap points, etc, the robot needs to do a self rotation in clockwise mode ($v_l = v_{pc}, v_r = -v_{pc}$) for (LC) and counterclockwise mode ($v_l = -v_{pc}, v_r = +v_{pc}$) for (RC), where v_l and v_r are the linear velocities of the left and right driving wheels, respectively. The predetermined driving velocity is denoted by v_{pc}.

Emergency U-turn: The emergency U-turn means that the robot should do a U-turn while meeting the environment mode, as in the case of (DE) in Fig.3. It is activated when the distance between the robot and the obstacle becomes smaller than a certain value. Also, after U-turn, the robot continue the navigation module according to the environment classification. Since no well thought-out control law is required for the self-rotation and U-turn, they can be treated as the reactive behaviors.

3.3 Dynamic Multi-Behavior Coordination

In order to explore complex and unknown environments, AMR not only need to feel sensitive, it also must act safely and smoothly. Moreover, AMR can break away from local minima trap and arrive at the goal finally. This section describes how to co-ordinate with these behaviors by dynamic selection mechanism.

Multi-behavior coordination strategy The proposed flow chart of dynamic multi-behavior selection is depicted in Fig.10. In Fig.10, $Flag = 1$ means AMR is moving towards the goal until some "obstacles" are detected, where d_{gr} is the distance between goal and robot. It is defined as $d_{gr} = \sqrt{(x_g - x_r)^2 + (y_g - y_r)^2}$, where ($x_g$,$y_g$) and ($x_r$,$y_r$) represents the coordinate of goal and robot, respectively. It should be getting smaller and smaller while running towards to the goal, but in contrast, if it is becoming bigger, it means that the obstacle avoidance or wall following mode is operated and the robot has moved far away. AMR can determine the accurate status relationship between itself and the obstacle by environment classifier at once. The "obstacles" can be grouped as obstacle case $6,7,8$, wall follow case $1,\ldots,5$ (corridor walking also classified as this case), and the dead end case $9,10$. AMR might fall into the trap while avoiding the obstacle or following the wall. In order to resolve the local minimum problem, AMR must investigate the problems such as positional relationship among goal, obstacle, wall and robot. Also must investigate whether the distance d_{gr} is minimal and what kind of obstacle is around? For instance, if some obstacles or walls are located at the right side of the AMR according to the environment model case, then the goal is located at the left side of the robot, and d_{gr} is the minimal distance. Also, AMR should enter the goal reaching mode. On the other side, if the goal and obstacle are located on the same side of AMR, then even if d_{gr} is minimal, the goal reaching mode cannot be activated. In another example, in order to go out of the maze, if the robot has just passed the wall (obstacle) and entered into the open area, it should go to the judge state and select goal reaching mode directly or self-rotation mode to follow wall again. In this work, an interesting dynamic multi-behavior selection strategy is constructed to

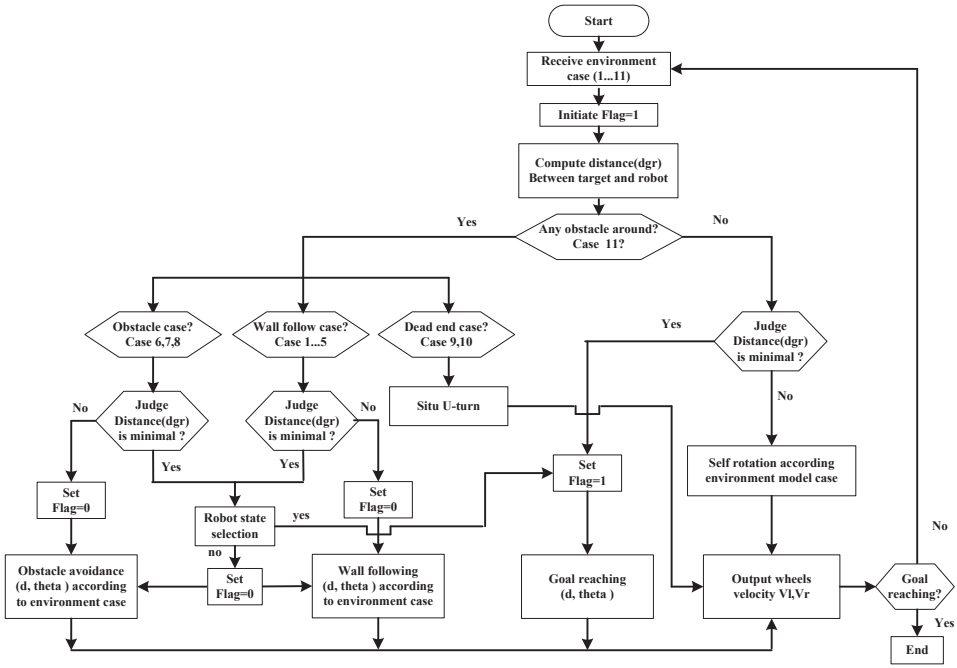


Fig. 10. Flow chart of multi behavior coordination controller

speed up the behavior coordination by parallel processing. Moreover, the corresponding co-ordination controller based on P system is shown in Fig.11.

Multi-behavior coordination membrane controller design: MBCMC is shown in Fig. 11. It is designed by using a P system with a hierarchical membrane structure containing eight membranes. The skin membrane *Main* has 27 variables and $C_i = \left[(input_{no} - i)^2\right]$, $i = 1, 2, \ldots, 11$ are the environment case variables. These variables have the initial value $(input_{no} - i)^2$, $input_{no}$ and it is one of the 11 patterns from environment model membrane classifiers. $A_{gr1}\left[input_{angle}\right]$ and $A_{gr2}\left[-1 * input_{angle}\right]$ are the angle variables and have the input value $\theta = \theta_R - \theta_G$ as initial value, which depicts the positional relationship between robot and goal. $\theta < 0$ means that the goal is located at the left hand of robot and vice versa (Fig.7(b)). Also, $Com_{ob}^{left}[0]$,$Com_{ob}^{right}[0]$ and $Com_{ob}^{front}[0]$ are left, right side and front obstacle avoidance behavior control output variable, respectively. Again, $Com_{wf}^{left}[0]$, $Com_{wf}^{right}[0]$ and $Com_{wf}^{hall}[0]$ are left, right wall following and hall crossing behavior computation output variable, respectively. The variable $Com_{de}[0]$ is the U-turn output variable for dead end case and $Com_{gr}[0]$ is the goal reaching output variable. Moreover, $Com_{no}^{rs}[0]$ is the self-rotation behavior control output variable. All of the output variables have the initial value 0, but when some of the

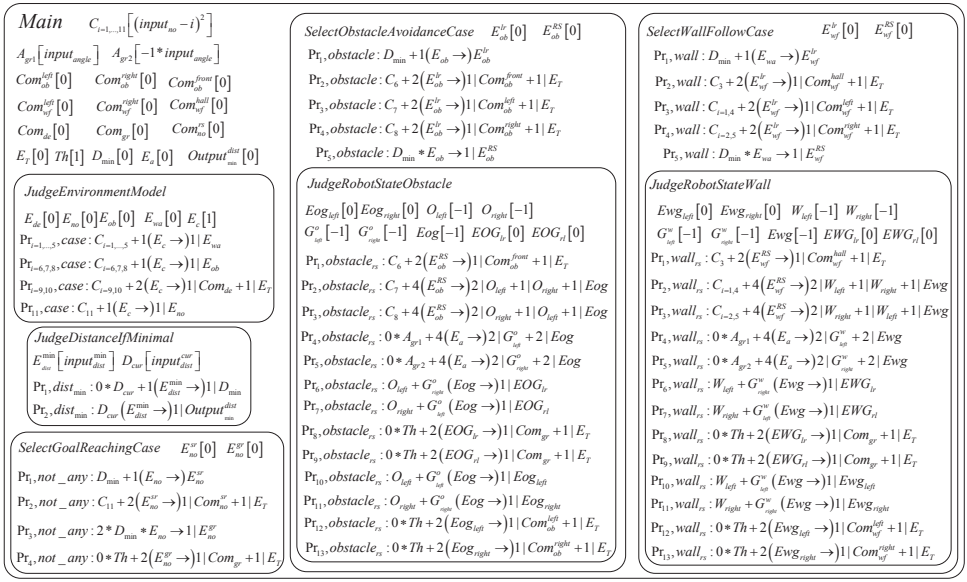


Fig. 11. Dynamic multi-behavior coordination membrane controller

behavior gets triggered, then the corresponding variable value is set to 1. The variable $Th[1]$ is the threshold variable with initial value 1 and $D_{\min}[0]$ is the minimal distance variable with initial value 0. The enzyme variable $E_a[0]$ and $E_T[0]$ have the initial value 0, but when E_T is not equal 0, the controller is terminated. The $Output^{dist}_{\min}[0]$ is the minimal distance output variable which has initial value 0.

The inner membrane *Judge Environment Model* has five enzyme variables, where $E_{no}[0]$, $E_{de}[0]$, $E_{ob}[0]$ and $E_{wa}[0]$ work as trigger enzymes for not any obstacle case, meet dead end case, meet obstacle case and wall case, respectively. Enzyme E_c has initial value 1, and is used to decide whether the 11 rules $\mathrm{Pr}_{i=1,\ldots,11}, case$ should be executed or not according to the environment case variables $C_{i=1,\ldots,11}$. For instance, if $input_{no}=9$, then the initial value of $C_{11}=(9-11)^2=4$ and hence enzyme $E_c<C_{11}$. So, the rule $\mathrm{Pr}_{11}, case$ can not be activated. On the contrary, for rule $\mathrm{Pr}_9, case$, the initial value of $C_9=0$, hence $E_c>C_9$, and rule $\mathrm{Pr}_9, case$ is executed. Moreover, Com_{de} has set value 1, U-turn behavior is selected, E_T is set to 1 and controller ends.

The inner membrane *Judge Distance If Minimal* has two variables. The enzyme $E^{\min}_{dist}$ with the input value $input^{\min}_{dist}$, which is also the minimum distance of all the distances between the robot and goal. The variable D_{cur} has the input value $input^{cur}_{dist}$ as the initial value, which is the current distance between the robot and goal. Both of those variables decide whether the rules $\mathrm{Pr}_1, dist_{\min}$ and $\mathrm{Pr}_2, dist_{\min}$ should be applied or not. If $D_{cur}<E^{\min}_{dist}$, both rules are activated and the minimal distance variable $D_{\min}[0]$ is set to 1. Meanwhile, the minimal current distance variable D_{cur} is collected as the output of the variable $Output^{dist}_{\min}[0]$.

The inner membrane *Select Goal Reaching Case* has two enzyme variables, $E^{sr}_{no}[0]$ and $E^{gr}_{no}[0]$ with initial value 0. This membrane will be activated by enzyme $E_{no} = 1$ as case 11 (no obstacle around the robot). Rule Pr_1, not_any is used to judge the robot. If there is any reverse movement away from the goal, then the value of D_{min} is equal to 0. It means that the robot has just left the obstacle or wall case. The rule Pr_2, not_any is activated when $E^{sr}_{no} = 1$ and Com^{sr}_{no} is set to 1. The robot will implement the rotation mode. It will turn left or right according to the previous behavior case. For instance, the robot will turn left when the previous case is left obstacle avoidance or left side wall following, and vice versa. This mode helps the AMR to find the wall or obstacle surface again while running around the maze or to avoid the U shape obstacle. Rule Pr_3, not_any will let enzyme E^{gr}_{no} be equal to 1 as $D_{min} = 1$ (trend to goal movement). Again, $E^{gr}_{no} = 1$ will activate the rule Pr_4, not_any and the robot will obtain the goal reaching mode. Moreover, E_T is set to 1 and the controller ends.

The inner membrane *Select Obstacle Avoidance Case* has two enzyme variables $E^{lr}_{ob}[0]$ and $E^{RS}_{ob}[0]$ and one sub membrane. This membrane is activated by the enzyme $E_{ob} = 1$ as in case $6, 7, 8$ (front, left or obstacle). Rule $\mathrm{Pr}_1, obstacle$ is also used to judge the robot wether there is any reverse movement away from the goal like rule Pr_1, not_any in membrane *Select Goal Reaching Case*, and rule $\mathrm{Pr}_2, obstacle$, $\mathrm{Pr}_3, obstacle$ or $\mathrm{Pr}_4, obs\ tacle$ is activated according to the values of C_6, C_7 and C_8. For instance, if $C_8 = 0$, then the rule $\mathrm{Pr}_4, obstacle$ is executed, Com^{right}_{ob} is set to 1 and the robot implements right side obstacle avoidance. Moreover, E_T is set to 1 and the controller ends. On the other side, rule $\mathrm{Pr}_5, obstacle$ assigns $E^{RS}_{ob} = 1$ as $D_{min} = 1$ (trend to goal movement), and it activate the rules in sub membrane *Judge Robot State Obstacle*.

Judge Robot State Obstacle has 5 enzyme variables and 4 common variables. Enzymes $Eog_{left}[0]$, $Eog_{right}[0]$, $EOG_{lr}[0]$, $EOG_{lr}[0]$, $EOG_{rl}[0]$ have initial value 0 and $Eog[-1]$ has initial value -1. The common variables $O_{left}[-1]$ and $O_{right}[-1]$ are used to mark the obstacle and locate it at the left or right side of the robot by changing the initial value from -1 to 1. Similarly, $G^{o}_{left}[-1]$, $G^{o}_{right}[-1]$ are used to record the goal and locate it at the left or right side of the robot. The rule $\mathrm{Pr}_1, obstacle_{rs}$ is activated as $C_6 = 0$ (case 6: the obstacle is located at the front of the robot) and both Com^{front}_{ob} and E_T are set to 1. The AMR should compute the obstacle avoidance at once and without any further analysis. Rule $\mathrm{Pr}_2, obstacle_{rs}$ should be activated as case 7, two contribution is assigned to O_{left}, one contribution is assigned to O_{right}, then $O_{left} = -1 + 2 = 1$, $O_{right} = -1 + 1 = 0$ (means obstacle is located at left side of the robot). It is same as rule $\mathrm{Pr}_2, obstacle_{rs}$ to represent the right side of the obstacle. Both rules $\mathrm{Pr}_4, obtacle_{rs}$ and $\mathrm{Pr}_5, obtacle_{rs}$ are used to judge the location of the goal at the left side or right side of the robot according to the variables A_{gr1} and A_{gr2}. Enzyme Eog can obtain contribution from rules $\mathrm{Pr}_{2,\ldots,5}, obtacle_{rs}$. Also if Eog is large enough (2 is obtained in this controller), then it will activate rules $\mathrm{Pr}_6, obtacle_{rs}$ and $\mathrm{Pr}_7, obtacle_{rs}$. Moreover, both the rules are used to judge whether the obstacle and goal are located on both sides of the robot. If rule $\mathrm{Pr}_8, obtacle_{rs}$ or $\mathrm{Pr}_9, obtacle_{rs}$ is activated, and both of Com_{gr} and E_T are set to 1, then the AMR should be able to compute the goal reaching. On the contrary, if the rules $\mathrm{Pr}_{10,\ldots,13}, obtacle_{rs}$ are executed while the obstacle and target are located at the

same side of the robot, Com_{ob}^{left} or Com_{ob}^{right} is set to 1. So, AMR continues to maintain obstacle avoidance despite closer to the goal.

The operating mechanism of inner membrane *Select Wall Follow Case* and its sub membrane *Judge Robot State Wall* for wall following are similar to obstacle avoidance. To restrict the length of the paper, we do not expand the description further.

4 Experimental Results

In this section, the performance of the proposed environment model classifier and dynamic multi-behavior co-ordination controller is verified based on the Matlab simulation. Furthermore, the simulation under Webots (robot simulation software) environment is used to test the performance of mobile robot navigations in different environment models. All the experiments are conducted on the PC with CPU 2.8GHz, 4GB RAM, and the software platform MATLAB7.4 and Windows 7 OS.

4.1 Experiment for Environment Classifier

Since the navigation environment is usually unpredictable, complex and partially unknown, a single environment model membrane classifier can hardly take charge of the whole task. If a single membrane classifier (SMC) is used, it must have a complex structure with many internal parameters to solve the problems of navigation in complex environment. Therefore, a multi-membrane classifier (MMC) (in this paper, two or three) has been employed to identify the environment model with good fault-tolerance capabilities. Since the MMC uses the SMC modules (each covering a specific local environment), it can quickly and easily find good local solutions.

The experiments on e-puck robot with 8 infra-red sensors around have been shown in Fig.4(a). The main parameter of e-puck is presented in Table.1. In order to reduce the impact of sensor noise, the sensor's value is filtered with a given threshold before being sent to the membrane classifier. All values smaller than 70 are ignored. At the same time, in order to simplify the environmental identification model, once the value of some sensor is greater than 70 (close to the obstacle), it activates this channel and is set to 1. Otherwise, is set to 0.

Table 1. Features of the E-puck robot

Features	Technical information
Size, weight	70 mm diameter, 55 mm height, 150 g
Motors, Distance between wheels	2 stepper motors, 55 mm
SpeedRange, Given speed range	[-1024,1024], [-300,300]
IR sensors	8 infra-red sensors (proximity of objects up to 40 mm)
Max IR value, Given threshold	4096, 70

Using aforementioned informations, three kinds of SMC can be constructed in the following manner:

$$C_1 = \begin{bmatrix} 0\,0\,0\,0\,1\,1\,1\,0 \\ 0\,1\,1\,1\,0\,0\,0\,0 \\ 0\,1\,1\,1\,1\,1\,1\,0 \\ 1\,0\,0\,0\,0\,1\,1\,1 \\ 1\,1\,1\,0\,0\,0\,0\,1 \\ 1\,0\,0\,0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0\,1\,1\,1 \\ 1\,1\,1\,0\,0\,0\,0\,0 \\ 1\,1\,0\,0\,0\,0\,1\,1 \\ 1\,1\,1\,0\,0\,1\,1\,1 \\ 0\,0\,0\,0\,0\,0\,0\,0 \end{bmatrix} \quad C_2 = \begin{bmatrix} 0\,0\,0\,0\,0\,1\,1\,0 \\ 0\,1\,1\,0\,0\,0\,0\,0 \\ 0\,1\,1\,0\,0\,1\,1\,0 \\ 1\,0\,0\,0\,0\,1\,0\,1 \\ 1\,0\,1\,0\,0\,0\,0\,1 \\ 1\,0\,0\,0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0\,0\,1\,1 \\ 1\,1\,0\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0\,1\,0 \\ 1\,0\,1\,0\,0\,1\,0\,1 \\ 0\,0\,0\,0\,0\,0\,0\,0 \end{bmatrix} \quad C_3 = \begin{bmatrix} 0\,0\,0\,0\,0\,1\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,1\,0\,0 \\ 1\,0\,0\,0\,0\,1\,1\,1 \\ 1\,1\,1\,0\,0\,0\,0\,1 \\ 1\,0\,0\,0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0\,0\,1\,0 \\ 0\,1\,0\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0\,1\,0 \\ 1\,0\,1\,0\,0\,1\,0\,1 \\ 0\,0\,0\,0\,0\,0\,0\,0 \end{bmatrix} \tag{26}$$

Note that row $1,\ldots,11$ of the binary encoding of C_1, C_2 and C_3 modular represent the 11 environment patterns shown in Fig.5. The last row is all zeros that represents not any obstacle is around the robot. Figure .12 shows that the actual paths are taken by SMC and MMC. SMC uses C_1 and MMC uses C_1 and C_2.

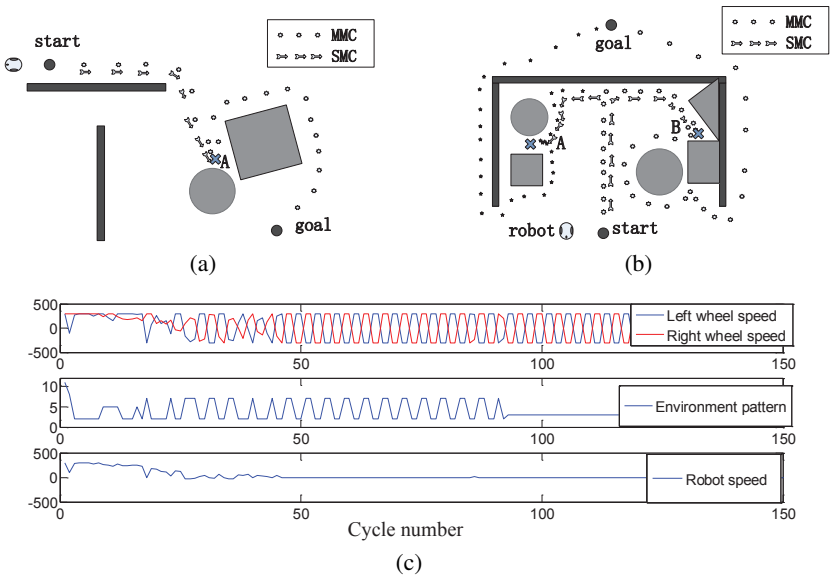


Fig. 12. Escape from a local minimum in a complex environment

There are several local minimum traps in Figure.12(a) and (b). As shown in Figure.12, MMC can break away from both of the local minimum trap and arrive at the destination successfully as in Figure.12(a) and (b). But SMC can not struggle to break away from the local minimum trap (A) point in Figure.12(a), and sometimes can not break

away from the local minimum trap (A) or (B) in Figure.12(b). It has been shown in Figure.12(c) that the SMC meets one local minimum trap at (A) point while navigating towards the goal in Figure.12(a). SMC alternately judge the environment patterns by switching from case 2 to 7 constantly while reaching the edge of the trap. The environment pattern changes to case 3 (Hall way) while reaching the bottom of the neck trap. But if the robot size is bigger than the spacing of the hall way, it will fall into the trap and the robot speed will become zero while the left and right wheel are still running. But MMC can move away from the trap successfully because MMC judge this pattern as 9 cases and would activate U-turn behavior to break away from the trap.

Table 2. Performance comparisons between SMC and MMC while escape from local minimum

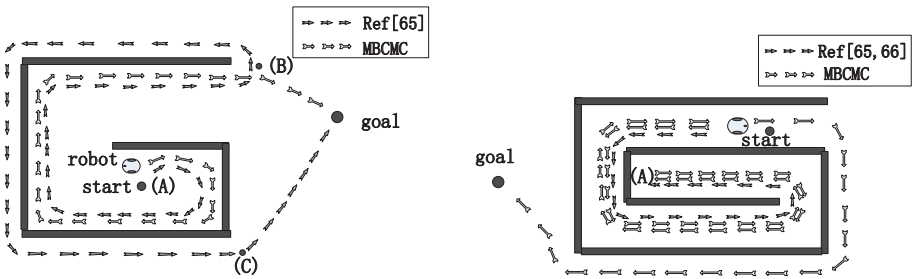
		N_{module}	N_{fail}	T_{tog} (s)	L_{tog} (mm)	$D_{obs}^{\min}$ (mm)	N_{coll}
Figure.12(a)	SMC	1	10	fail	fail	fail	fail
	MMC	2	0	48.7	355.3	1.32	0
	Single NN [53]	1	10	fail	fail	fail	fail
	M-NN [53]	5	0	49.1	357.6	1.37	0
Figure.12(b)	SMC	1	4	64.2	564.2	1.53	4
	MMC	2	0	57.1	528.3	1.68	1
	MMC	3	0	56.9	526.9	1.71	0

Under the same obstacle configuration as in Figure.12, we have changed the start and goal positions and ran the experiment ten times. For the same navigation task, Table. 2 is listed in the performance of SMC and MMC. The N_{module} is the number of modular and N_{fail} is the number of failures. Whenever SMC fall into the trap in Figure.12(a) every time, N_{fail} is 10. But MMC can have better identification of the environment and can move away from the trap successfully, where N_{fail} is 0 and T_{tog} is the total execution time. Since MMC has a low elapsed time, it has better performance than SMC. The L_{tog} is the total length of the path and SMC has the longer path length than MMC because it walks a duplicate path due to the environment model identify error. Again, $D_{obs}^{\min}$ is the minimum distance between obstacle and robot's sensor and this index indicates that the risk taken through the entire movement. The number of collisions N_{coll} indicates a safe navigation. The results in Table.2 show that MMC has a better performance than SMC. In addition, unlike in [53] where "5-by-1" modular neural network (M-NN) environment classifier is required to replace single NN classifier to realize successful navigation in Figure.12(a). This paper considers only two kinds of modular (C_1, C_2) to achieve the same task. As shown in Table.2, the performance of different sizes of $MMC(2,3)$ for Figure.12(b) is not obvious. So, the number of modules used depends on the specific local environment. Furthermore, NN environment classifiers need larger and greater amount of samples to train the controller. There is a need for about 3000 ultrasonic patterns to train NN classifiers [53] and $50,000$ samples with speed of 4.5 hours for training the navigation reservoirs [70]. But SMC or MMC based on ENPS does not need to train any processing and is simple to initialize the environment model.

4.2 Experiment for Multi-Behavior Coordinator

Several behavior coordination schemes are employed to evaluate the performance, such as fuzzy logic approach [65], expert fuzzy cognitive map (FCM)-based approach [66], fuzzy discrete event systems FDES-based approach [62], optimized modular NN approach [53], modification of potential field method [71]. Throughout the experiments, we have adopted modular (C_1, C_2) as in MMC in Test I and Test II, modular (C_1,...,C_3) as MMC in Test III. The following simulation tests are carried out for validating the proposed approach.

Test I: **G-shape and snail shape environment:** Figure.13 shows the expected results as previously depicted in MBCMC. In [65], a new fuzzy logic controller for robot navigation has been developed, which has adopted an actual-virtual target to escape from the local minimum by defining a sum of turning angles. If the sum of turning angles throughout the way is near 0^o, the robot would decide to go toward the real target. If the total amount of turning angles is negative, then the robot will have a counterclockwise motion to compensate the amount at the opening point. Since the sum of turning angles is -360^o at point "(B)" in Figure.13(a), the robot will not execute goal reaching and will turn counterclockwise to continue following the wall until the point "(B)". But MBCMC will switch the control scheme and run towards the goal directly. Although after breaking away from the G-shape obstacle [65], it will spend more time and run more distance than MBCMC to get goal point.



(a) G-shape (b) snail shape

Fig. 13. Escape from a G-shape and snail shape environment

Snail shape in Figure.13(b) is more complicated trap than G-shape. The distance between the corridors of the snail must be wide enough. The robot in [65] after encountering the first wall (left side or right side), follow the left side or right side wall and then break away from the snail shape obstacle successfully. But the snail shape environment in this paper has a very narrow corridor and with a dead end. Hence, it will effect the definition of the virtual target [65] and event weights of the expert-FCM graph [66]. Also, the robot falls into a trap at dead ends "(A)" as shown in Figure.13(b). Since hall way and dead end are in the general definition of environment patterns and MBCMC can identify those cases in this paper. Moreover, the wall following method is also modified

by Formula (25) to a suitable corridor environment. The results of Figure.13(b) prove that the robustness of the proposed approach is better than the approaches in [65, 66], whether it is a wide corridor or narrow corridor.

Test II: **Building environment:** The robot starts in room 1 and navigates to the goal at room 2. Figure.14 shows that both MBCMC and M-NN [53] started at room 1, crossed the narrow corridor and arrived at the turning point "(A)". MBCMC can implement self-rotation strategy according to the environment model and aim the room 2 as the goal. But the robot (M-NN [53]) failed to enter through the "door" at (A), because it was confused by the corridor module and the left turn module (adopt the competitive coordination). The robot ([53]) can break away the dead end "(B)" in Figure.14, but it spends more time to reach the goal than the proposed approach.

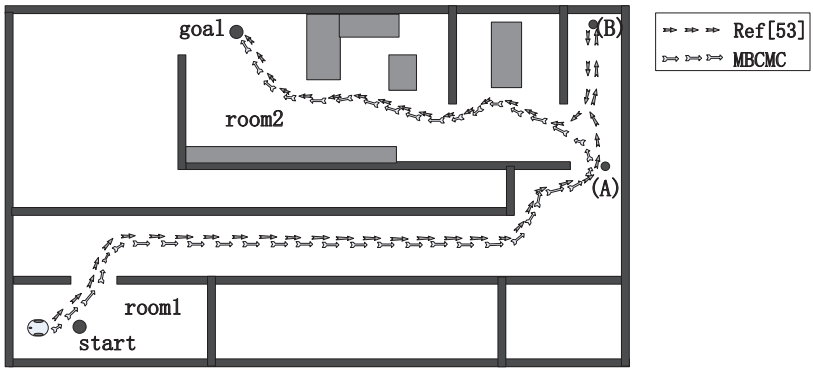
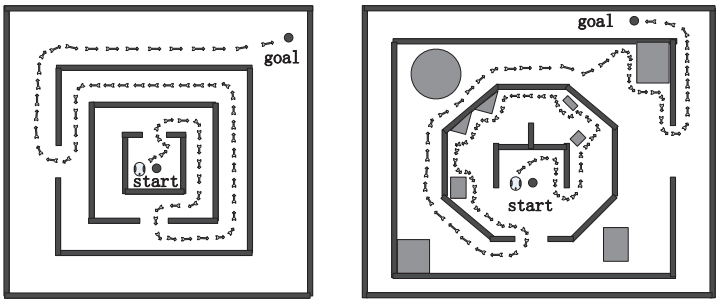


Fig. 14. Robot starts at room 1 and goes to room 2

Test III: **Maze environment:** The performance of MBCMC was examined in the similar environments ([62]) with more complex mazelike traps in Figure.15. Figure.15(b) shows the similar navigation scenarios of the robot moving in the maze environment with irregular obstacles. FDES-based approach [62] employs supervisory control theory of fuzzy discrete event systems to model and control several navigation task of a mobile robot. Two deliberative behaviors ("Go to Target"(*GT*) and "Route Follow" (*RF*)) and three reactive behaviors ("Wall Follow"(*WF*), "Avoid Obstacle"(*AO*) and "Avoid Dead ends"(*AD*)) are weighted through FDES and navigate the robot to the final target successfully. In this method, target seeking is based on following a series of immediate sub-targets (waypoint). *GT* is used for path optimization and aims to find the next nearest waypoint. *RF* is used to navigate the robot through way points. Therefore the robot can trace a collision-free path with optimum distance towards the actual target in mazelike environments. Unlike in [62], the start and end points are identified and moreover the waypoints are given manually. The robot in this paper only knowns the start point and goal point and also can identify the surrounding unknown environments by MMC

accurately. The dynamically chosen reasonable behaviors by MBCMC, the AMR can help to walk out of the maze safely. Figure.15 shows the traveled trajectory with these environments, where both MBCMC and FDES-based approach have the similar path. Also, Figure.16 depicts the angle error between robot direction and goal, environment pattern and speed control results of MBCMC related to the complex maze-like environment.

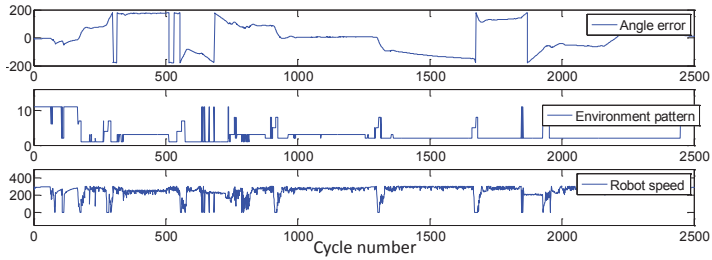


(a) Maze 1 (b) Maze 2

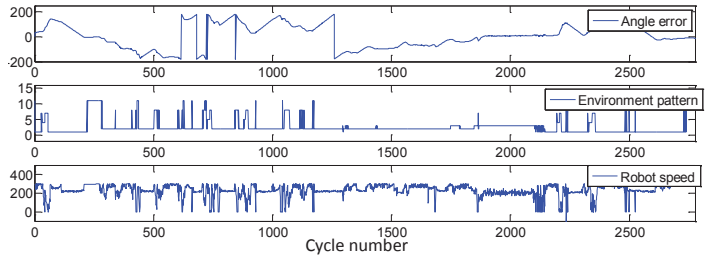
Fig. 15. Escape from a maze environment

5 Conclusions

In this paper, we presented a hybrid control framework for robust robot navigation tasks based on P systems. A simple and effective environment pattern membrane classifier is proposed, which can be identified by 11 environment patterns and can build or modify environment modules quickly. The proposed MBCMC possesses multi behavior dynamic coordination scheme in reactive layer. It is observed that the proposed MMC and MBCMC are able to provide a robust and successful navigation with a smooth path in different type of environments, while the other approaches failed to do so in more complex environments. The proposed approach eases the design of the behavior-based hybrid control architectures with the higher modularity which is obtained by associating P systems. Moreover, MMC with binary environment model is able to cope with sensor imprecision and ambiguous situations. Reflective (such as obstacle avoidance and wall following) and reactive behavior co-ordination have succeeded by properly blending the outputs with the ambiguous situation modules. Also, introduction of more behaviors to the membrane hybrid control architecture is easily performed by adding more environment models to MMC and events to MBCMC. To address the more complex navigation tasks, studies on decentralized and modular membrane controller can be carried out in the future.



(a) Control result to maze 1



(b) Control result to maze 2

Fig. 16. Control results of MBCMC related to mazelike environment

Acknowledgements

The work is supported by the National Natural Science Foundation of China (61672437, 61702428, 61771411), the Sichuan Science and Technology Program (2018GZ0185, 2018GZ0086), the Fundamental Research Funds for the Central Universities (2682016CX038) and New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0044)..

References

1. Gh. Păun, G. Rozenberg and A. Salomaa, *The Oxford Handbook of Membrane Computing*. NY, USA: Oxford University Press, (2010).
2. Gh. Păun, Membrane Computing - An Introduction. Berlin: Springer,(2002).
3. G. Zhang, M. Gheorghe, L. Pan, M.J. Pérez-Jiménez, Evolutionary membrane computing: a comprehensive survey and new results. *Information Sciences*, **279**(2014), 528-551.
4. G. Zhang, M.J. Pérez-Jiménez, M. Gheorghe, Real-life Applications with Membrane Computing (Emergence, Complexity and Computation). Berlin Germany: Springer, (2017).
5. G. Zhang, J. Cheng, T. Wang, X. Wang, J. Zhu, *Membrane Computing:Theory & Applications*, Science Press, Beijing, China,(2015).
6. M., Gheorghe, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, Research Frontiers of Membrane Computing: Open Problems and Research Topics. *International Journal of Foundations of Computer Science*, **24**(5)(2013), 547-624.

7. Z., Zhang, T., Wu, A., Păun, L., Pan: Numerical P systems with migrating variables, Theoretical Computer Science, **641**(2016), 85-108.
8. Z., Zhang, T., Wu, A., Păun, L., Pan: Universal enzymatic numerical P systems with small number of enzymatic variables, Science China Information Sciences, (2018), DOI:10.1007/s11432-017-9103-5.
9. L., Pan, Z., Zhang, T., Wu, J., Xu: Numerical P systems with production thresholds, Theoretical Computer Science, **673**(2017), 30-41.
10. Z. Zhang, L. Pan: Numerical P systems with thresholds. International Journal of Computers Communications & Control, **11**(2)(2016), 292-304.
11. G. Zhang, F. Zhou, X. Huang, J. Cheng, M. Gheorghe, F. Ipate, R. Lefticaru, A novel membrane algorithm based on particle swarm optimization for solving broadcasting problems, *Journal of Universal Computer Science* **18**(13)(2012), 1821-1841.
12. M.H Rafiei, H. Adeli, A New Neural Dynamic Classification Algorithm, *IEEE Transactions on Neural Networks and Learning Systems*, **28**(12)(2017), 3074-3083 (doi: 10.1109/TNNLS.2017.2682102).
13. H. Adeli, H. S. Park, Neurocomputing for Design Automation. CRC Press,(1998).
14. H. Adeli, H. Park, Method and apparatus for efficient design automation and optimization, and structure produced thereby, US Patent 5,815,394,29 September 1998.
15. H. Adeli, H. S. Park, A neural dynamics model for structural optimization-theory, *Comput. Struct.*, **57**(3)(1995), 383-390.
16. H. Adeli, H. Kim, Cost optimization of composite floors using the neural dynamics model, *Commun. Numer. Meth. Eng.*, **17**(11)(2001), 771-787.
17. H. Adeli, H. S. Park, Optimization of space structures by neural dynamics, *Neural Networks*, **8**(5)(1995), 769-781.
18. M. H. Rafiei, H. Adeli, NEEWS: A novel earthquake early warning model using neural dynamic classification and neural dynamic optimization Soil Dynamics and Earthquake Engineering, **100**(2017), 417-427.
19. S. Ghosh-Dastidar, H. Adeli, Improved spiking neural networks for EEG classification and epilepsy and seizure detection, *Integr. Comput.-Aided Eng.*, **14**(3)(2007), 187-212.
20. H. S. Park, H. Adeli, Distributed neural dynamics algorithms for optimization of large steel structures, *J. Struct. Eng.*, **123**(7)(1997), 880-888.
21. E.K. Zavadskas, J. Antucheviciene, Z. Turskis, H. Adeli. Hybrid multiple-criteria decision-making methods: A review of applications in engineering, *Scientia Iranica*, **23**(1)(2016), 1-20.
22. M. S. Gutierrez, H. Adeli, Many-objective control optimization of high-rise building structures using replicator dynamics and neural dynamics model Structural and Multidisciplinary Optimization, **56** (6)(2017), 1521-1537 (doi: 10.1007/s00158-017-1835-9).
23. J. Xiao, Y. Huang, Z. Cheng, J. He and Y. Niu, A hybrid membrane evolutionary algorithm for solving constrained optimization problems, *Optik*, **125**(2)(2014), 897-902.
24. G. Zhang , J. Cheng, M. Gheorghe, et al. A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems. *Applied Soft Computing*, **13**(3)(2013), 1528-1542.
25. G. Zhang, H. Rong, F. Neri, M.J. Pérez-Jiménez. An optimization spiking neural P system for approximately solving combinatorial optimization problems, *International Journal of Neural Systems*, **24**(5)(2014), Article No.1440006.
26. J. L. Rosselló, V. Canals, A. Oliver and A. Morro, Studying the Role of Synchronized and Chaotic Spiking Neural Ensembles in Neural Information Processing, *International Journal of Neural Systems*, **24**(5)(2014), Article No.1440003.
27. M.A. Colomer, A. Montorib, E. Garcia and C. Fondevilla, Using a bioinspired model to determine the extinction risk of Calotriton asper populations as a result of an increase in extreme rainfall in a scenario of climatic change, *Ecological Modelling*, **81**(2014), 1-14.

28. S. Shapero, M. Zhu, J. Hasler, C. J. Rozell, Optimal Sparse Approximation with Integrate and Fire Neurons, *International Journal of Neural Systems*, **24**(5)(2014), Article No.1440001.
29. C. Buiu, C. I. Vasile, and O. Arsene, Development of membrane controllers for mobile robots. *Inform. Sci.*, **187**(2012), 33-51.
30. Pavel.A.B, Buiu.C, Using enzymatic numerical P systems for modeling mobile robot controllers. *Natural Computing*, **11**(3)(2012), 387-393.
31. C. I. Vasile, A. B. Pavel, I. Dumitrache and G. Păun, On the power of enzymatic numerical P systems, *Acta Informatica*, **49**(6)(2012), 395-412.
32. Pavel.A.B, Vasil.C.I, Dumitrache.I, Robot localization implemented with enzymatic numerical P systems. Proceedings of the International Conference on Biomimetic and Biohybrid Systems. Barcelona, Spain,(2012), 204-215.
33. Gh. Păun, R. Păun. Membrane computing and economics: Numerical P systems. *Fundamenta Informaticae*, **1**(73)(2006), 213-227.
34. Wang X, Zhang G, Neri F, 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**(2016), 15-30.
35. L. Pan, G. Păun, G. Zhang, F. Neri, Spiking Neural P Systems with Communication on Request, *International Journal of Neural Systems*, **27**(8)(2017),art. no.1750042.
36. F, Neri, E. Mininno, G. Iacca, Compact particle swarm optimization, *Information Sciences*, **239**(2013), 96-121.
37. S. Rostami, F. Neri, M. Epitropakis, Progressive preference articulation for decision making in multi-objective optimization problems, *Integrated Computer-Aided Engineering*, **24**(4)(2017), 315-335.
38. T. Wu, F. D. Bible, A. Păun, L. Pan, F. Neri, Simplified and Yet Turing Universal Spiking Neural P Systems with Communication on Request, *International Journal of Neural Systems*, **28**(08)(2018), 1850013.
39. F. Neri, A. Caponio, A differential evolution for optimisation in noisy environment, *International Journal of Bio-Inspired Computation*, **2**(3-4)(2010), 152-168.
40. G. Iacca, F. Caraffini, F. Neri, Multi-strategy coevolving aging particle optimization, *International Journal of Neural Sysems*, **24**(1)(2014), 1450008.
41. S. Rostami, F. Neri, Covariance matrix adaptation pareto archived evolution strategy with hypervolume-sorted adaptive grid algorithm, *Integrated Computer-Aided Engineering*, **23**(4)(2016), 313-329.
42. D. Nakhaeinia, S.H. Tang, S.B. Mohd Noor, O. Motlagh, A review of control architectures for autonomous navigation of mobile robots, *Int J Phys Sci*, **6** (2011), 169-174.
43. R.J.Wai, C.M.Liu, Y.W.Lin, Design of switching path-planning control for obstacle avoidanceofmobilerobot, *Journal of The Franklin Institute.*,**348**(2011), 718-737.
44. P.Pirjanian, Multiple objective behavior-based control, *Robot Auton Syst.*, **31**(2000), 53-60.
45. D. Nakhaeinia, B. Karasfi, A behavior-based approach for collision avoidance of mobile robots in unknown and dynamic environments, *Journal of Intelligent and Fuzzy Systems*, **24**(2)(2013), 299-311.
46. R. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*, **2**(1986), 14-23.
47. V.J. Lumelsky and A.A. Stepanov, Dynamic path planning for a mobile automation with limited information on the environment, *IEEE Transaction*, **31** (1986), 1058-1063.
48. Y. Koren,J .Borenstein, Potential field methods and their inherent limitations for mobile robot navigation, *in:Proc.IEEE Conf.on Robotics and Automat*,(1991), 1398-1404.
49. J. Minguez, L. Montano, Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios, *IEEE Trans. Robot. Autom*,**20**(1), (2004), 45-59.

50. X.Yang, M. Moallem, R.V.Patel, A layered goal-oriented fuzzy motion planning strategy for mobile robot navigation, *IEEE Trans.Systems, Man, Cybernet.,PartB: Cybernet*, **35**(6)(2005), 1214-1224.
51. S. Horn, K. Janschek, A Set-based Global Dynamic Window Algorithm for Robust and Safe Mobile Robot Path Planning, *Proceedings of the 41st International Symposium on Robotics and the 6th German Conference on Robotics Munich, Germany*,(2010), 1-7.
52. K.Y.Tu, J. Baltes, Fuzzy potential energy for a map approach to robot navigation, *Robot. Auton. Syst.*, **54**(7)(2006), 574-589.
53. S.J.Han, S.Y.Oh, An optimized modular neural network controller based on environment classification and selective sensor usage for mobile robot reactive navigation, *Neural Computing and Applications*, **17**(2)(2008), 161-173.
54. Lumelsky. VJ, Stepanov. AA, Path-planning strategies for a point mobile automaton moving amidst obstacles of arbitrary shape, *Algorithmica*, **2**(1987), 403-430.
55. T. Zhang, Y. Zhu, J. Song, Real-time motion planning for mobile robots by means of artificial potential field method in unknown environment, *Ind. Robot. Int. J.*, **37**(2010), 384-400.
56. K.Bence, Szayer.G, T.Ferenc, B.Mauricio, P.Korondi, A novel potential field method for path planning of mobile robots by adapting animal motion attributes, *Robotics Autonomous Systems*, **82**(2016), 24-34.
57. J. Ng,T. Braunl, Performance comparison of bug navigation algorithms, *J. Intell. Robot. Syst.*, **50**(1),(2007), 73-84.
58. Y. Gabriely, E. Rimon, CBUG: A quadratically competitive mobile robot navigation algorithm, *IEEE Trans. Robot.*, **24**(6)(2008), 1451-1457.
59. F. Mastrogiovanni, A. Sgorbissa and R. Zaccaria, Robust navigation in an unknown environment with minimal sensing and representation, *IEEE Trans. Syst. Man Cybern*, **39**(1)(2009), 212-229.
60. G. Antonelli, S. Chiaverini and G. Fusco, A fuzzy-logic-based approach for mobile robot path tracking, *IEEE Transactions on Fuzzy Systems*, **15**(2007), 211-221.
61. J.H. Lilly, Evolution of a negative-rule fuzzy obstacle avoidance controller for an autonomous vehicle, *IEEE Trans on Fuzzy Systems*, **15**(2007), 718-728.
62. A.Jayasiri, G.Mann, R.Gosine, Behavior coordination of mobile robotics using supervisory control of fuzzy discrete event systems, *IEEE Transactions on Systems Man Cybernetics*, **41**(5)(2011), 1224-1238.
63. S.H. Tang, D. Nakhaeinia, B. Karasfi, Application of fuzzy logic in mobile robot navigation, fuzzy logic-controls, concepts, theories and applications, ISBN: 978-953-51-0396-7,InTech, (2012).
64. M.Scheutz, V.Andronache, Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems, *IEEE Systems, Man, and Cybernetics*, **34**(6)(2005), 2377-95.
65. O.Motlagh, S.Tang, N.Ismail, Development of a new minimum avoidance system for a behavior-based mobile robot, *Fuzzy Sets Systems*, **160**(13)(2009), 1929-1946.
66. O.Motlagh, SH.Tang, N.Ismail, AR.Ramli, An expert fuzzy cognitive map for reactive navigation of mobile robots, *Fuzzy Sets Systems*, **201**(12)(2012), 105-121.
67. M.Janiak, C.Zielinski, Control system architecture for the investigation of motion control algorithms on an example of the mobile platform Rex, *Bulletin of the Polish Academy of Sciences Technical Sciences*, **63**(3)(2015), 667-678.
68. E. Freire, T. Bastos-Filho, M. Sarcinelli-Filho, and R. Carelli, A new mobile robot control approach via fusion of control signal, *IEEETrans. Syst., Man, Cybern. B*,**34**(2004), 419-429.
69. R. Carelli, EO. Freire, Corridor navigation and wall-following stable control for sonar-based mobile robots, *Robotics Autonomous Systems*, **45**(3)(2003), 235-247.

70. E.A.Antonelo, B.Schrauwen, On Learning Navigation Behaviors for Small Mobile Robots with Reservoir Computing Architectures, *IEEE Transactions on Neural Networks Learning*, **26**(4),(2015), 763-780.
71. M.Guerra, D.Efimov, G.Zheng, W.Perruquetti, Avoiding local minima in the potential field method using input-to-state stability, *Control Engineering Practice*,**55**(2016), 174-184.
72. Păun Gh, Perez-Jimenez M J, Riscos-Nunez A. Tissue P systems with cell division. *Int. J. Comput. Commun. Control*, **3**(2008), 295-302.
73. Diaz-Pernil D, Gutierrez-Naranjo M A, Perez-Jimenez M A, et al. A linear-time tissue P system based solution for the 3-coloring problem. *Electr. Notes Theor. Comput. Sci.*, **171**(2007), 81-93.
74. Diaz-Pernil D, Gutierrez-Naranjo M A, Perez-Jimenez M A, et al. A uniform family of tissue P system with cell division solving 3-COL in a linear time. *Theor. Comput. Sci*, **404**(2008), 76-87.
75. Diaz-Pernil D, Gutierrez-Naranjo M A, Perez-Jimenez M A, et al. Solving subset sum in linear time by using tissue P system with cell division. *Lect. Notes Comput .Sci.*, **4527**(2007), 170-179.
76. Diaz-Pernil D, Gutierrez-Naranjo M A, Perez-Jimenez M A, et al. A fast solution to the partition problem by using tissue-like P systems. In: Proceedings of the Third International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2008. Adelaide, SA, Australia, (2008), 43-48.
77. Păun, G.: P systems with active membranes: Attacking NP-complete problems. *J. Automata, Languages and Combinatorics*, **6**(2001), 75-90 .
78. Martin-Vide C, Păun A, Păun Gh, On the power of P systems with symport rules.*Journal of Universal Computer Science*, **8**(2)(2002), 317-331.
79. P. Sosik, A. Rodriguez-Paton, Membrane computing and complexity theory: A characterization of PSPACE, *J. Comput. Syst. Sci.*, **73**(2007), 137-152.
80. C. Martin-Vide, Gh. Păun, J. Pazos, A. Rodriguez-Paton, Tissue P systems, *Theoretical Computer Science*, **296**(2)(2003), 295-326.
81. H. A. Chrstinal, D. D. Pernil, P. Real, Region-based segmentation of 2D and 3D images with tissue-like P systems, *Pattern Recognition Letters*, **32**(2011), 2206-2212.
82. D. D. Pernil, M. A. Gutierrez-Naranjo, H. M. Abril, P. Real, Designing a new software tool for Digital Imagery based on P systems, *Natural Computing*, **11**(2012), 381-386.
83. M. Ionescu, Gh. Păun, T. Yokomori, Spiking neural P systems, *Fundamenta Informaticae*, **71**(2-3)(2006), 279-308.

A μfluidic system design and simulation for Spiking neural P systems

Tseren-Onolt Ishdorj[1⋆], Otgonnaran Ochirbat[2], Chuluunbandi Naimannaran[3]

[1] Department of Computer Science
School of Information and Communication Technology
Mongolian University of Science and Technology
Ulaanbaatar 21335, Mongolia
tseren-onolt@must.edu.mn
[2] Department of Information and Computer Science
School of Engineering and Applied Sciences
National University of Mongolia
Ulaanbaatar 14201, Mongolia
otgonnaran@seas.num.edu.mn
[3] Department of Communication Technology
School of Information and Communication Technology
Mongolian University of Science and Technology
Ulaanbaatar 21335, Mongolia
chuluunbandi@must.edu.mn

Abstract. In this paper, we propose a preliminary microfluidic computing system design for spiking neural P systems. The system is designed particularly to solve the computational hard problem Boolean satisfiability SAT by implementing the model studied in [10]. We have also developed a computer model for the considering system and have been doing *in silico* experiments. Dielectrophoretic force (DEP), generated in the microfluidic channels by AC voltage facilitated electrodes, is employed as the main functioning tool of the proposing computing system.

Keywords: Microfluidic system, spiking neural P systems, parallel and distributed computation, computational hard problems

1 Introduction

Application and implementation related research of membrane computing (also so-called P systems) theory have been conducted actively in the recent years [30, 2, 26].

In the present work we aim to propose a biochip based implementation design for Spiking Neural P Systems (SN P systems, for short).

The approach used in the proposing biochip device design is microfluidic systems technology, where information is encoded in biological atomic particles

⋆ Corresponding author.

such as blood cells and bacteria. The computation on the encoded information takes place in parallel in the microfluidic network channels while the information carries fluid flows throughout the device.

Main functioning characteristic of the considering system is physical properties of biological particles, microfluid, and fluidic microchannel geometry as well as particle motion related to governing forces of fluidic velocity, viscosity, drag forces, electric field gradient, dielectrophoresis DEP, etc.

DEP force is considered as a rather efficient and easier tool to govern bioparticles in microchannels than other approaches of acoustic, inertia and magnetic forces.

The considering computing device design is a type of implementation of a particular SN P systems, which solves a computational hard problem of the Boolean satisfiability `SAT` presented in [10]. A computation model for the microfluidc system is developed on COMSOL multiphysics software and have been carried out *in silico* experiments, which is presented in Section 4.

Another approach of microfluidic system design for P systems to solve computational hard problems has been introduced in [13], where DNA molecules are facilitated for data storage and manipulation mechanism rather than cells and/or biophysical forces. A number of studies on Tissue P Systems have been conducted recently [25, 23, 24].

Spiking neural P systems (in short, SN P systems) were introduced in [15] as a class of computing devices inspired from the way the neurons cooperate by exchanging *spikes*, electrical impulses of identical shapes. Other type of SN P systems we refer reader to [17, 19] and [27, 26]. Details can be also found at the membrane computing website, at `http://ppage.psystems.eu`.

The paper is organized as follows: in the next section we specify some general definitions and notations; μfluidic P systems design with interpretation of SN P systems is introduced in Section 3, and computer model architecture and simulation are described in Section 4. Final remarks are provided in Section 5.

2 Preliminaries and notions

The reader is assumed to be familiar with general Spiking neural P systems (SN P systems, for short) [15, 18, 28] and a class of Spiking neural P systems with pre-computed recourses [10] as well as microfluidic systems [1, 12].

2.1 Microfluidics and Bio-Chip

Microfluidics is the science of studying fluid flow behavior at the microscale and the development of miniaturized analysis systems that take advantage of the unique physics emergent at these small scales.

Biochips can be defined as microelectronic-inspired devices that are used for delivery, processing, analysis, or detection of biological molecules and species. Microfluidics-based biochips have become an actively researched area in recent years. Sometimes also referred to as lab-on-a-chip, biochips integrate different

biochemical analysis functionalities (e.g., dispensers, filters, mixers, separators, detectors) on-chip, miniaturizing the macroscopic chemical and biological processes to a sub-millimetre scale [5]. Like a computer chip that can perform millions of mathematical operations in one second, a biochip can perform thousands of biological reactions, such as decoding genes, in a matter seconds.

Dielectrophoresis. Dielectrophoresis (DEP) has been used for characterizing [5, 14, 21] and separating [22, 29, 3] small artificial and biological particles since H. A. Pohl discovered it in the 1950s [20]. Fig. 1 illustrates the basic principle of the dielectrophoresis process.

The AC signals applied to electrodes generate the DEP force which makes cells move in the mixture flow in the direction where the DEP forces and directions are dependent on cell properties. Due to the different DEP force directions, the cells with different DEP responses move continuously to the different locations.

However, if the field is non-uniform, then the forces on either side of the cells will be different, and the net DEP force can induce the movement of cells [9]. If electrical polarizability of cells exceeds that of the suspending medium, the DEP force will be in the same direction as the gradient of electric fields.

In this case, cells move to the strong electric field region (positive dielectrophoresis or pDEP). On the contrary, when the electrical polarizability of cells is less than that of the medium, the direction of DEP force is reversed to the gradient of electric fields and cells move to the weak electric field region (negative dielectrophoresis or nDEP), (Fig. 1). The polarizability of cells depends strongly on their composition, morphology, phenotype and the electric field frequency; therefore, the cells of different types or physiological states including viability can be discriminated by the DEP. The time-averaged DEP force is given by [8, 7]:

$$F_{DEP} = 2\pi\varepsilon_m r^3 Re[f_{CM}] \cdot \nabla \mid E_{rms} \mid^2 \tag{1}$$

In (1), r is a radius of cell; ε_m the permittivity of the medium; f_{CM} the Clausius-Mossotti factor and E_{rms} is the root mean square value of an electric field. $Re[f_{CM}]$ means a real part of the f_{CM} which can be represented as follows:

$$f_{CM} = (\varepsilon_p^* - \varepsilon_m^*)/(\varepsilon_p^* + 2\varepsilon_m^*) \tag{2}$$

In (2), ε^* is the complex permittivity $\varepsilon^* = (\varepsilon - j\delta/\omega)$. δ the conductivity and ω is the electric field frequency. Subscripts p and m mean cells and the medium, respectively. $Re[f_{CM}] > 0$ means that cells show pDEP response while $Re[f_{CM}] < 0$ means nDEP response.

Fluid flow and velocity. When a specimen is forced into the microfluidic channel, the hydrodynamic force (F_{HD}) applied on a cell is influenced by the flow velocity. It can be described with Stokes Law, as follows:

$$\overrightarrow{F}_{HD} = 4\pi\eta R\overrightarrow{v}.$$

In which v is the relative velocity between the cell with a radius of R and the flow. This equation shows that the flow velocity is directly proportional to the

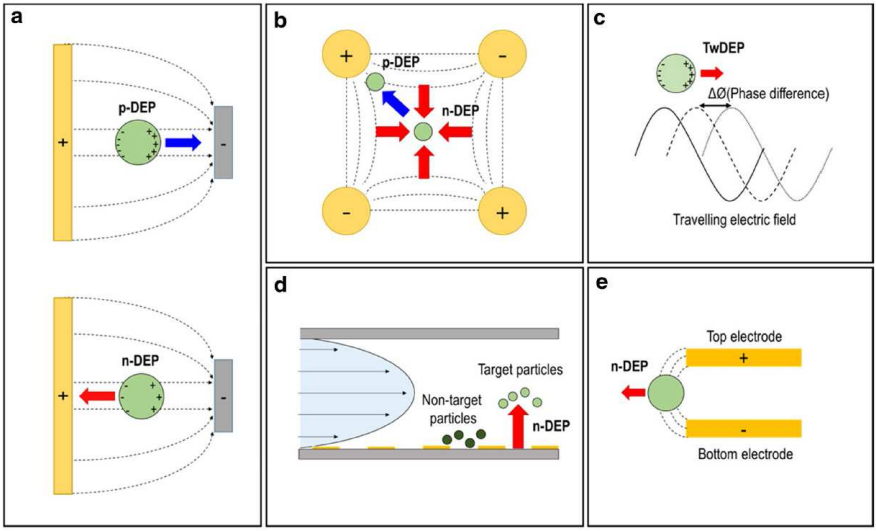


Fig. 1. Dielectrophoresis DEP force. Dielectric particle motion in fluidic and non-uniform electric field

hydrodynamic force. It must be noted that apart from the hydrodynamic force, the cell will experience a DEP force in a microchannel. Therefore, as the cells are directed into the DEP microchamber, there are some points corresponding to the electrical field gradient in the tangential direction relative to the flow. These forces cause deviation of the flow direction of the cells and direct them to their targeted outlet under F_{HD}.

The incompressible Navier-Stokes flow is used to direct the fluid flow:

$$\rho\frac{du}{dt} - \nabla \cdot \sigma + \rho\mathbf{u} \cdot \nabla\mathbf{u} = \mathbf{F}$$

In this equation, $\mathbf{u}$ is the velocity vector containing u and v components along the x, y, and z direction. Meanwhile, ρ is the fluid's density, p is the flow's pressure, and σ represents the total stress tensor of Newtonian fluid which consists of pressure stress and viscous stress.

Electric Field. The electric current interface was used to model the DEP stage, whereby the magnetic-inductive effects were neglected in this stage and only resistive-conductive and electric-capacitive effects were accounted for. In this simulation, the electric potential in the microchannel was governed by Laplace's equation of continuity such that:

$$-\nabla \cdot ((\sigma + j\omega\varepsilon_r\varepsilon_o)\nabla\varphi)$$

where σ represents the electrical conductivity of the cell, ω is the angular frequency of the driving field, ε_r is the relative permittivity of the medium, and ε_o is the relative permeability of the vacuum.

The voltage is defined as a sine wave at a particular frequency, whereby electric potentials of $+5V$ and $-5V$ are assigned across alternating electrodes embedded within the microchannel. A non-uniform electric field is established either vertically or laterally, depending on the microelectrode configuration.

Particle Trajectory. The trajectory of cells, which were suspended in the blood sample, move through microchannels is calculated by solving the equition of motion for each set of cells. The equition is governed by Newton's second law such that: $m\frac{d\mathbf{v}}{dt} = \mathbf{F_c}$ where m is the mass of cells, v is the cell velocity, and $\mathbf{F_c}$ is the total force experienced by the bioparticles such as cells.

2.2 Spiking neural P systems with pre-computed resources

In this work we consider a special class of SN P systems that a pre-computed recourse of possibly exponential size are give in advance. This model has been introduced and studied in a number of works and proved to be an efficient problem solving abstract machine [6, 10, 11].

In SN P systems the cells (also called *neurons*) are placed in the nodes of a directed graph. The contents of each neuron consists of a number of copies of a single object, called *spike*, and a number of *firing* and *forgetting* rules.

Firing rules allow a neuron to send information to other neurons in the form of spikes (electrical impulses) which are accumulated at the target cells. The applicability of each rule is determined by checking the contents of the neuron against a regular set associated with the rule. In each time unit, if a neuron can use some of its rules then one of such rules must be used. The rule to be applied is non-deterministically chosen. Thus, the rules are used in a sequential manner in each neuron, but the neurons function in parallel with each other.

If no firing rule can be applied in a neuron, there may be the possibility to apply a *forgetting* rule, that removes a predefined number of spikes from the neuron.

Formally, a *Spiking Neural P Systems* (SN P systems, for short) of degree $m \geq 1$, as defined in [16] in the computing version (i.e., able to take an input and provide an output), is a construct of the form

$$\Pi = (O, \sigma_1, \sigma_2, \cdots, \sigma, syn, in, out)$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \sigma_2, \ldots, \sigma$ 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. *firing* (also *spiking*) rules $E/a^c \to a; d$, where E is a regular expression over a, and $c \geq 1, d \geq 0$ are integer numbers; if $E = a^c$, then it is usually written in the simplified form: $a^c \to a; d$; similarly, if $d = 0$ then it can be omitted when writing the rule;

ii. *forgetting* rules $a^s \to \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \to a; d$ of type (1) from R_i, we have $a^s \notin L(E)$ (the regular language defined by E);

3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of *synapses* between neurons;
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and the *output* neurons of Π, respectively.

A firing rule $E/a^c \to a; d \in R_i$ can be applied in neuron σ_i if it contains $k \geq c$ spikes, and $a^k \in L(E)$. The execution of this rule removes c spikes from σ_i (thus leaving $k-c$ spikes), and prepares one spike to be delivered to all the neurons σ_j such that $(i, j) \in syn$. If $d = 0$ then the spike is immediately emitted, otherwise it is emitted after d computation steps of the system. As stated above, during these d computation steps the neuron is *closed* and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost), and cannot fire (and even select) rules. A *forgetting* rule $a^s \to \lambda$, can be applied in neuron σ_i if it contains *exactly* s spikes; the execution of this rule simply removes all the s spikes from σ_i.

Boolean satisfiability problem. Let us consider the **NP**-complete decision problem `SAT` ([4]). The instances of `SAT` depend upon two parameters: the number n of variables, and the number m of clauses. We recall that a *clause* is a disjunction of literals, occurrences of x_i or $\neg x_i$ built on a given set $X = \{x_1, x_2, \ldots, x_n\}$ of Boolean variables. Without loss of generality, we can avoid the clauses in which the same literal is repeated or both the literals x_i or $\neg x_i$, for any $1 \leq i \leq n$ occur. In this way, a clause can be seen as a *set* of at most n literals. An *assignment* of the variables $x_1, x_2, \ldots, x_n$ is a mapping $a : X \to \{0, 1\}$ that associates to each variable a truth value. The number of all possible assignments to the variables of X is 2^n. We say that an assignment satisfies the clause C if, assigned the truth values to all the variables which occur in C, the evaluation of C (considered as a Boolean formula) gives 1 (*true*) as a result.

We recall a way of encoding of any given instance γ_n of `SAT` (n, m) in spikes from [10]. Each clause C_i of γ_n can be seen as a disjunction of at most n literals, and thus for each $j \in \{1, 2, \ldots, m\}$ either x_j occur in C_i, or $\neg x_j$ occurs, or none of them occurs. In order to distinguish these three situations, we define the *spike variables* α_{ij}, for $1 \leq i \leq m$ and $1 \leq j \leq n$, as variables whose values are the amounts of spikes, and we assign to them the following values:

$$\alpha_{ij} = \begin{cases} a & \text{if } x_j \text{ occurs in } C_i \\ a^2 & \text{if } \neg x_j \text{ occurs in } C_i \\ \lambda & \text{otherwise.} \end{cases}$$

In this way, clause C_i will be represented by the sequence $\alpha_{i1}\alpha_{i2}\ldots\alpha_{in}$ of spike variables; in order to represent the entire formula γ_n we just concatenate the representations of the single clauses, thus obtaining the sequence $\alpha_{11}\alpha_{12}\ldots\alpha_{1n}\alpha_{21}\alpha_{22}\ldots\alpha_{2n}\ldots\alpha_{m1}\alpha_{m2}\ldots\alpha_{mn}$. As an example, the representation of $\gamma_3 = (x_1 \vee \neg x_2) \wedge (x_1 \vee x_3)$ is the sequence $aa^2\lambda a\lambda a$.

3 Designing biochips for SN P systems

We start with translation of SN P systems notion in terms of microfluidic P systems (μfluidic P systems) which we propose. The proposing μfluidic P system is a kind of computing device which implements the SN P system model studied in [10].

SN P system	*Fluidic Neural P system*
Neuron $\bigcirc$	$\sqsubset$ Chamber
Spike a	a Particle,
Number of spikes a^d	diametr d of particle a,
Synapse $\longrightarrow$	$=$ Microchannel ,
Input and output neurons	Inlet and Outlet chambers,
Spiking rules $a^d \rightarrow a^k$	DEP ruled particle separation,
Pre-computed spiking neural network	Pre-designed biochip circuit.

3.1 Architecture of the μfluidic P system

The architecture sketch of the μfluidic P systems is illustrated in Fig. 2, while its corresponding original version of the SN P systems structure is depicted in Fig. 3. Note that both structures are devoted to an example of `SAT` problem with 2 variables and m number of clauses (`SAT(2,m)` for short).

The μfluidic P system architecture consists of three main modules:

- Input module with duplication channels (IDM);
- Parallel and distributed computing module (PDCoM);
- Decision, detection and output module (DOM).

The first module IDM is composed of an *input chamber* and *tree-like structured duplication channels* leading to the base-channels.

A canyon-like shape is placed in the middle of microchannels which supports divides the flows and separates the particles equally in advance to bifurcate towards pair-branch channels.

Corresponding to the main computation unit of the computing model described in [10], the main computing module considering μfluidic P system is designed as PDCoM.

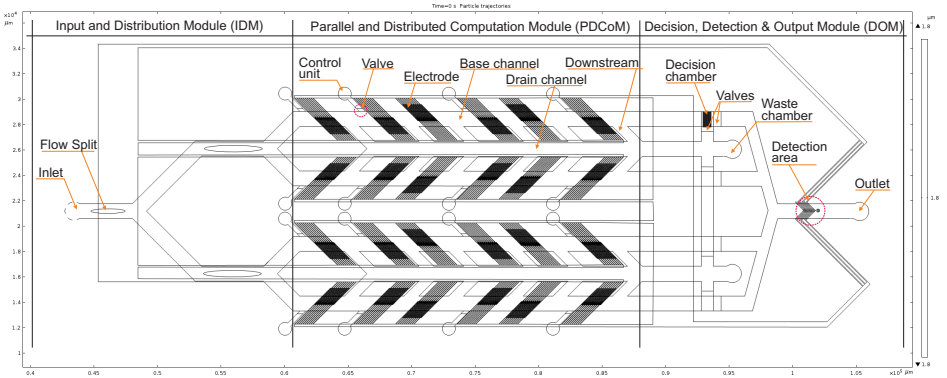


Fig. 2. μfluidic P system architecture for `SAT(2,m)`. Its original SN P systems design is depicted in Fig. 3.

PDCoM has been facilitated with two types of microchannels: a) the *base-channels* and b) the *drain-channels*. These two microchannels are connected by several downstream microchannels from the base-channel to the drain-channel. There are a number of *control units* (corresponding to the generator block in [10], see Fig.3) attached to each base-channel. On the junction of the base-channel and the control unit, an *On/Off valve* is placed to regulate downstream. Every drain-channel of the module ends up with a waste-chamber. Main functioning tool of PDCoM are DEP chambers of electrodes embedded within the bottom of the base-channels.

The last module DOM consists of the *decision chambers*, the *detection region* and the *Outlet chamber*. Each decision chamber has two valves attached, one to the drain-channel and another to the output channel, see Fig. 2.

3.2 Functioning of the μfluidic P system

Input and duplication phase. Input data (or signal) spikes are encoded into the *data-particles* dp_i, with different sizes entering into the *Inlet* chamber consequentially one after another with a certain microsecond delay to distinguish data.

For each signal data, a 2^n number of identical copies of corresponding data-particles flow simultaneously into the input chamber.

Furthermore, these entered data-particles are equally separated and flow firstly into layer of duplication channels, then again separate into the ith layer channels, then finally a single copy of each data-particle arrive at the beginning of each base-channel (there are 2^n base-channels), see Fig. 4 A). The s copies of a spike a (as a^s) of SN P systems have been translated into the μfluidic P systems as the size s of a particle dp_s .

Computation phase. The main functioning unit of the SN P systems with pre-computed recourses from [10] is constructed in the nth layer of the design struc-

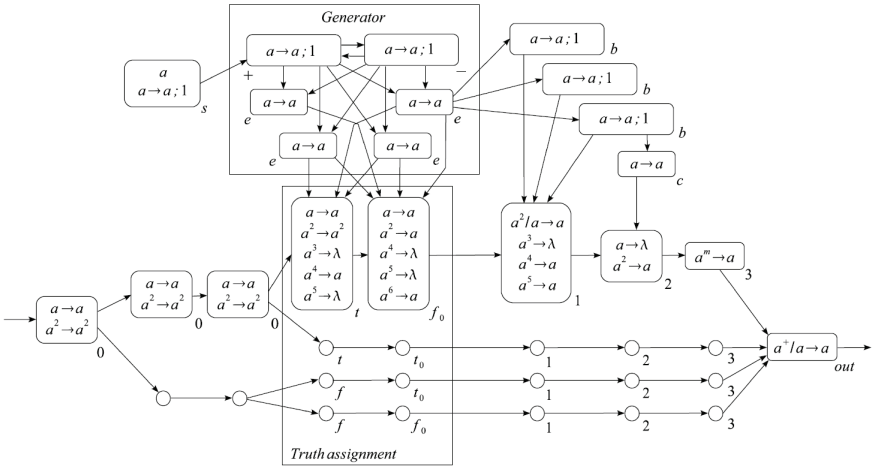


Fig. 3. SN P system design for `SAT(2,m)`, [10].

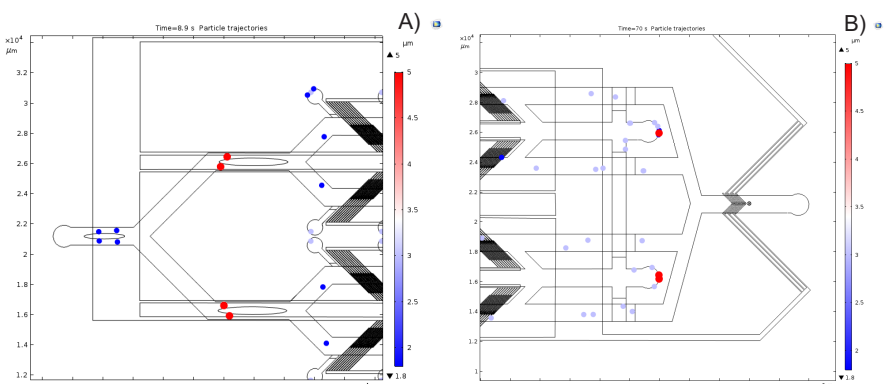


Fig. 4. Input and Output modules: A) IDM, 2^n copies of each particle represents a spiking signal which encodes a logical variable entering the *Inlet*. Spike duplication is modeled by particle bifurcation through the tree-like structured microchannels. Finally, the particles are distributed uniformly and arrive at the base-channels. B) DOM, Decision chambers count the number of particles accumulated and open either the detection channel or the waste channel in order to drain its content. Meanwhile, if there are any particles detected in the detection range which flow `YES` particles towards the Outlet, the waste particles collected in the waste chambers are sucked out.

ture, see Fig. 3. This unit's function is to check whether the considering problem has been satisfied or not by assigning all possible truth assignments to all propositional variables and then it tests all assignments which satisfy the Boolean formula.

In the current microfluidic system model, the main computing module is constructed in PDCoM, see Fig. 2, as mentioned above. At some points, an n number of data-particles $dp_k, 1 \leq k \leq n$, corresponding to an n number of Boolean variables $x_{ik}, 1 \leq k \leq n$ of a clause C_i take place at its related Boolean control units, respectively.

At that moment, the *valves* of the control units switch to *On* mode and release a Boolean control-particle, either dp_3 for truth value *true* or dp_4 for truth value *false* with size of 3 and 4, respectively, into the base-channels and immediately turn back on the *Off* mode. The meaning of the valve *On/Off* function is that it mimics Boolean computations illustrated in Table 1 of [10]), where Boolean values are assigned to the variables. Checking satisfiability of a clause is performed in the DEP chambers by separating the data-particles at the electrode region and streaming them towards the base-channel and down to the drain-channel, respectively. A detailed scheme of particle separation in the DEP chambers is illustrated in Fig. 5.

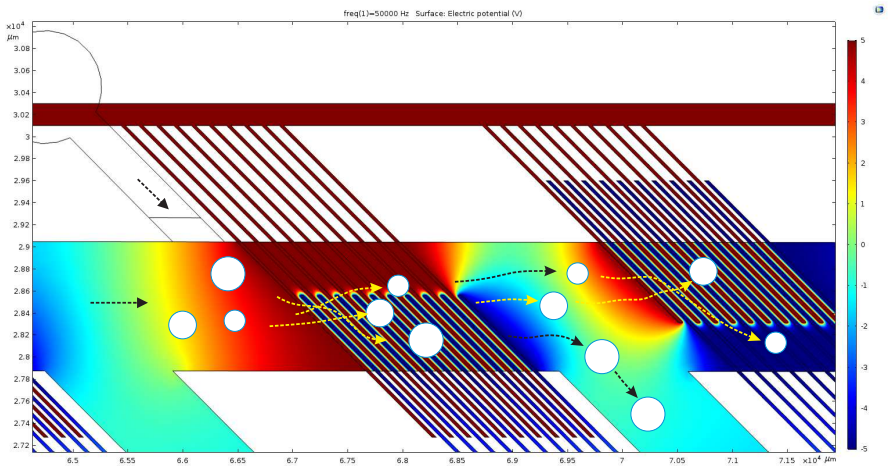


Fig. 5. Particle separation scheme in DEP chambers. One big and two small particles are separated in the first DEP region by pushing down the big one into the drain channel where the electric field is strong while the smaller particles flow towards the next DEP region following the weaker electric field. Then in the next step, the smallest particle is pushed into the weak electric field region in the second DEP chamber while the other small particle flows through the base-channel towards the next DEP region. See Section 2.1 for the theory explanation of DEP based separation.

Simulation of SN P systems rules in the μfluidic P systems. A detailed simulation of the spiking rules in terms of μfluidic P systems and its function for particle separations are explained in Fig. 6. The rules of neurons with labels t and f of the nth layer of the system structure depicted in Fig. 3 are simulated as described in Fig. 6 A) and B). Whereas spikes a, a^2 and a^3 are expressed by data-particles dp_1, dp_2, and dp_3 with size 1, 2, and 3, respectively. The spike a^4 is composed of data-particles dp_1 and dp_3, with size 1 and 3, and spike a^5 is composed of data-particles dp_2 and dp_3 with size 2 and 3, respectively.

As long as the AC voltage applies, simulations of the spiking rules are performed by separating the particles using DEP force in the electric field regions. Fig. 6 A) depicts the simulation of the rules in neuron with label t: rule $a^3 \to \lambda$ is done as particle dp_3 is pushed into the drain channel, rule $a^4 \to a$ is done as particle dp_3 is pushed into the drain channel while particle dp_1 flows along the base-channel towards the next DEP chamber. Rule $a^5 \to \lambda$ is simulated as both particles dp_2 and dp_3 are thrushed into drain channel. First two rules $a \to a$ and $a^2 \to a^2$ are used in placing spike variables of a clause in the computation unit, which is simulated by just flowing the data-particles dp_1 and dp_2 towards the next DEP chamber along the base-channel, because no DEP force effect is applied to the particles as well as to the medium since no AC voltage has been plugged-in yet.

Rules in neuron with the label 2 are simulated in a method similar to the rules of neuron with the label t as it has illustrated so in Fig. 6 D).

One might have already observed from Fig. 6 that not all spiking rules of the neurons are simulated in the same way as in the microfluidic channels. For instance, one can see that rules in neuron with label f_0 are simulated differently from similar rules of neuron t.

Rules $a^2 \to a$ and $a^6 \to a$ in neuron f_0 are simulated in a chamber of type B as sending data-particle dp_2, but not dp_1, towards the next level.

The rule $a^5 \to \lambda$ is simulated by expelling two particles dp_1 and dp_4 into the drain channel in the following two stages: firstly, the bigger particle, dp_4, is separated and pushed into the drain channel while the smaller particle, dp_1, keeps on going along the base-channel towards the next region, but it is drained too in the next stage. Therefore, the simulation has been completed.

One can see from Fig. 5 that the electrodes in the two DEP regions are positioned opposite of each other. After passing the first DEP region, the bigger particle is attracted to the area with stronger electric field close to the drain channel and then drained while the smaller particle is expelled to the base-channel because of the weak electric field. Consequently, the small particle arrives in the next DEP region, which again gets expelled to the weak electric field area, which eventually streams down into the drain channel. It is worth noting that each DEP region has its own functions for particle separation which performs the task independently from each other.

In Fig. 6 C), a computation step of the layer $n + 1$ of SN P systems model has been simulated, where the translation of the spiking rules is similar with one presented in Fig. 6 B).

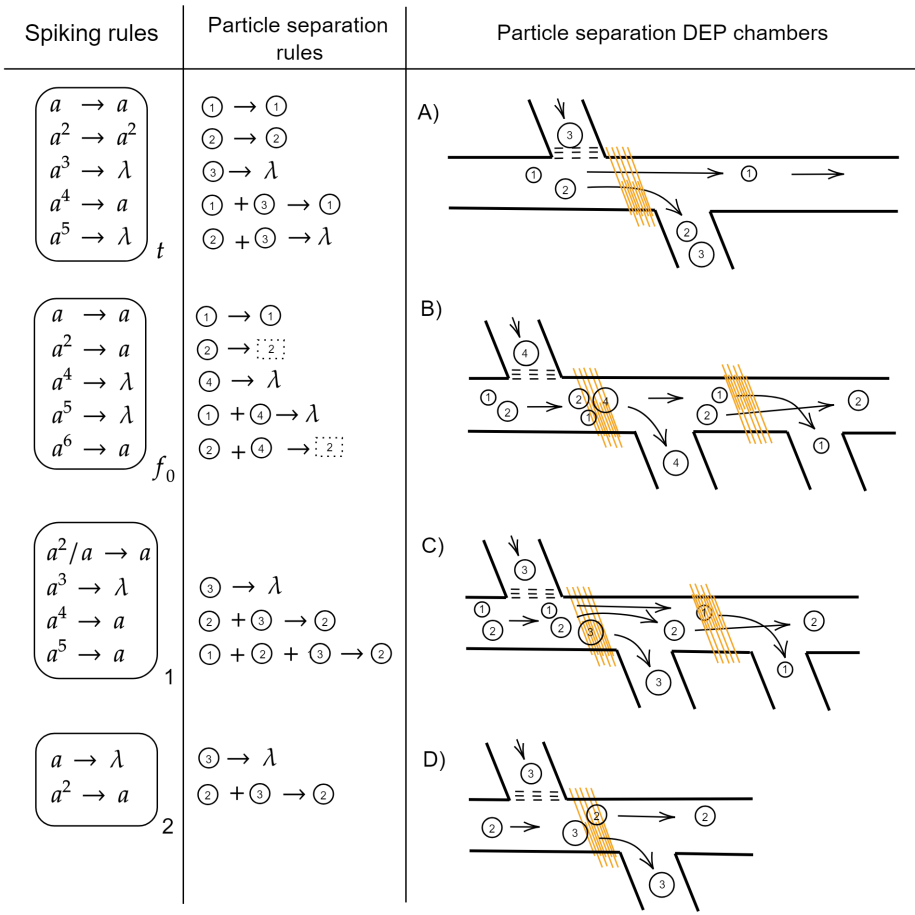


Fig. 6. Spiking rules Vs. Scheme for particle separation.

In the SN P systems model, we finally check if there exists any assignments that satisfy the Boolean formula by counting the number of spikes accumulated in the neurons with the label 3 of layer $n + 3$.

If the counted number of spikes is equal to the number of clauses of the Boolean formula, then a spike is sent to the output neuron, signaling `YES` solution.

To do it in our μfluidic computing system, we design a *decision chamber* at the end of each base-channel which has two valves placed on the connections to the Outlet and the drain channels, respectively. A particle-counter function executes in the *decision chamber*. If the amount of accumulated data-particles satisfy a certain *condition*, then the *valve* to the Outlet opens yielding data-particles flow towards the Outlet. Otherwise, the data-particles are wasted into the drain channel as the second *valve* opens. This process is applied to all decision-chambers of DOM.

At the end of the microfluidic computation, one detects whether if any particles appear in the *detection region* or not. If at least one particle is detected this would mean that we have computed a solution for the problem and so, a `YES` signal is sent out to the Outlet. If not, a `NO` solution is concluded as the result of the computation.

4 Modelling and *in silico* experiment

4.1 Fluidic network architecture and geometrical design

As described in Section 3.1, the overall architecture of the μfluidic P system is composed of three main modules: IDM, PDCoM and DOM.

The single phase microfluidic system does not change its state during thermodynamic changes and shows the high suitability for DEP application due to its ability to control pressure within channel.

Fluidic laminar flow in the Input-Duplication (IDM) part and the Decision-Detection-Output (DOM) part of the system are mainly controlled by the hydrodynamic mechanism. In general, the hydrodynamic force influences the fluid flow and force distribution within the microfluidic network, see Section 2.1. A geometrical design with a canyon-like shape, positioned in the middle of the microchannels of the IDM, improves the particle separation efficiency by splitting the flow symmetrically toward the bifurcation branches, see Fig. 7 B.

PDCoM is the main functioning block of the system, where DEP force is employed for the computation of sorting and separation of data-particles by their intrinsic characteristics. In total, 2^{n-1} number of pair base-channels and the same number of drain channels are placed in the middle of two base-channels, connecting them by an equal numbered, symmetric downstream microchannels - constructing the computation block.

Moreover, $m + 2$ pieces of *control units/inlets* with *valves* on the junctions are connected to each base-channel. The control units are devoted to providing control-particles of types *true* and *false* logical values into the DEP computation chambers.

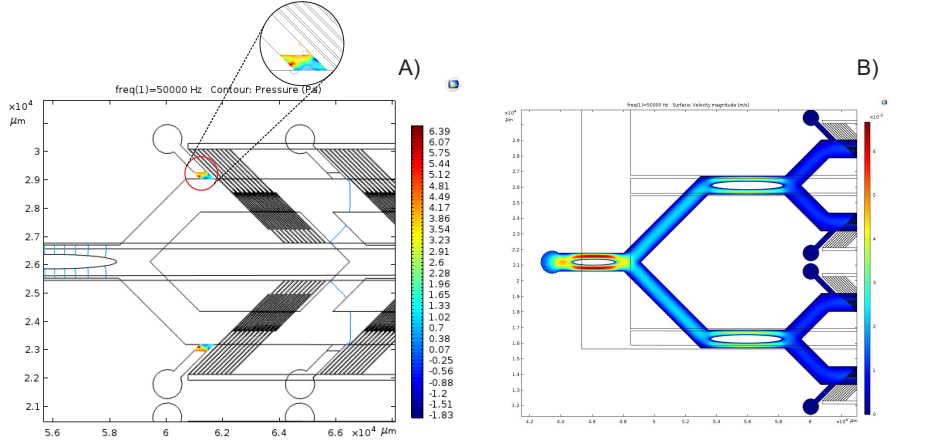


Fig. 7. A) *On/Off* valves. B) Fluid flow velocity magnitude.

The valves of PDCoM function prevents pressure leak out of the control units into the base-channels - the fluid flow should not be affected because of side pressure.

The *valves* have a periodic *On/Off* switch property. When the data-particles approach the valves, the valves are opened for a moment and releases the control-particles into the base-channels.

The valves that we considered in the microfluidic system are designed as the valve region is filled in with liquid, see Fig. 7 A. When the valve switches to the *Off* mode, *the density* and *the viscosity* of the liquid increase enough to prevent single particles and pressure from possibly leaking through the valve region to the base-channels. On the contrary, by decreasing the valve region's *density* and *viscosity*, the valve is switched to the *On* mode, then the particles from the control units enter into the base-channels.

All the valves connected to the base-channels alter between *On* and *Off* modes, periodically. According to the SN P system model depicted in Fig. 3 and its microfluidic design in Fig. 2, each control unit of the first two levels pump a control-particle of type dp_4 simultaneously into the base-channels at the time t_1 then the valves are switched to the *On* mode for a while releasing the particles, then switch back to *Off*. At the moment of t_2, the valves of the 3rd level switch to the *On* mode which allow control-particles of type dp_3 to be released. The valves of the 1st, 2nd, and 4th levels are switched to the *On* mode at the moment of t_3, then releases the corresponding control-particles of either dp_3 or dp_4 types. The valves of the 3rd level and 4th level are switched to the *On* mode at the moment of t_4 in order to release the necessary control-particles. The valves of the fourth level are again flushed at the moments t_5 and t_6, respectively, then execute its function. In other words, the main computation of the

PDCoM part of the system took place at the time series t_1, t_2, $\cdots$, t_6, parallel in each base-channel.

To implement the above mentioned function of the PDCoM, time dependent periodic functions of types rectangle $f(x) = f(x+T)$ and sin $y = Asin(\varphi t + \pi k)$ have been utilized.

When a bunch of the data-particles and the control-particles pass through a DEP chamber, those non-uniform electric field and frequency affect the particles' private physical characteristics and then cause them to separate, see Fig. 5 and Section 2.1.

The architecture scheme of the final part of the system (DOM) is a composition of five basic parts, namely 1) the decision chambers on the base-channels; 2) a couple of valves attached to each decision chamber; 3) the waste-chambers at the end of each drain-channel; 4) the detection area; 5) the Outlet, see Fig. 2.

The main function of the DOM is to decide whether or not to send a signal to the detection region depending on the PDCoM computation result. The particles in each decision chamber accumulated from a base-channel are counted. If the total amount of particles satisfies a certain condition such as being equal to the number clauses of considered propositional formula, then the valve to the *detection area* is opened, allowing the particles flow forward. Otherwise, the valve to the waste chamber opens to thrush all the particles.

The valve type used in the DOM is different from the one used in the PDCoM part. The functioning scheme of the accumulators and the end valves can be illustrated as follows:

Initial states: *valveOk.Off* and *valveNo.Off*.
`if` *acc(amount of particles)* $\geq$ *condition* `and` *time* $\geq$ *Time*
`then` *valveOk.On* `and` *the accumulated particles flow forward*,
`otherwise` *valveNo.On* `and` *thrush the accumulated particles*.

Finally, one checks the detection area in case it contains any particle to be sent to the Outlet.

In the end, the computation halts and the device is reset fully as the fluid is sucked out of the waste-chambers, as well as the Outlet, which renders the system to be ready for the next computation to proceed, Fig. 4 B.

In the next section, a computer model for the μfluidic P systems and its simulation experiments will be introduced.

4.2 Computer simulation and analysis

The 2D microfluidic architecture of the μfluidic P system was created by using AutoCAD 2017 (Autodesk Inc., USA). The design geometry was imported into COMSOL. The μfluidic system was simulated with COMSOL Multiphysics 5.3 (COMSOL Inc., Palo Alto, USA), using AC/DC (Electric Currents) module and Fluid Flow (Laminar Flow, Particle Tracing for Fluid Flow) module. The reader is referred to Section 2.1 for physical meaning of these modules.

The steady state Navier-Stokes model has been used, where the fluid inside the microchannels was simulated as liquid (non-Newtonian fluid). The model

	Electrical properties			Shell		
	size, μm	conduct., s/m	permittivity	thickness, nm	conduct., S/m	permittivity
ParticleX	2	0.25	50	8	1e-6	4

	Fluidic properties		
	density, kg/m^3	dynamic viscosity, $Pa * s$	medium conductivity, S/m
Liquid	1000	1e-3	55

Table 1. Particle and Fluid properties. Four different particles, ParticleX, $0 \leq X \leq 4$, are used.

particles are chosen so as to have dynamic viscosity of particles suspended in an aqueous solution.

The dimension of the particles was set to range from 0μm to 4μ. Missing Boolean variable x_j in the propositional formula is represented by an empty particle $dp0$ meaning that no particle is carried in the corresponding time slots. Simulation was done with parameters of electric potential in the range 2V-5V, frequency between 50kHz-100kHz, and fluid velocity 100μm/s-800μm/s.

The initial configuration of the μfluidic P system is set up as illustrated in Table 1 and Fig. 9. The particles, fluidic, and electric properties can vary in order to adapt the device functionality and task dimension.

In the simulations, Stationary, Frequency domain, and Time dependent analysis have been used. As mentioned above, the control valves alter *On/Off* modes

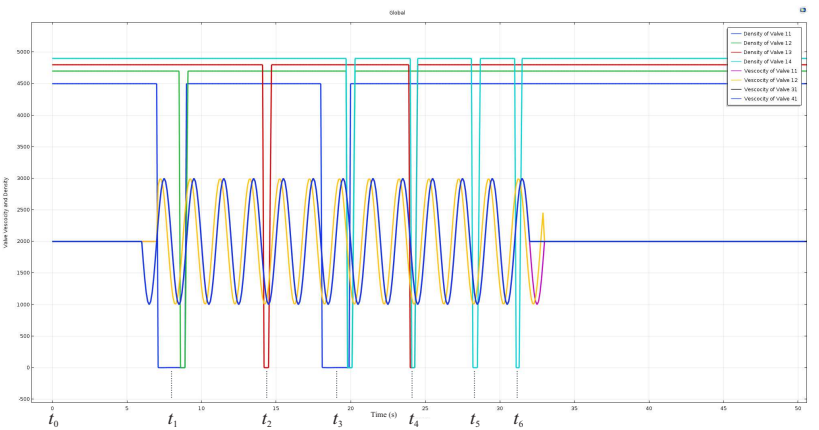


Fig. 8. Time dependent periodic control of *On/Off* type valves which governs the fluid density and viscosity.

periodically to release the control-particles into the base-channels. As an example, a diagram in Fig. 8 shows the oscillation of the control valves' periodic

switching: at time t_1, $valve_1$ and $valve_2$ switch to the *On* mode and at time t_2, $valve_3$; at time t_3, $valve_1$, $valve_2$, and $valve_4$; at time t_4, $valve_3$ and $valve_4$; at time t_5 and t_6, $valve_4$, respectively, change the state to the *On* mode. Note that a valve keeps its *On* state only for a moment to release a particle and switches back to *Off* mode immediately, preventing extra leak.

The channel fluid flow and particle tracing studies were computed and from the results, the model surface plots for the velocity magnitude, pressure distribution, and electric gradient have been generated as depicted in Fig. 9.

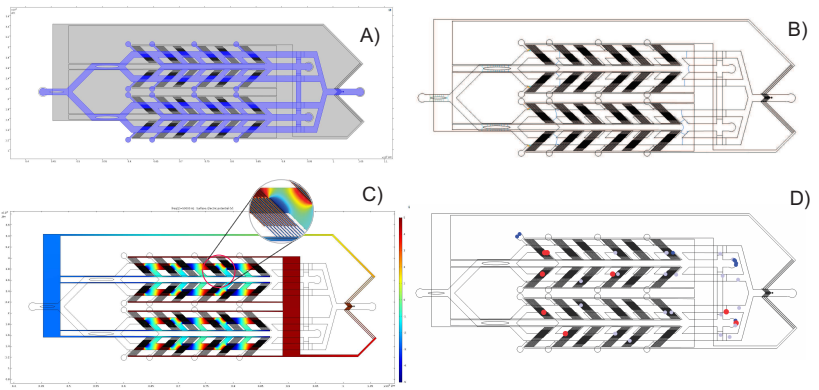


Fig. 9. Simulation of μfluidic P system: A) Fluid flow and particle tracing domain, B) Pressure distribution throughout the microchannel network, C) Electric current conservation domain and electric field gradient, D) Particle trajectory.

Depending on the amount of particles accumulated in the decision region, either wall, *valveOk* or *valveNo*, switches to the *On* mode from *Off* allowing particles to pass through the trough for the further flow, see Fig. 10.

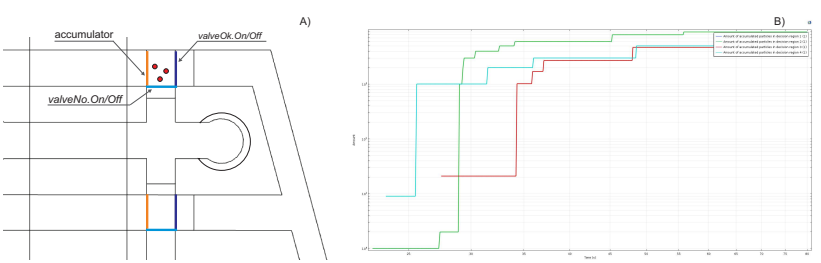


Fig. 10. Decision region of DOM. A) The accumulator and the wall valves, B) Counted particles in the decision regions

The μfluidic P systems model simulation has been introduced with certain parameters.

5 Final Remarks

The μfluidic P systems for parallel and distributed computing device design and its corresponding computation model have been introduced in the present work. Computer simulations *in silico* experiments have been carried out for different parameters. Further in-depth elaborations of the device design and parameter estimations of computer simulations have to be calibrated for physics and biological requirements, as well as the mission of the computing system functioning.

In the next stages of the research, device fabrication of the μfluidic P systems as bio-chips should be considered.

Acknowledgments

This work has been supported by the Mongolian Foundation for Science and Technology (Research Grants ShUSS-2018/04 and MOST-MECSS2017001), and the School of Information and Communication Technology, MUST (Research Grant SICT201801).

References

1. Ali, H., and Park, C. Numerical study on the complete blood cell sorting using particle tracing and dielectrophoresis in a microfluidic device. *Korea-Australia Rheology Journal 28*, 4 (Aug 2016), 327–339.
2. Ciobanu, G., Perez-Jimenez, M. J., and (Eds.), G. P. *Applications of Membrane Computing*, 1 ed., vol. 25 of *Natural Computing Series*. Springer-Verlag Berlin Heidelberg, Jul 2006. Part of the Emergence, Complexity and Computation book series (ECC, volume 25).
3. Ciprian, I., Liming, Y., Francis, E. H. T., and Chen, B. Bidirectional field-flow particle separation method in a dielectrophoretic chip with 3D electrodes. *Sensors and Actuators B: Chemical 129*, 1 (2008), 491–496.
4. Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1 ed. Books in the Mathematical Sciences. W. H. Freeman, Jan 1979.
5. Green, N. G., and Morgan, H. Dielectrophoresis of Submicrometer Latex Spheres. 1. Experimental Results. *The Journal of Physical Chemistry B 4*, 103 (Jan 1999), 41–50.
6. Haiming, C., Mihai, I., and Tseren-Onolt, I. On the Efficiency of Spiking Neural P Systems. *Proceedings of 8th International Conference on Electronics, Information, and Communication (ICEIC 2006), Ulaanbaatar, Mongolia* (Jun 2006), 49–52.
7. Huang, Y., Holzel, R., Pethig, R., and Wang, X.-B. Differences in the AC electrodynamics of viable and non-viable yeast cells determined through combined dielectrophoresis and electrorotation studies. *Physics in Medicine and Biology 37*, 7 (1992), 1499.

8. Huang, Y., Joo, S., Duhon, M., Heller, M., Wallace, B., and Xu, X. Dielectrophoretic Cell Separation and Gene Expression Profiling on Microelectronic Chip Arrays. *Analytical Chemistry 74*, 14 (2002), 3362–3371.
9. Hughes, M. P. Strategies for dielectrophoretic separation in laboratory-on-a-chip systems. *Electrophoresis 4*, 16 (Aug 2002), 2569–2582.
10. Ishdorj, T.-O., and Leporati, A. Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources. *Natural Computing 7*, 4 (Jul 2008), 519–534.
11. Ishdorj, T.-O., Leporati, A., Pan, L., Zeng, X., and Zhang, X. Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources. *Theoretical Computer Science 411*, 25 (May 2010), 2345–2358.
12. King, P., Corsi, J., Pan, B., Morgan, H., de Planque, M., and Zauner, K. Towards molecular computing: Co-development of microfluidic devices and chemical reaction media. *Biosystems 109*, 1 (Jul 2012), 18–23.
13. Ledesma, L., Manrique, D., and Rodríguez-Patón, A. A tissue P system and a DNA microfluidic device for solving the shortest common superstring problem. *Soft Computing 9*, 9 (Sep 2005), 679–685.
14. Lee, S.-W., Kim, Y.-W., and Kim, Y.-K. Electrical Insulation and Dielectric Phenomena. In *IEEE Annual Report* (San Francisco, California, USA, Oct 1996), pp. 241–244.
15. Mihai, I., Gheorghe, P., and Takashi, Y. Spiking Neural P Systems. *Fundamenta Informaticae 71*, 2 (2006), 279–308.
16. Mihai., I., Păun, A., Gheorghe, P., and Pérez-Jiménez, M. J. Computing with Spiking Neural P Systems: Traces and Small Universal Systems. In: Mao C. and Yokomori T. (Eds) DNA Computing. DNA 2006. Lecture Notes in Computer Science. *Springer, Berlin, Heidelberg 4287*, 16 (2006), 1–16.
17. Pan, L., Păun, G., G, G. Z., and Neri, F. Spiking neural P systems with communication on request. *International Journal of neural systems 27*, 8 (2017).
18. Pan, L., Song, B., Valencia-Cabrera, L., , and rez Jimé nez, M. J. P. The Computational Complexity of Tissue P Systems with Evolutional Symport/Antiport Rules. *Complexity 2018*, 3745210 (May 2018), 21.
19. Pan, L., Wu, T., Su, Y., and Vasilakos, A. V. Cell-Like spiking neural P systems with request rules. *IEEE Transactions on Nanobioscience 16*, 6 (2017), 513–522.
20. Pohl, H. A. The Motion and Precipitation of Suspensoids in Divergent Electric Fields. *Journal of Applied Physics 22*, 7 (2004), 869.
21. Satoshi, T., and Hitoshi, W. Dielectrophoresis of microbioparticles in water with planar and capillary quadrupole electrodes. *IEE Proceedings - Nanobiotechnology 150*, 2 (Nov 2003), 59–65.
22. Satoshi, T., Katsuyuki, Y., and Hitoshi, W. Flow Fractionation of Microparticles under a Dielectrophoretic Field in a Quadrupole Electrode Capillary. *Analytical Chemistry 73*, 23 (2001), 5661–5668. PMID: 11774905.
23. Song, B., Hu, Y., Adorna, H. N., and Xu, F. A Quick Survey of Tissue-Like P Systems. *Romanian Journal of Information Science and Technology 21*, 3 (2018), 310–321.
24. Song, B., Song, T., and 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 (2017), 17–32.
25. Song, B., Zhang, C., and Pan, L. Tissue-like P systems with evolutional symport/antiport rules. *Information Sciences 378*, C (Feb 2017), 177–193.

26. Wu, T., Bîlbîe, F.-D., Păun, A., Pan, L., and Neri, F. Simplified and Yet Turing Universal Spiking Neural P Systems with Communication on Request. *International Journal of Neural Systems 28*, 8 (May 2018).
27. Wu, T., Păun, A., Zhang, Z., and Pan, L. Spiking Neural P Systems With Polarizations. *IEEE Transactions on Neural Networks and Learning Systems 29*, 8 (Aug 2018), 3349–3360.
28. Wu, T., Zhanga, Z., Păun, G., and Pan, L. Cell-like spiking neural P systems. *Theoretical Computer Science 623* (Apr 2016), 180–189.
29. Youlan, L., Colin, D., H.John, C., Gregory, N., and Kaler, K. V. I. S. Continuous dielectrophoretic cell separation microfluidic device. *Lab Chip 7*, 2 (2007), 239–248.
30. Zhang, G., Perez-Jimnez, M. J., and Gheorghe, M. *Real-life Applications with Membrane Computing*, 1 ed., vol. 25. Springer International Publishing AG, Jul 2017. Part of the Emergence, Complexity and Computation (ECC) book series.

Improved initialization method for K-means algorithm optimized by Tissue-like P system

Shaolin Wang[1], Xiyu Liu[2,*], and Laisheng Xiang[3]

Shandong Normal University, Jinan, Shandong, China
wangshaolin1116@163.com;sdxyliu@163.com;xls3366@163.com

Abstract. The K-means algorithm gets widely used due to its simplicity and effectiveness. But it is sensitive to the selection of initial cluster centers. In this paper, we proposed a initialization method to select initial clustering centers for K-means algorithm. Furthermore, because of the boundedness of the initialization method, we modified it and designed a Tissue-like P system to realize the new method. The experiments are operated on five UCI datasets, the results proved that the designed Tissue-like P system based on the new initialization method is effective.

Keywords: Initialization, K-means, Tissue-like P system

1 Instruction

Clustering becoming more and more popular nowadays because it has shown its usefulness in several fields like industry, business and processing. The K-means algorithm, which is a nonhierarchical clustering algorithm,gets widely used due to its simplicity and effectiveness[1]. But it is sensitive to the selection of initial cluster centers, it gets trapped in local optimum easily when serval bad initial cluster centers have been selected. Many researchers are devoted in the initialization of K-means algorithm.

Khan and Ahmad[2] proposed the cluster center initialization algorithm (CCIA) to solve the cluster initialization problem. It initiates by calculating the mean and the standard deviation for data attributes and then separates the data with a normal curve into a certain partition. Redmond and Heneghan[3], first constructed a kd-tree of the points to perform density estimation and then used a modified maximin method to select K centers from densely populated leaf buckets. In the FSDP[4] method, the density of a point is estimated by a gaussian kernel, whereas the computation complexity will increase correspondingly due to the fact that the gaussian kernel is a nonlinear function. Arthur and Vassilvitskii[5] proposed the k-means++ approach. In k-means++, the point will be chosen with the probability proportional to the minimum distance of this point from already chosen seeds. Due to the random selection of the first seed and the lack of the density definition of points, different runs have to be performed to obtain good clustering. Bai, Liang, Dang, and Cao[6] integrated the distance and the density together to select initial cluster centers. They used the total

distance between an object and all objects from a data set as the density of the object. Khan and Ahmad[7] proposed a new method for selecting the most relevant attributes, namely Prominent attributes, compare it with another existing method to find Significant attributes for unsupervised learning, and perform multiple clustering of data to find initial cluster centers. The proposed algorithm ensures fixed initial cluster centers and thus repeatable clustering results.

In this paper, we proposed a initialization method for K-means and designed a Tissue-like P system for the modified initialization method. Five UCI[8] data sets are used to evaluate the performance of the proposed method and system.

The article is organized as follows. Section 2 gives the preliminaries. Section 3 presents the proposed new initialization method for K-means. Section 4 shows the Tissue-like P system we designed for the modified initialization method. Section 5 provides an experimental validation of the proposed algorithm. Section 6 draws the conclusions.

2 Preliminary for Tissue-like P System

2.1 Tissue-like P System

Membrane computing is a new branch of natural computing, which derived from the construct and functions of cells or tissues. The membrane are arranged as a hierarchical structure and operates in a parallel way thus can reduce the time complexity greatly.

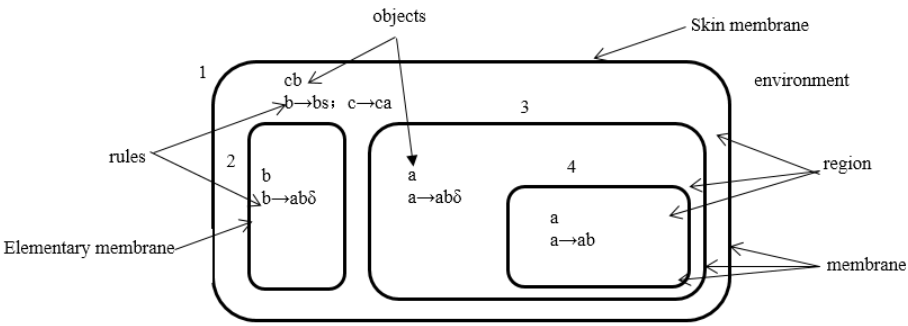


Fig. 1. A basic membrane structure

As depicted in Fig.1, a membrane m with no upper neighbor is called a skin membrane and a membrane m with no lower neighbor is called elementary membrane. The membranes always have objects and rules with it. The whole system is divided into different regions by the membranes. The space outside the skin membrane is called environment. A region is either a space delimited by an

elementary membrane or a space delimited by non-elementary membranes. In the membrane, there are some rules and objects to execution the algorithm.

Formally, The Tissue-like P system(of degree q>0) with symport/antiport rules is a construct

$$\Pi = (O, \omega_1, \ldots, \omega_q, R_1, \ldots, R_q, R^{'}, i_0) \tag{1}$$

where

(1) O is a finite alphabet, whose symbols are called objects;

(2) $\omega_i (1 \leq i \leq q)$ is finite set of strings over O, which represents multiset of objects initially present in cell i;

(3) $R_i (1 \leq i \leq q)$ is finite set of evolution rules in cell i;

(4) $R^{'}$ is finite set of communication rules of the form $(i, u/v, j)$, which represents communication rule between cell i and cell j, $i \neq j$, $i, j = 1, 2, \ldots, q, u, v \in O^*$;

(5) i_0 indicates the output region of the system.

As usual in the framework of membrane computing,every cell in tissue-like P systems,works as a computing unit in a maximally parallel way. In a computing step, each object in a cell can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can participate in a rule of any form must do it, i.e, in each step we can apply a maximal set of rules.

A computation in a tissue-like P system of degree q is a sequence of steps which start with the cells $1, \ldots, q$ containing the multisets $w_1, \ldots, w_q$ and where, in each step, one or more rules are applied to the current multisets of symbol objects. A computation is successful if and only if it halts. When it halts, it produces a final result in output cell.

3 A Initialization Method For K-means Algorithm

In this section, we present the method for selecting the initial cluster centers in detail. To begin with, several definitions are described.

The average distance of the data set X within n data points is defined as

$$AVD(X) = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=j+1}^{n} d(X_i, X_j) \tag{2}$$

where $d(X_i, X_j)$ is the Euclidean distance between data point X_i and data point X_j.

The density of the data point X_i in data set X is defined as:

$$\rho(X_i) = \sum_{j=1}^{n} f[d(X_i, X_j) - AVD(X)] \tag{3}$$

where $f(x) = 1$ if $x < 0$, otherwise $f(x) = 0$.

The average distance between the data point X_i and its neighborhoods is defined as

$$D(X_i) = \frac{1}{\rho_{(X_i)}} \sum_{X_j \in N_i} d(X_i, X_j) \tag{4}$$

where $N_i = \{X_j | 0 < |X_i - X_j| \leq AVD(X)\}$.

$S(X_i)$ which represents the distance of the data point X_i is defined as $\min\{d(X_i, X_j)\}$ if existing X_j satisfy $\rho(X_j) > \rho(X_i)$. Otherwise, it is defined as $\max\{d(X_i, X_j)\}$, that is

$$S(X_i) = \begin{cases} \min\limits_{j:\rho(X_j)>\rho(X_i)} \{d(X_i, X_j)\}, & \exists j, \rho(X_j) > \rho(X_i) \\ \max\{d(X_i, X_j)\}, & \text{otherwise} \end{cases} \tag{5}$$

The possibility of the data point is a clustering center is defined as

$$P(X_i) = \rho(X_i) * \frac{1}{D(X_i)} * S(X_i) \tag{6}$$

The possibility $P(X_i)$ is not limited to $[0, 1]$, it's a value represents the data point is actually a clustering center to what extent. The value of $\rho(X_i)$ represents the quantity of the data points around data point X_i. $D(X_i)$ represents the tightness and degree of concentration to potential clustering center X_i, the smaller the value $D(X_i)$, the tighter and more concentrated the neighbors around the clustering center. $S(X_i)$ represents the dissimilarity between the two clusters. To sum up, the greater value the $P(X_i)$, the data point X_i more likely to be a clustering center.

To improve the quality of clustering results of K-means algorithm, we proposed a new selection method of initial clustering centers. Firstly, given the data set X and clustering number k, the average distance of the data set X is calculated according to equation (3) and the value of $\rho(X_i), D(X_i), S(X_i)$ of each data point in data set X is calculated according to equation (4)(5)(6) in sequence. Then, the k data points which have larger value of $P(X_i)$ than the remaining data points are set clustering centers.

4 The Designed Tissue-like P system For K-means With Improved Initialization Method

In section 3,we proposed a new initial clustering centers determining method for K-means. However, the quality of clustering results with the improved K-means algorithm is mainly depend on the value of average distance of the data set. The data points that belongs to different clusters will be grouped into the same cluster if the value of average distance becomes larger than former value. The data points will be divided into several different clusters if the value is too small. Thus, the value is essential to the quality of the clustering results. Also, the value is difficult to determined and to solve this problem, we designed a P system to execute the improved K-means algorithm.

4.1 A Tissue-like P system designed

As depicted in Fig.2, the P system contains $q+1$ cells in total. The cell which labeled by 1 is input cell, it stores the data objects and delivers them to cells which labeled 2 to q. These $q-1$ cells are called initialization cell which will run independently, they are used to select the initial objects by the proposed method and passing them to cell which labeled $q+1$. The cell $q+1$ executes K-means algorithm with objects passed by cell 2 to q, then pass the best object to the output region which labeled by 0, that is, the environment in Tissue-like P system.

In the initialization cell 2 to q, the initialization method are modified. we set a multiplier λ changes between 0.1 to 10 except 1 for average distance. For each initialization cell, the value of multiplier λ are determined by two steps. Firstly, the value of multiplier λ are chosen in interval $[0.1, 1)$ or $(1, 10]$ by a equal possibility. Then the specific value are determined randomly. Thus, the system can produce a more robust results compared to original method.

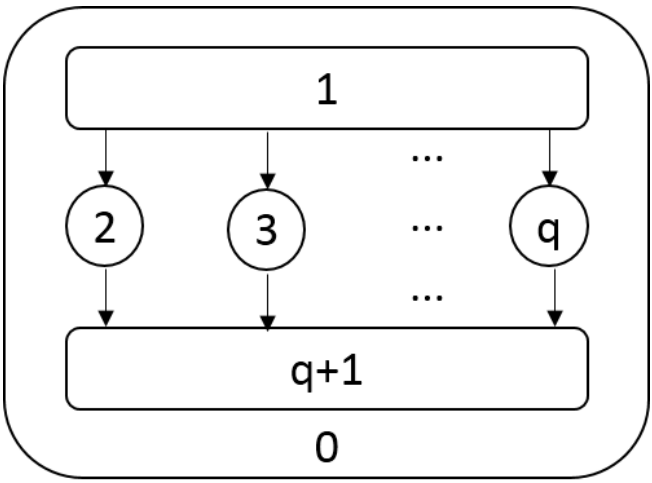


Fig. 2. The loop structure of designed P system

4.2 Objects

Each cell in the Tissue-like P system contains a number of objects. Cell 1 has n objects, each object is considered as a d dimensional vector with the form

$$z_i = z_{i1}, z_{i2}, \ldots, z_{id} \tag{7}$$

where d are number of dimensions the object has. i represents the ith object in cell 1 and $i = 1, \ldots, n$.

For objects in cell 2 to $q+1$ and the environment, each object is considered as a $(k*d)$ dimensional vector. Each object is denoted as the following form

$$Z = (z_{11}, \ldots, z_{1d}, \ldots, z_{i1}, \ldots, z_{id}, \ldots, z_{k1}, \ldots, z_{kd}) \tag{8}$$

where k are number of the cluster centers and $1 \leq k \leq n$. According to equation (8), that is

$$Z = (z_1, \ldots, z_i, \ldots, z_k) \tag{9}$$

The initial object Z generates in the initialization cell according to the objects within cell 1 by the proposed method. Each initialization cell generates one object thus q initialization cells generate q objects. Initialization cells pass the q objects to the cell $q+1$, then cell $q+1$ selects the best object from the q objects and delivers it to environment.

4.3 Rules

Except for object, there are two types of rules in the Tissue-like P system: evolution rules and communication rules. The evolution rules aim to evolve the objects in cells and the communication rules aim to exchange and share the objects. Evolution rules are used to evolve the objects associated with cluster centers, so the Tissue-like P system is able to find the optimal cluster centers for a data set via the evolution of objects. Moreover, communication rules will realize the exchange and sharing of better objects between adjacent cells.

Evolution rules Evolution rules is to evolve the objects in cells to generate new objects used in next computing step. For the designed Tissue-like P system, the initialization cells is used to generate objects Z which represent a set of cluster centers. The evolution rules associated with initialization cells are:

(1)$r_1 = \{z_1 z_2 \ldots z_n \rightarrow z_i^1 z_i^2 \ldots z_i^k, 1 \leq i \leq n\}$. Rule r_1 express that initialization cells select k objects by the modified method. Because of the different value of λ, the k objects maybe different.

(2)$r_2 = \{z_i^1 z_i^2 \ldots z_i^k \rightarrow Z, 1 \leq i \leq n\}$. Rule r_2 express that initialization cells generate a new object Z represent cluster centers.

The evolution rule within the cell $q+1$ is:

$r_3 = \{Z^1, \ldots, Z^q \rightarrow Z^j, 1 \leq j \leq q\}$. The rule express that cell $q+1$ select the best object from previous q cells.

Communication rules Communication rules are used to exchange the objects between each cell and update the best object found so far in the environment. In the designed Tissue-like P system, cell 1 to q pass the object to its following cell.

The communication rules are:

(1) $(1, z/\lambda, i), i = 2, ..., q$. The rule expresses that object z is communicated from cell 1 to cell 2-q.

(2) $(i, Z/\lambda, q+1), i = 2, ..., q$. The rule expresses that object Z is communicated from cell 2-q to cell $q+1$.

(3) $(q+1, Z/\lambda, 0)$, The rule expresses that object Z is communicated from cell $q+1$ to the environment.

where λ represents no object will be transport inversely.

4.4 Halting and output

The designed Tissue-like P system is single-track and cell can only passes the object to its following part. So when the environment is not null anymore, that is, cell $q+1$ has delivered a object to the environment, the system halts. The object stored in the environment will be regard as the best object, which is same as optimal cluster centers.

5 Experiment Analysis

5.1 Data Set Description

We tested the proposed algorithm on five data sets from the UCI, including the Wine, Glass, Haberman, Soybean-small and Zoo. The detailed descriptions are in Table 1.

Table 1. Descriptions of the 5 Data Sets

Data Set	Number of Data Points	Number of Attributes	Number of Classes
Wine	178	13	3
Glass	214	9	6
Haberman	306	3	2
Syebean-small	47	35	4
Zoo	101	16	7

5.2 Experimental results

In order to check the influence of different numbers of initialization cells in the Tissue-like P system. We applied the system within different numbers of initialization cells on the five datasets. The numbers of initialization cells in three P systems are set to 4 cell, 8 cells, 16 cells. Because of the stochastic mechanism existed in the P system, we run the algorithm for 50 times with different numbers of cells independently. We use the number of correct points to represent the clustering quality. The mean values and standard deviations of the 50 runs are calculated to represent the average clustering quality and robustness of the P systems. Table 2 shows the performance of Tissue-like P

systems with different numbers of initialization cells. We can observe that the Tissue-like P system with 16 initialization cells can always gets a higher mean value and a higher value of standard deviation compared to the other two Tissue-like P systems. In fact, the Tissue-like P system with 16 initialization cells gets the largest mean value and smallest standard deviation, that is, the Tissue-like P system with 16 initialization cells produce a pretty good clustering result with high clustering quality and robustness.

Table 2. The performance of P systems with different numbers of initialization cells

Data Set	4 cells	8 cells	16 cells
Wine	127.96±1.7545	128.9±0.9000	129.12±0.5879
Glass	187.14±6.8819	188.36±4.0731	188.94±0.4200
Haberman	170.82±1.5961	171.60±1.6248	172.30±2.1932
Syebean-small	41.72±4.0102	43.66±2.8608	45.50±2.2913
Zoo	86.72±3.2621	88.38±1.2632	88.94±0.2375

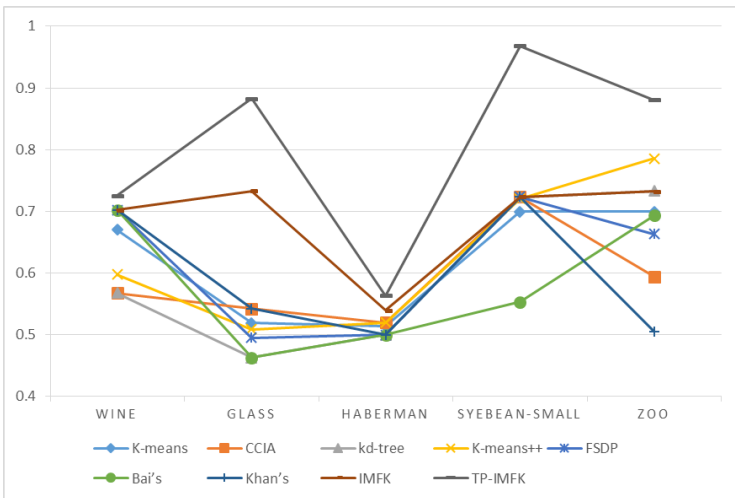


Fig. 3. The average correct rates obtained by the algorithms for 50 runs

In order to further evaluate clustering performance, the proposed initialization method for K-means (IMFK instead in Table 3) and the Tissue-like P system designed for the modified initialization method based K-means algorithm(TP-IMFK instead in Table 3) is compared with algorithms:classical k-means, CCIA,

kd-tree, K-means++, FSDP, Bai's, Khan's. We computed the average correct rates by the average correct points obtained from the 50 runs and the results of TP-IMFK in Table 3 are from the Tissue-like P system with 16 initialization cells. Table 3 gives the average correct rates obtained by the algorithms for 50 runs on the five data sets and the corresponding line graph are in Fig.3. The comparison results show that the Tissue-like P system designed for the modified initialization method based K-means algorithm provides the largest value of correct rates in compare to those of other algorithms. For data sets except Zoo, the proposed initialization method also shows good performance compared with other algorithm except the TP-IMFK.

Table 3. The average correct rates obtained by the algorithms for 50 runs

Data Set	K-means	CCIA	kd-tree	K-means++	FSDP	Bai's	Khan's	IMFK	TP-IMF
Wine	0.6704	0.5674	0.5674	0.5978	0.7022	0.7022	0.7022	0.7022	0.7254
Glass	0.5193	0.5421	0.4626	0.5086	0.4953	0.4626	0.5421	0.7330	0.8829
Haberman	0.5132	0.5196	0.5000	0.5196	0.5000	0.5000	0.5000	0.5392	0.5631
Syebean-small	0.6996	0.7234	0.7234	0.7204	0.7234	0.5532	0.7234	0.7234	0.9680
Zoo	0.6995	0.5941	0.7327	0.7855	0.6634	0.6931	0.5050	0.7326	0.8806

6 Conclusion

In the paper, we designed a Tissue-like P system with communication rules and evolution rules for the modified initialization method. The initialization method was proposed in order to improve the clustering quality of K-means algorithm. We evaluated the system with different number of initialization cells: 4 cells, 8 cells, 16 cells. Finally, we compare the system(16 cells) with the other algorithms on five UCI data sets to prove the validity of the system.

Acknowledgement

This work is supported by the Natural Science Foundation of China(nos.61472231, 61502283,61170038),Ministry of Education of Humanities and Social Science Research Project,China(12YJA630152), Social Science Fund Project of Shandong Province,China(16BGLJ06, 11CGLJ22).

References

1. Forgy E W. Cluster analysis of multivariate data : efficiency versus interpretability of classifications[J]. Biometrics, 1965, 21(3):41-52.

2. Khan S S, Ahmad A. Cluster center initialization algorithm for K -means clustering[M]. Elsevier Science Inc. 2004.
3. Redmond S J, Heneghan C. A method for initialising the K -means clustering algorithm using kd -trees[J]. Pattern Recognition Letters, 2007, 28(8):965-973.
4. Rodriguez A, Laio A. Clustering by fast search and find of density peaks[J]. Science, 2014, 344(6191):1492.
5. Arthur, David, Vassilvitskii, et al. k-means++: the advantages of careful seeding[J]. Proceedings of the Eighteenth Annual ACM-SIAM Symposiumon Discrete algorithms, Society for Industrial and Applied Mathematics, 2007, 11(6):1027-1035.
6. Bai L, Liang J, Dang C, et al. A cluster centers initialization method for clustering categorical data[J]. Expert Systems with Applications An International Journal, 2012, 39(9):8022-8029.
7. Khan S S, Ahmad A. Cluster center initialization algorithm for K-modes clustering[J]. Expert Systems with Applications, 2013, 40(18):7444-7456.
8. M. Lichman, UCI machine learning repository [http://archive. ics.uci.edu/ml], Univ. California, School of Information and Computer Science, Irvine, CA, USA, 2013.
9. Liu Xiyu, Zhao Yuzhen. Tissue P Systems with Cooperating Rules. Chinese Journal of Electronics, 2018,27(2):324-333.
10. Zhao Yuzhen, Liu Xiyu, Wang Wenping. Spiking Neural P Systems with Neuron Division and Dissolution. PLOS ONE. 2016,11(9):e0162882.
11. Liu Xiyu, Xue Jie. A Cluster Splitting Technique by Hopfield Networks and P Systems on Simplices. Neural Processing Letters, 2017:1-24.

Image Thresholding using a modified membrane-inspired algorithm based on particle swarm optimization with hyperparameter

Dequan Guo[1,2,3], Gexiang Zhang[5,4] ⋆, Yi Zhou[2], Jianying Yuan[2,1,3], Prithwineel Paul[5], Kechuang Fu[2], and Ming Zhu[2]

[1] The Postdoctoral Station at Xihua University Based on Collaboration Innovation Center of Sichuan Automotive Key Parts, Xihua University, Chengdu, 610039, China
newgdq@126.com,yuanjy@cuit.edu.cn
[2] Chengdu University of Information Technology, Chengdu 610225, China
1033161554@qq.com,kcfucx@cuit.edu.cn,zhuming@126.com
[3] Aeronautics and Astronautics, University of Electronic Science and Technology, Chengdu 610054, China
[4] Robotics Research Center, Xihua University, Chengdu, 610039, China
zhgxdylan@126.com
[5] School of Electrical Engeneering, Southwest Jiaotong University, Chengdu, 610031, China
prithwineelpaul@gmail.com

Abstract. This paper presents a novel approach for thresholding of image segmentation using a modified membrane-inspired algorithm based on particle swarm optimization (PSO) with hyperparameter, namely MIPSOH. In this approach, to solve the optimal thresholding problem in image segmentation, a cell-like P system with a specially designed membrane structure is presented and an improved PSO evolution mechanism is integrated into the cell-like P system with hyperparameter. With the merits of fast convergence of improved PSO and high parallelism of P systems, the proposed approach can achieve better segmentation results effectively and quickly. Both the qualitative and quantitative experimental results in the proposed approach verify the computational efficiency and segmentation effectiveness.

Keywords: Image segmentation; particle swarm optimization; membrane computing; P system; hyperparameter optimization; thresholding approach

1 Introduction

Image segmentation is generally one of the first stages in any attempt to analyze or interpret an image automatically. It bridges the gap between low-level image processing and high-level image processing. Low/mid-level image processing algorithms are usually designed to make the input image easier to process for the application in high-level image processing. Image segmentation is a basic pre-processing step to deal with subsequent practical problems [1]. Thresholding is widely used as a popular tool in image

⋆ Corresponding author.

segmentation, due to its simplicity and efficiency. The goal of thresholding is to separate objects from background in an image or discriminate objects from objects that have distinct gray levels [2]. The pixels with gray values greater than a certain threshold are classified as object pixels, and those with gray values lesser than the threshold are classified as background pixel [2]. Otsus method [3] and Kapurs method [4] find the optimal thresholds by maximizing the between-class variance of gray levels and the entropy of the histogram respectively. In addition, thresholding-based approaches occasionally have been used in color image segmentation [5,6], mostly for medical image processing, and their usages are often investigated on grayscale image segmentation [7]. More detailed division which roughly covers algorithms in image segmentation algorithms includes six main categories: histogram thresholding-based methods, clustering-based methods, edge detection-based methods, region-based methods, graph-based methods, and hybrid methods [1,8].

Image thresholding is an important method in image segmentation. The key point in image thresholding is to find an appropriate threshold, which is assumed to be an optimal problem in the gray levels. There are many intelligent optimal methods in this fields. PSO is an important optimal method for image segmentation. The algorithm was initially inspired by the social behavior of the birds and then used swarm intelligence to build a simplified model [9]. PSO algorithm is a kind of evolutionary algorithm like genetic algorithm (GA) and differential evolution (DE). It starts from the random solutions, and obtains the optimal solution by the iterative search. The fitness is used to evaluate the quality of the solution, because it is simpler than rules of GA, i.e., without ‘crossover‘ and ‘mutation‘ operation in GA. In PSO, the potential solutions, called particles, search the problem space by following the current optimum particles. This algorithm has gained research attention since it was proposed and is widely applied in solving practical problems [1,9,24,37,49,50].

In recent years, some new intelligent algorithms are introduced in image segmentation using membrane computing (MC). A comprehensive literature review pertaining to image segmentation using MC models has been presented in [27]. MC was initiated by Păun [12,13,14]. It focuses on the investigations of computational model, called membrane system or P system. Many researchers in the field of MC have proposed various techniques inspired by cell biology for applications. More especially, they considered cell organization in tissues, organs, and most recently, from the organization of neurons. MC is an interdisciplinary research directions and has been widely applied in the areas of computer science, biology, biomedicine, bio-informatics and several other fields such as mathematics, artificial intelligence, automation, economics. These models are abstracted from the structure and functioning of living cells, as well as from the way the cells are organized in tissues or high order structures [15]. P systems have the characteristics of distribution, parallelism and expansibility. It is also suitable for solving a variety of practical problems [17], such as engineering optimization [18,19], fault detection [20], image processing [21,22], and modeling biological and ecological systems [23].

Inspired by the the characteristic of pixels clustering excellent schema recording and parallelism of PSO, transformation or communication-like rules in P systems for each pixel is introduced to design thresholding method. The main contributions of this

paper can be summarized as follows: (1) A membrane-inspired algorithm based on particle swarm optimization with hyperparameter optimization is proposed, here we name it MIPSOH. A dynamic double one-level membrane structure (D-OLMS) with membrane division and dissolution is presented to combine with PSO to allocate the particles and execute the communications between elementary and skin membranes. (2) The proposed approach is used to obtain best thresholding in image segmentation. The solving of image thresholding process is considered as an optimal problem to maximize the variance of the different classes (or minimize the within-class) in gray levels. (3) Extensive experiments are carried out by considering maximization of between-class variance and minimization of within-class variance to verify the effectiveness and practicality of the proposed approach, and it outperforms several existing methods in terms of the computational efficiency and segmentation effect.

The organization of this article is as follows. Section 2 describes the particle swarm optimization and structure of membrane. Section 3 presents the proposed approach in detail. In Section 4, the effectiveness of algorithm is verified in the experiments. Conclusions and future works are finally drawn in Section 5.

2 Related works

This section starts with a brief introduction of PSO and the structure of membrane with OLMS.

2.1 PSO

PSO is a population-based stochastic algorithm that starts with an initial population of randomly generated particles [9]. For a search problem in a D-dimensional space, each particle keeps track of its coordinates in the problem space which are associated with the best solution ($fitness$) it has achieved so far. The fitness value is also stored. This value is called $pbest$. Another 'best' value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle neighborhood of the particle. When a particle takes all the population as its topological neighbors, the best value is a global best and is called $gbest$. During a search process, each particle is attracted by its previous best particle ($pbest$) and the global best particle ($gbest$) as follows [28,29].

$$\begin{aligned} v_{ij}\left(t+1\right) = w \cdot v_{ij}(t) + c_1 \cdot rand1_{ij} \cdot \left(pbest_{ij}(t) - x_{ij}(t)\right) \\ + c_2 \cdot rand2_{ij} \cdot \left(gbest_j(t) - x_{ij}(t)\right) \end{aligned} \tag{1}$$

$$x_{ij}\left(t+1\right) = x_{ij}\left(t\right) + v_{ij}\left(t+1\right) \tag{2}$$

where $i = 1, 2, \ldots, N$ is the particles index, N is the population size, $Xi = (xi1, xi2, ..., xiD)$ is the position of the ith particle. $V_i = (v_{i1}, v_{i2}, ..., v_{iD})$ represents the velocity of the ith particle. The $pbest_i = (pbest_{i1}, pbest_{i2}, ..., pbest_{iD})$ is the best previous position yielding the best fitness value for the ith particle. $gbest = (gbest_1, gbest_2, ..., gbest_D)$ is the global best particle found by all particles so far. The parameter w is called inertia factor, which is used to balance the global and local search abilities of particles

[28]. Larger inertia weights indicate larger exploration through the search space while smaller values of the inertia weight restrict the search on a smaller space [11]. Generally, PSO starts with a larger w, and decreases gradually over the iterations [1]. $rand1_{ij}$ and $rand2_{ij}$ are two uniform random numbers generated independently within the range of [0,1], c_1 and c_2 are two learning factors which control the influence of the social and cognitive components, and t = 1, 2, . . . indicates the iteration number [29].

2.2 The structure of membrane with OLMS

In the MIPSO, a P system with a distributed parallel framework is used to properly organize PSO. To keep a good balance between exploration and exploitation with a limited computational work. The OLMS with m elementary membranes is shown in Fig.1, and is used to design the membrane structure, labeled with $1 \sim m$ elementary membranes and a skin membrane. The structure in OLMS is listed as follows [38,37,40].

(1) Initialize a one level membrane structure as $[[\ ]_1, [\ ]_2, [\ ]_3, ...[\ ]_m]_0$, where m elementary membranes are labeled by $1, ..., m$, and the skin membrane is labeled by 0;

(2) Initialize the population $X = (x_1, x_2, x_3, ..., x_n)$, through D−dimensional space with n particles. $x_i = (x_{i1}, x_{i2}, x_{i3}, ..., x_{iD})$ denotes the position of ith particle in this population. $v_i = (v_{i1}, v_{i2}, v_{i3}, ..., v_{iD})$ denotes the current speed of ith particle in population. Put the particles into m elementary membranes by equal distribution from the population X, and make sure every elementary membrane have at least one individual, and skin membrane is empty. The remaining $(n - m)$ individuals are randomly assigned to m elementary membranes. The specific operations are as follows: $\omega_0 = \lambda, \omega_1 = q_1q_2q_3...q_{n_1}, \omega_2 = q_{n_1+1}q_{n_1+2}q_{n_1+3}...q_{n_2}, ..., \omega_m = q_{n_{m-1}+1}q_{n_{m-1}+2}...q_{n_m}, n_1+n_2+...n_m \leq n$, where n is the population size, $q_i(1 \leq i \leq n)$ denotes the particle individual.

(3) $G = (g_1, g_2, ..., g_m)$ is the evolution iteration in each elementary membrane. They are mutually independent. $g_i(i = 1, 2, ..., m)$ is the evolution iteration for ith particle, which is a random number in $[1, g_{max}]$.

(4) Conduct the optimal operation with PSO separately in each elementary membrane, and the detailed operation is as follows.

  (i) Evaluate the fitness value for each particle in the membrane.
  (ii) Compare the best value of the $pbest$ to the fitness value of each particle experienced. If it is better, assume it to be the optimal value of the current particle.
  (iii) Compare each particle's optimal value $pbest$ with the group's optimal value $gbest$. If it is better, set it as the optimal value of the current group.
  (iv) Update the speed and location of each particle by the equation (1) and (2).

(5) Perform transformation and communication rules, i.e., information exchange between the elementary membranes and the skin membrane. PSO is guided by the current optimal self and the optimal overall population. It passes through speed and location to complete evolution for new species. Therefore, PSO transmits the optimal individual (from the m elementary membranes) in each membrane to the skin membrane by transshipment operation as well as [26].

3 The Proposed Approach

In this section, we introduce the structure based on a cell-like P system where the local optimization is performed in each elementary membrane. Through the exchange of individual information from the elementary membrane, the optimal individual is selected, and returned to each region to replace the worst solution. The next generation of PSO evolution is influenced in the elementary membrane to complete the population evolution. For more robustness and more quick convergence, the set of local optimal solution (SLOS) and the set of hyperparameter optimization (SHO) are used to adjust the fitness function. The relationships between the membrane system model here used and the evolution-communication p-system model are: Similarities: The evolutionary computing is performed inside the elementary membrane, Each elementary membrane task is carried in multiple processes. Differences: 1. The surface membrane interferes with the global optimal solution of PSO in each elementary membrane, that is, the optimal one in the elementary membrane is selected and sent back to the submembrane; 2. The results of each elementary membrane operation is sorted and combined for the new round in hyperparameters.

3.1 The description of MIPSO with OLMS

PSO has become a popular optimizer and is applied in practical optimization problems. In the past decades, many variants of PSO have been proposed. As researchers have learned about the technique, they derived new versions aiming at different demands and published theoretical studies of the effects of the various parameters and proposed many variants of the algorithm [31]. The research status and the current application of the algorithm as well as the development in the future direction are reviewed in [32]. The different variants of PSO with respect to initialization, inertia weight and mutation operators are listed in [33]. The important factors and parameters of PSO are summarized with the available literature of the PSO algorithm [34]. It provided the advances in PSO, including its modifications, hybridization, extensions, theoretical analysis, and parallel implementation. Also, provided a survey on applications of PSO in the following eight fields: electrical and electronic engineering, automation control systems, communication theory, operations research, mechanical engineering, fuel and energy, medicine, chemistry, and biology [35]. This study comprises of a snapshot of particle swarm optimization from the authors perspective, including variance in the algorithm, modifications and refinements introduced to prevent swarm stagnation and hybridization of PSO with other heuristic intelligent algorithms [36]. After the first process of PSO, their optimal convergence solution will send to the skin elementary from elementary membrane, then the best individual will be selected in the skin elementary, and send back to the elementary membrane for replacement as the global optimal solution. The convergence results in each elementary are sent to the skin membrane, the variance is used to determine the performance of results, and lead to the next round of hyperparameter in PSO evolution for elementary membrane.

In this paper, to consider the superiority of a global and local optimization algorithm, we use a membrane-inspired optimization algorithm with OLMS [25,26] structure. OLMS is currently the most widely used membrane structure, which is a spe-

cial hierarchical membrane structure derived from a cell-like P system. It has a certain number of elementary membranes inside the skin membrane. The communication in OLMS is usually a global process and can be executed between any two or more elementary membranes. The membrane computing algorithm with OLMS shows better optimization performance than their counterparts because of their improved capacity of balancing exploration and exploitation, which is derived from their better balance between convergence and diversity [40]. For example, the membrane-inspired algorithm and particle swarm optimization (MIPSO) [37,38] are used to optimize the design of a proportional-integral-derivative controller [39], mobile robot path planning [40,41]. Moreover, MIPSO uses the representation of individuals, evolutionary rules of particle swarm optimization, one-level membrane structure (OLMS) [46] and transformation or communication-like rules in P systems to design its algorithm.

3.2 The modified MIPSO structure

Membrane computing can provide flexible evolution rules and parallel-distributed framework [15], which is very beneficial to the membrane-inspired evolutionary algorithms (MIEAs). In [16], a certain number of hierarchical membrane structures in the skin membrane were combined with evolutionary algorithms (EAs) for multi-objective optimization problems. The quantum-inspired evolutionary algorithms are combined with P systems to solve image processing problems [21]. Also PSO with one-level membrane structure is used to solve broadcasting problems of P systems [25]. The investigations verify the usefulness of the introduction of P systems into EAs to solve many real-world applications [41]. In our search, there is no work which focuses on the use of membrane computing to solve image thresholding by PSO, which is one of very significant applications.

The modified membrane-inspired particle swarm optimization (MIPSO) structure is shown in Fig.1. It is a hierarchical arrangement of membranes in a cell-like P systems [42]. Several membranes can be placed in the skin membrane, and separates the system from the environment. A membrane without any other membranes inside is called an elementary membrane. Each membrane determines a compartment or a region. Each region contains a multi-set of objects, which evolve in terms of various rules such as transformation/communication rules [37].

Formally, a cell-like P system with an output membrane set of objects is as follows [43,44].

$$\Pi = (V, T, \mu, \omega_1, ..., \omega_m, R_1, ..., R_m, i_0) \tag{3}$$

where

(1) V is an alphabet and its elements are called objects;
(2) $T \subseteq V$ (the output alphabet);
(3) μ is a membrane structure consisting of m membranes, with the membranes labeled by the elements of a given set H containing m labels, i.e., $H = 1, 2, ..., m$, where m is called the degree of Π;
(4) ω_i , $1 \leq i \leq m$, are strings which represent multisets over V associated with the regions $1, 2, ..., m$ of μ ;

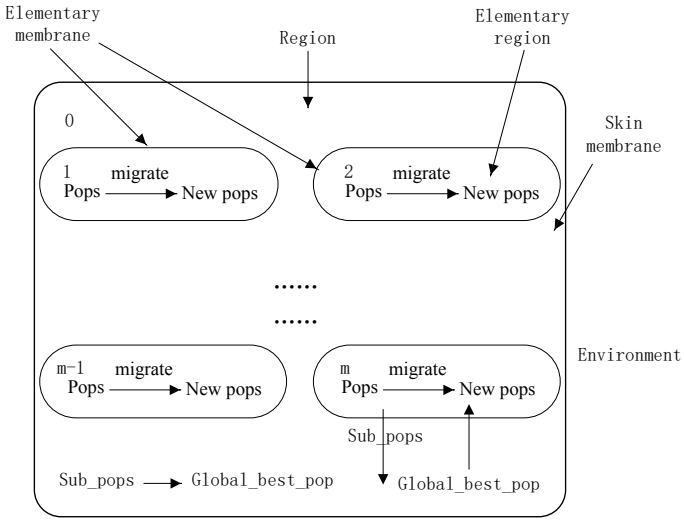


Fig. 1. The structure of cell-like P system in modified MIPSO

(5) R_i, $1 \leq i \leq m$, are finite sets of evolution rules over V associated with the regions $1, 2, ..., m$ of μ;
(6) i_0 is the from one of the labels of $1, 2, ..., m$ and region is considered as output region of the system.

The rules of R_i, $1 \leq i \leq m$, have the form $a \rightarrow v$, where $a \in V$ and $v \in (V \times (here, out, in))^*$. The multiset v consists of pairs (b, t), where $b \in V$ and $t \in (here, out, in)$, $here$ means that b stays in the region where the rule is used; out indicates that b exits the region and in means that b will be transferred to one of the membranes contained in the current region. All the rules are chosen in a non-deterministic way. Moreover, the rules can be cooperative, non-cooperative, or catalytic.

3.3 Hyperparameter optimization in MIPSO

By optimizing a series of hyperparameter [51,52,53,54], it is shown that the descending of convergence speed can be achieved and the intelligent body with higher performance can be obtained. The number of elementary membrane is m, from the equation of PSO, we list the vectors of parameters (hyperparameter) , as follows:

$$\overrightarrow{W} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}; \quad \overrightarrow{C_1} = \begin{bmatrix} c_{11} \\ c_{12} \\ \vdots \\ c_{1m} \end{bmatrix}; \quad \overrightarrow{C_2} = \begin{bmatrix} c_{21} \\ c_{22} \\ \vdots \\ c_{2m} \end{bmatrix}; \quad \overrightarrow{mv} = \begin{bmatrix} mv_1 \\ mv_2 \\ \vdots \\ mv_m \end{bmatrix} \tag{4}$$

Step 1: Construct hyperparameter optimization matrix:

$$M = \begin{bmatrix} w_1, c_{11}, c_{21}, mv_1 \\ w_2, c_{12}, c_{22}, mv_2 \\ \vdots, \quad \vdots, \quad \vdots, \quad \vdots \\ w_m, c_{1m}, c_{2m}, mv_m \end{bmatrix} \tag{5}$$

For 1-m, put them in descending order, that is, $fit_1 \geq fit_2 \geq ... \geq fit_m$.
Step 2: Store the vector of fitness function according to variance.

$$\overrightarrow{fit} = \left[fit_1, fit_2, ..., fit_m \right]^{\mathrm{T}} \tag{6}$$

Step 3: Construct linear combination based on the fitness function.

$$\overrightarrow{fit} := (\overrightarrow{fit} - fit_m)/fit_m \tag{7}$$

Step 4: Make a strategy for the situation such that fit_m may be 0.

$$fit_m = 0.5 * fit_{m-1} \tag{8}$$

Step 5: Add a random number to the fitness function for more robustness.

$$\overrightarrow{fit} = \overrightarrow{fit} + rand(0, 1) * max(\overrightarrow{fit}) \tag{9}$$

Step 6: Normalize $\overrightarrow{fit}$.
Step 7: Obtain new parameters $(w^*, c_1^*, c_2^*, mv^*)$ from next computing.

$$\overrightarrow{fit} = \left[fit_1, fit_2, ..., fit_m \right]^{\mathrm{T}} \circ \begin{bmatrix} w_1, c_{11}, c_{21}, mv_1 \\ w_2, c_{12}, c_{22}, mv_2 \\ \vdots, \quad \vdots, \quad \vdots, \quad \vdots \\ w_m, c_{1m}, c_{2m}, mv_m \end{bmatrix} \tag{10}$$

where '$\circ$' defines Hadamard product. Put the fitness function of new parameters and the optimal individual to each elementary membrane.

In this paper, we consider the communication rules since the objects in the PSO with cell-like P system evolve independently. The flowchart of the proposed MIPSOH is shown in Fig. 2, and the details of the pseudo code of the MIPSOH is designed as follows.

4 Experimental Results

In several experiments, the individual in each elementary membrane is randomly initialized. Compared with grid search, it has been proved that random search has greater advantages in the scientific community. It is easy to implement. The elementary membrane initializes itself with its own function, not need the skin membrane in the process

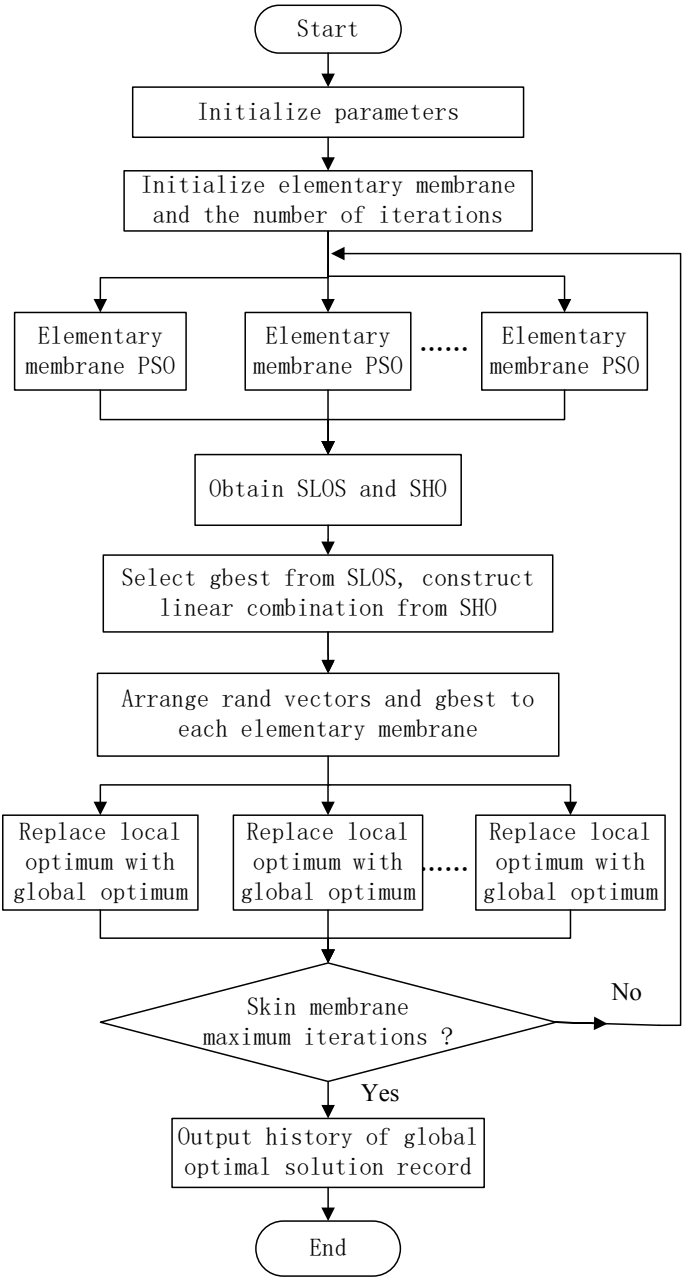


Fig. 2. Flowchart of the proposed MIPSOH

Algorithm 1 The pseudo code of the MIPSOH

Input: The pixel values of image $I(n * n)$
Output: The thresholding of image $I(n * n)$

```
Begin
t = 1
(1) Initialization of parameters for PSO and membrane structure, number of iterations
(2) Allocate particles in each elementary membrane
While (not termination condition) do
  for i = 1 : m
(3)   Perform PSO in the ith elementary membrane by maximizing the variance
(4)   Obtain the SLOS and SHO from each elementary membrane
(5)   Select gbest from SLOS, construct linear combination based on SHO
(6)   Add a random number to the fitness function
  end
(7) Implement communication rules
t = t + 1
end
end
```

of I/O. The parameters of the proposed multi-level thresholding method based on P systems are given as follows: (i) The cell-like P system includes 8 elementary membranes (m = 8), where the number of objects (population) contained in each elementary membrane is 32, and the maximum execution number step is 3, (ii) In the position-velocity model, maximizing the between-class variance, or minimizing within-class variance is regarded as our criterion to find the optimal segmentation thresholds in gray level images.

For PSO-based method, basic position-velocity model is employed and its parameters are: population size 32, randomly, $Gmax$ (maximum generation number)= 10, c_1 = random(m) * 4, $m \in (0, 1)$, $c_2 = 4 - c_1 + 1$, and w randomly varies from 0.0 to 1.0. For DE-based method, its parameters are: population size 30, F (scale parameter) = 0.5, CR (crossover rate) = 0.3 and $Gmax$ = 10. The experimental results are obtained on a computer with a Core 8 Intel(R) Core(TM) 2.8 GHz, and 16.0GB RAM.

4.1 The relation of thresholding results and between-class variance

The higher between-class variance with better segmentation is obtained for the same image in a certain extent in Fig.3 and Fig.4. Because the fitness function is decided by the maximization of between-class variance, i.e., the discriminant criterion is to maximize the separability of the resultant classes in gray levels. In Fig.3, the lena face is more intact in the higher convergence, which means better thresholding. To further express the relationship between segmentation effect and convergence value, we select a bar image in Fig.4. The obtained convergence becomes larger, the subtle line part that can be separated is more complete and consistent with the real target.

Fig. 3. The lena segmentation result by different between-class variances

Fig. 4. The edge segmentation result by different between-class variances

Table 1. Time and between variance of different methods for lena test image

K-means		DE		PSO	
time(s)	between variance	time(s)	between variance	time(s)	between variance
4.621	3569.79	0.418	3727.16	2.738	4004.69
4.967	3569.79	0.584	3796.09	2.714	4011.44
5.362	3569.79	0.397	3730.14	0.880	3997.51
4.967	3569.79	0.225	2065.97	0.882	3931.10

4.2 The evaluation of time and between-class variance

From the above discussion, we know that for the same image, the larger between-class variance is better segmentation through different methods. The time (in seconds) and between-class variance of different methods for lena (512×512) and patch (231×34) test image are listed in table [1-4]. It can be seen that the method MIPSOH has the best and stable convergence in the lena and the patch images. If the between-class variance of the image is bigger, the MIPSOH method has more advantage. In table 1 and 2, DE method costs the shortest time. However, its between-class variance is obviously lower than MIPSOH and with intense fluctuation. Compared to PSO method, the MIPSOH possess more time, but has better between-class variance. The rapidity and stability of between-class variance in MIPSOH are obviously superior to other methods.

Table 2. Time and between variance of different methods for lena test image

DEPS		MIPSO		MIPSOH	
time(s)	between variance	time(s)	between variance	time(s)	between variance
4.448	3951.27	1.441	4004.56	0.934	4012.39
3.351	3978.41	1.124	4013.53	0.963	4011.93
3.341	3998.90	1.227	4003.76	0.934	4013.44
3.335	4003.92	1.230	4009.13	0.956	4010.96

Table 3. Time and between variance of different methods for patch image

K-means		DE		PSO	
time(s)	between variance	time(s)	between variance	time(s)	between variance
0.122	2890.603	0.021	2891.787	0.064	2526.152
0.105	2891.685	0.011	2436.941	0.064	509.9181
0.125	2890.603	0.014	1462.304	0.069	702.2191
0.103	2890.603	0.013	1495.740	0.019	1255.884

Table 4. Time and between variance of different methods for patch image

DEPS		MIPSO		MIPSOH	
time(s)	between variance	time(s)	between variance	time(s)	between variance
0.337	2889.614	0.251	2889.122	0.236	2850.66
0.334	2836.958	0.235	2891.076	0.234	2795.04
0.561	2648.647	0.241	2889.135	0.259	2886.31
0.328	2876.170	0.235	2892.008	0.258	2876.17

4.3 Statistical comparisons of test images

To compare the performance of several algorithms on the test suite, Friedman test [29,30] is used. Table 5 shows the average ranking of Kmeans, PSO, PSOPS, PSOTPS[2], DE, MIPSO, MIPSOH. The highest ranking is shown in bold. It can be seen that, the order of the performance of the seven methods ranks are as follows: MIPSOH, differential evolution P system (DEPS) [**?**], PSO, PSOPS, DE, PSOTPS, Kmeans. The highest average ranking is obtained by the MIPSOH method. It demonstrates that DNSPSO is the best one among the seven methods. Furthermore, to compare the performance differences between MIPSOH and the other six methods, we conduct a Wilcoxon signed-rank test [29]. Table 6 shows the resultant p-values when comparing MIPSOH and the other

five algorithmssix methods. From the results, it can be seen that MIPSOH is significantly better than all methods within-class variance and computation time.

Table 5. Average rankings achieved by Friedman test for the seven methods.

Methods	Rankings of Within-class variance	Rankings of computation time
MIPSOH	**6.25**	5.90
DEPS	4.80	3.90
PSO	3.95	**7.00**
PSOPS	3.85	2.60
DE	3.45	5.10
PSOTPS	2.85	1.65
Kmeans	2.85	1.85

Table 6. Wilcoxon test between MIPSOH with other methods on within-class variance and computation time.

MIPSOH vs.	p-Values of within-class variance	p-Values of computation time
Kmeans	2.191e-04	8.857e-05
PSO	8.918e-04	8.857e-05
PSOPS	2.925e-04	8.857e-05
PSOTPS	2.191e-04	8.857e-05
DE	1.318e-04	2.535e-04
DEPS	4.274e-03	8.857e-05

If the assumption that ”The performance of all algorithms are equaled” is rejected, the conspicuousness performance of the algorithms is different.At the time, the ”post-hoc test” is needed to distinguish the algorithms further. Nemenyi post-hoc test is commonly used. From the friedman-test, the algorithm average order value in Nemenyi post-hoc test is computed and the following results and obtained: 1) the conspicuousness of MIPSOH is better than PSO, PSOPS, DE, PSOTPS and Kmeans in within-class variance, 2) the conspicuousness of MIPSOH is better than PSOPS, PSOTPS and K-means in computation time.

4.4 The histogram distribution of test images

The well-known images of Lena, Peppers, Hunter and Baboon are shown in Fig. 5(a1, b1, c1, d1). The size of images is 512×512. Fig. 5(a2, b2, c2, d2) shows the histograms of the test images. The distribution of the histogram has multiple valley and peaks in Fig. 5(a2, b2), there is a flat uniform distribution in Fig. 5(c2), and a wave peak in the middle of the concentration in Fig. 6(d2).

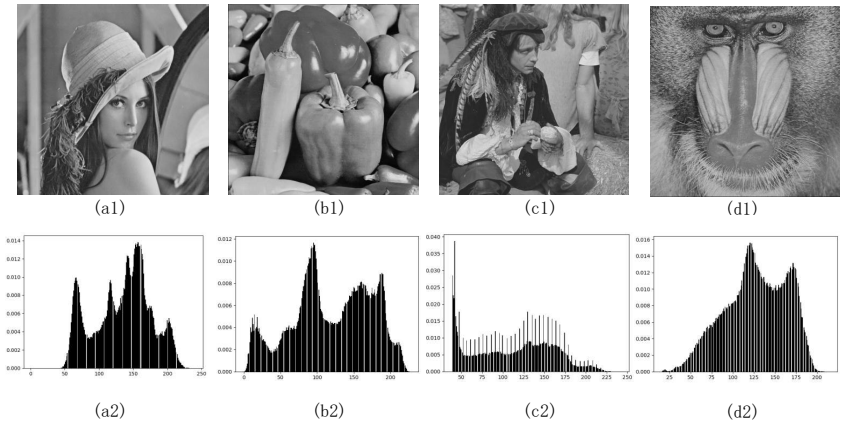


Fig. 5. Histogram of four test images

4.5 The segmentation results

In the experiment, we conduct the 3 and 4 level gray thresholding segmentation. Table 7 lists the optimal thresholds obtained by different methods. Table 8 and 9 show the comparisons of CPU time(s) and within-variance for different methods. The proposed MIPSOH is almost the shortest time except for pure PSO method. However, PSO possesses the higher within-class variance, which means the PSO method has worse result. To further demonstrate the subtle difference, the lena segmentation result by different within variations are shown in Fig.7. The smaller within variance indicate better segmentation result. For example, the hair on the shoulder, the transition is more natural and smooth, especially in the black and white part.

The segmentation thresholding is shown in table 7 by three and four levels. Their corresponding cost time and within-variance are listed in table 8 and table 9. We can see that the proposed MIPSOH cost less time except the PSO in table 8, and obtain the almost best within-variance in table 9. A Hyper-parameter optimization is added to the MIPSO, called MIPSOH. It can get a smaller within-variance and less time than MIPSO. Because of the search strategy, it makes the MIPSO better than PSO. MIPSO can correct mistakes in time. MIPSOH cost less time and with smaller within-variance, that is, the proposed method gain better results.

Table 7. Optimal thresholds obtained by different methods

Images	C	Kmeans	PSO	MIPSO	DE	DEPS	MIPSOH
baboon	2	13, 95	101, 153	97, 147	106, 153	102, 150	100, 150
	3	1, 30, 93	83, 113, 155	91, 142, 157	112, 139, 159	104, 125, 162	87, 117, 15
butterfly	2	25, 69	79 ,155	74, 152	73, 143	67, 153	134, 219
	3	2, 60, 65	56, 142, 222	72, 134, 239	78, 138, 224	77,155, 236	92, 154, 22
House	2	16, 74	91, 160	108, 176	103, 150	97, 166	94, 159
	3	27,69, 102	54, 104, 170	49, 90, 166	86, 108, 161	59, 91, 157	94, 159, 22
Hunter	2	19, 91	78, 134	85, 121	90, 125	93, 143	96, 144
	3	3, 44,82	87, 130, 151	87, 112, 154	105, 139, 188	69, 106, 166	85, 122, 14
Lena	2	21, 100	98, 142	108, 185	111, 193	107, 187	112, 184
	3	12, 49, 98	6, 127, 187	104, 137, 183	79, 124, 170	83,142, 185	84, 127, 17
Peppers	2	18, 85	67, 139	65, 135	74, 146	63, 134	65, 137
	3	17, 52, 98	59, 120, 151	47, 125, 165	61, 127, 171	79,143, 190	63, 131, 16

Table 8. Comparisons of CPU time(second) for different methods

Images	C	Kmeans	PSO	MIPSO	DE	DEPS	MIPSOH
baboon	2	6.008	0.353	4.799	1.952	5.548	1.088
	3	10.123	0.958	4.384	1.928	6.062	1.269
butterfly	2	7.241	0.152	4.342	0.666	5.151	0.564
	3	8.603	0.473	4.042	1.088	5.154	0.590
House	2	1.129	0.144	4.373	0.359	4.199	0.346
	3	3.030	0.059	3.837	0.385	4.225	0.241
Hunter	2	10.523	0.213	4.638	2.052	5.557	1.055
	3	12.835	0.195	4.152	1.149	5.609	0.982
Lena	2	10.850	0.282	4.716	0.970	1.182	0.824
	3	11.959	0.466	5.943	1.597	5.895	0.670
Peppers	2	11.103	0.483	4.518	1.113	5.523	0.812
	3	10.712	0.762	4.317	1.115	5.902	1.023

Table 9. Comparisons of within-variance for different methods

Images	C	Kmeans	PSO	MIPSO	DE	DEPS	MIPSOH
baboon	2	112429.506	64508.220	64129.684	65032.446	64041.875	**63983.384**
	3	113039.766	53859.849	54803.094	57198.104	56641.765	**53657.78**
butterfly	2	346776.735	**206724.048**	206988.749	210572.396	209088.022	221828.289
	3	346776.735	145928.746	160087.634	141161.562	144216.758	**140368.893**
House	2	245458.759	57861.197	67402.838	63785.278	58747.846	**57800.190**
	3	207427.622	55974.995	54413.223	51593.539	52627.199	**45257.855**
Hunter	2	137203.308	101098.105	100762.446	100648.895	97451.953	**97276.595**
	3	145724.256	84746.151	82602.652	82405.701	90472.711	**82231.434**
Lena	2	130779.576	95418.374	**90501.569**	93110.191	90776.062	90619.838
	3	133073.942	93425.283	**64231.631**	77758.786	69715.811	69208.137
Peppers	2	225563.980	116961.621	116490.022	121324.620	116531.881	**116528.324**
	3	173192.824	90631.167	86674.646	**83042.952**	105550.733	84363.763

Fig. 6. The lena segmentation result by different within variations

5 Conclusions and future work

In this paper, a novel approach is introduced by combining PSO, membrane computing and hyperparameter optimization to find best thresholding for image segmentation. This work is motivated by two aspects. On the one hand, the application extension of cell-like P systems requires further discussions. On the other hand, the image segmentation in computer vision is a very important problem and its thresholding accuracy needs to be enhanced. Furthermore, the excellent schema recording and parallelism of PSO is inspiring and energizing. The process of image segmentation is performed on the appropriate thresholding through the strategy. The pixel is regarded as a particle, and the particle swarm migration is carried out to obtain the local optimal solution. These local optimal solutions are candidate for the skin membrane, and the result of selected optimal solution will return to elementary membrane for the next iteration until the requirement is met. Experiments conducted on several cases of images verify the feasi-

bility and correctness of the presented approach. The proposed method is slightly slower than PSO in speed. In the performance of convergence, on the one hand, the variance is small in each algorithm performance test, on the other hand, the convergence effect is better than others, and with better significance, the P value in significance testing is less than 1

The proposed method has the below advantages: several membrane can be carried at parallel mechanism, the results is more convergence. It is easier to do something more upper operation such as transport operations, hyperparameter optimization. The higher dimension will reflects the swarm intelligence, the membrane make it as a system, not a bunch, and can assemble more abstract information. The selected PSO is easy to realized by matrix, with the advantages in speed. Compared with genetic variation, the transfer process of particle swarm is a simpler hypothesis with advantages brought by simple hypothesis: It is easy to combine and optimize.

In future, we will focus on the improvement of thresholding accuracy and the reliability of results. Following this work, histogram of image or other features will be also considered in the future study. In addition, the most serious problem related to the traditional histogram-based thresholding is that they are incapable of separating different regions having the same intensity but located in different locations of an image. Overall, thresholding-based methods do not work well in images without recognizable peaks and valleys, and cannot consider spatial relationships. Therefore, their applicability in non-trivial image datasets are not still feasible. Noisy image segmentation is an important topic in real world applications such as medical image processing. The clustering algorithm has great potential to deal with both segmentation-oriented denoising and segmentation at the same time. The clustering process is based on a similarity metric between a feature vector attributed to each datapoint and cluster centers. Therefore, using a variety of features such as intensity, color, texture, and spatial, and spatial-frequency features is possible.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (61806028, 61672437 and 61702428), Sichuan Science and Technology Program (2018GZ0245, 2018GZ0385, 2018GZ0086, 2018GZ0185, 18ZDYF3191, 2017GZ0431), Sichuan Education Department Program (2017Z053, 17ZB0095) and New Generation Artificial Intelligence Science and Technology Major Project of Sichuan Province (2018GZDZX0044).

References

1. Mirghasemi, S., Rayudu, R., & Zhang, M. A heuristic solution for noisy image segmentation using Particle Swarm Optimization and Fuzzy clustering. IEEE International Joint Conference on Computational Intelligence, 1, 17-27 (2015).
2. Peng H, Wang J, Shi P. A novel image thresholding method based on membrane computing and fuzzy entropy. Journal of Intelligent & Fuzzy Systems, 24(2), 229-237 (2013).
3. Otsu N., A threshold selection method from gray level histograms, IEEE Transactions on Systems, Man and Cybernetics SMC-9, 62-66 (1979).

4. Kapur J., Sahoo P.. and Wong A., A new method for gray-level picture thresholding using the entropy of the histogram, Computer Vision, Graphics and Image Processing, 29(3), 273-285 (1985).
5. Pare, S., Kumar, A., Bajaj, V., & Singh, G. K. An efficient method for multilevel color image thresholding using cuckoo search algorithm based on minimum cross entropy. Applied Soft Computing, 61 (2017).
6. Bhandari, A. K., Kumar, A., & Singh, G. K. Tsallis entropy based multilevel thresholding for colored satellite image segmentation using evolutionary algorithms. Expert Systems with Applications, 42(22), 8707-8730 (2015).
7. Zhu, H., Zhuang, Z., Zhou, J., Zhang, F., Wang, X., & Wu, Y. Segmentation of liver cyst in ultrasound image based on adaptive threshold algorithm and particle swarm optimization. Multimedia Tools & Applications, 76(6), 1-18 (2017).
8. Wang, X., Wang, Q., Yang, H., and Bu, J. Color image segmentation using automatic pixel classification with support vector machine. Neurocomput. 74, 18, 3898-3911 (2011).
9. Kennedy, J. and Eberhart, R. C. Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ. 1942-1948 (1995).
10. Eberhart R., and Kennedy, J. A new optimizer using particle swarm theory. In Micro Machine and Human Science, Proceedings of the Sixth International Symposium, 39-43 (1995).
11. Engelbrecht A., Computational Intelligence: An Introduction, 2nd ed.Wiley Publishing (2007).
12. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences, 61(1), 108-143 (2000).
13. Păun, Gh.: Membrane computing. An introduction. Springer-Verlag, Berlin (2002)
14. Păun G., Rozenberg, G., Salomaa A., The oxford handbook of membrane computing , New York: Oxford University Press, 55-60 (2010).
15. Zhang, G., Gheorghe, M., Pan, L., & Prez-Jimnez, M. J. Evolutionary membrane computing: a comprehensive survey and new results . Information Sciences, 279, 528-551 (2014).
16. Liang H., He X., Ning W., & Yi X. P systems based multi-objective optimization algorithm. Progress in Natural Science:Materials International, 17(4), 458-465 (2007).
17. Zhang, G., Gheorghe, M., Pérez-Jiminez, M.J.: Real-life Applications with Membrane Computing. Springer (2017).
18. 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(3), 1528-1542 (2013).
19. He, J., Xiao, J., Liu, X., Wu, T., Song, T.: A Novel Membrane-Inspired Algorithm for Optimizing Solid Waste Transportation. Optik-International Journal for Light and Electron Optics, 126(23), 3883-3888 (2015).
20. Rong, H., Ge, M., Zhang, G., & Zhu, M. An approach for detecting fault lines in a small current grounding system using fuzzy reasoning spiking neural p systems. International Journal of Computers Communications & Control, 13(4), 521-536 (2018).
21. Zhang, G.X., Gheorghe, M., Li, Y. A membrane algorithm with quantum-inspired subalgorithms and its application to image processing; Natural Computing (2012).
22. Dłaz-Pernil, D., Berciano, A., Pe?a-Cantillana, F., & Gutirrez-Naranjo, M. A. Segmenting images with gradient-based edge detection using membrane computing. Pattern Recognition Letters, 34(8), 846-855 (2013).
23. Frisco, P., Gheorghe, M., Pérez-Jiménez, M.J. (Eds.): Applications of Membrane Computing in Systems and Synthetic Biology, in Emergence, Complexity and Computation Series, Springer (2014)
24. Cheng, J., Zhang, G., & Neri, F. Enhancing distributed differential evolution with multicultural migration for global numerical optimization . Information Sciences, 247(15), 72-93 (2013).

25. Zhang, G., Zhou, F., Huang, X., Cheng, J., Gheorghe, M., & Ipate, F., et al. A novel membrane algorithm based on particle swarm optimization for solving broadcasting problems. Journal of Universal Computerence, 18(13), 1821-1841 (2012).
26. Zhang, G., Gheorghe, M., & Wu, C. A quantum-inspired evolutionary algorithm based on p systems for knapsack problem. Fundamenta Informaticae, 87(1), 93-116 (2008).
27. Yahya, R. I., Shamsuddin, S. M., Yahya, S. I., Hasan, S., Al-Salibi, B., & Al-Khafaji, G. Image Segmentation Using Membrane Computing: A Literature Survey. Bio-Inspired Computing - Theories and Applications, Springer Singapore, 314-335 (2016).
28. Shi Y., Eberhart R., A modified particle swarm optimizer, in: Proceedings of the Congress Evoluationary Computer, 69-73 (1998).
29. Wang H., Sun H., Li C., Rahnamayan S. & Pan J., Diversity enhanced particle swarm optimization with neighborhood search. Information Sciences, 223(2), 119-135 (2013).
30. Derrac, J., Garcła, S., Molina, D., & Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm & Evolutionary Computation, 1(1), 3-18. (2011).
31. Wang, D., Tan, D., & Liu, L. Particle swarm optimization algorithm: an overview. Soft Computing, 22(2), 1-22 (2018).
32. Zhu, Y. B. Overview of particle swarm optimization. Applied Mechanics & Materials, 543-547(2), 1597-1600 (2014).
33. Imran, M., Hashim, R., & Khalid, N. E. A. An overview of particle swarm optimization variants. Procedia Engineering, 53(7), 491-496 (2013).
34. Darzi, S., Kiong, T. S., & Salem, B. Overview of particle swarm optimization (pso) on its applications and methods. Australian Journal of Basic & Applied Sciences (2013).
35. Zhang, Y., Wang, S., & Ji, G. A comprehensive survey on particle swarm optimization algorithm and its applications. Mathematical Problems in Engineering, 2015(1), 1-38 (2015).
36. Eslami, M., Shareef, H., Khajehzadeh, M., & Mohamed, A. A survey of the state of the art in particle swarm optimization. Research Journal of Applied Sciences Engineering & Technology, 4(9), 1181-1197 (2012).
37. Zhang, G., Zhou, F., Huang, X., Cheng, J., Gheorghe, M., & Ipate, F., et al. A novel membrane algorithm based on particle swarm optimization for solving broadcasting problems. Journal of Universal Computerence, 18(13), 1821-1841 (2012).
38. Zhou, F., Zhang, G., Rong, H., & Gheorghe, M. A particle swarm optimization based on P systems. IEEE. Sixth International Conference on Natural Computation (Vol.5855, 3003-3007) (2010).
39. Wang, T., Wang, J., Peng, H., & Tu, M. Optimization of pid controller parameters based on MIPSO algorithm. Icic Express Letters, 6(1), 273-280 (2012).
40. Zhang G., Pérez-Jiménez M.J., Gheorghe M. Fundamentals of Evolutionary Computation. In: Real-life Applications with Membrane Computing. Emergence, Complexity and Computation, vol 25. Springer International Publishing (2017).
41. Wang, X., Zhang, G., Zhao, J., Rong, H., Ipate, F., & Lefticaru, R. A modified membrane-inspired algorithm based on particle swarm optimization for mobile robot path planning. International Journal of Computers Communications & Control, 10(6) (2015).
42. Păun , G. Introduction to membrane computing. In: Prez-Jimnez, M.J, et al. (Eds.) Applications of Membrane Computing. Springer, Heidelberg, 1-42 (2006).
43. Cheng, J., Zhang, G., & Zeng, X. A novel membrane algorithm based on differential evolution for numerical optimization. International Journal of Unconventional Computing, 7(3), 159-183 (2011).
44. Păun, G and Rozenberg, G. A guide to membrane computing. Theor. Comput. Sci. 287, 73-100 (2002).
45. Price, K., & Price, K. Differential Evolution C A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Kluwer Academic Publishers (1997).

46. Zhang, G., Gheorghe, M., & Wu, C. A quantum-inspired evolutionary algorithm based on p systems for knapsack problem. Fundamenta Informaticae, 87(1), 93-116 (2008).
47. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science, 296(2), 295-326 (2003).
48. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae, 71(2-3), 279-308 (2006).
49. Zhan, Z-H.; Zhang, J.; Li, Y; Shi, Y-H. Orthogonal Learning Particle Swarm Optimization. IEEE Transactions on Evolutionary Computation. 15 (6): 832-847 (2011).
50. Zhang, Y.; Wang, S. Pathological Brain Detection in Magnetic Resonance Imaging Scanning by Wavelet Entropy and Hybridization of Biogeography-based Optimization and Particle Swarm Optimization. Progress in Electromagnetics Research C Pier. 152: 41-58 (2015).
51. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In Proceedings of the 12th Python in Science Conference, 13-20. (2013).
52. Hutter, H., and Leyton-Brown K.. An efficient approach for assessing hyperparameter importance. In ICML, volume 32 of JMLR Workshop and Conference Proceedings, 754-762 (2014).
53. Claesen, M., Moor, B. Hyperparameter Search in Machine Learning, 1-5 (2015).
54. Probst, P., Bischl, B., and Boulesteix, A. L. Tunability: importance of hyperparameters of machine learning algorithms (2018).

Fig. 7. Three-level thresholding images obtained by different methods. (a1)−(d1) K-means; (a2)−(d2) DE; (a3)−(d3)DEPS; (a4)−(d4)MIPSO; (a5)−(d5)MIPSOH.)

Fig. 8. Four-level thresholding images obtained by different methods. (a1)−(d1) K-means; (a2)−(d2) DE; (a3)−(d3)DEPS; (a4)−(d4)MIPSO; (a5)−(d5)MIPSOH.)

Alternative Representations of P Systems Solutions to the Graph Colouring Problem

James Cooper[0000−0001−9954−3280] and Radu Nicolescu[0000−0003−2498−1002]

University of Auckland, Auckland, New Zealand
jcoo092@aucklanduni.ac.nz
http://www.cs.auckland.ac.nz

Abstract. This paper first presents a simulation of the Simple Kernel P systems solution to the Graph 3-colouring Problem presented in a previous paper by Gheorghe *et al.*, implemented in a programming style named Concurrent ML, which is based in the concept of synchronous communication between logical processing elements. Realising that in fact the solution requires little communication, this paper then presents a compact cP systems solution to the same problem, which has the benefit of being easily adaptable to an n-colouring problem, where n is the desired number of colours of the system (in fact, only the specified colour symbols need to be changed). Successful and failing examples of that solution are also presented.

Keywords: cP systems · Simple Kernel P systems · Graph Colouring Problem · Concurrent ML

1 Introduction

The graph colouring problem is a deceptively simple problem in graph theory. In its most basic form, it is the problem of assigning labels such as colours to the nodes in a graph, such that no two connected nodes share a label, usually with the addition of the extra requirement of finding a minimum or specific number of required colours – clearly, for any graph with n nodes, it is trivial to colour it using n colours. The problem finds applications in many areas, including in timetabling and register allocation for compilers [5]. For the totally general case, no polynomial time solution has been found so far, though they have been found for specific forms of graph or particular numbers of colours. For example, in the case of 3-colouring, the best known solution takes $\mathcal{O}(1.3289^n)$ time [1].

Gheorghe *et al.* presented a solution to the 3-colouring problem, using communicating Simple Kernel P systems [4]. Based on this work, we first present a Concurrent ML implementation of the Simple Kernel P systems solution in [4] and then a fairly concise cP systems solution to the problem. We wish to emphasise that we do not consider our simulations or cP systems solution to be 'superior' to that of Gheorghe *et al.* (though we do think it demonstrates the utility of cP systems), but instead wish to complement their work with another alternative.

2 Simple Kernel P systems solution to the Graph Colouring problem in Concurrent ML

Concurrent ML is an approach to concurrency originally created by Reppy [8] for the programming language Standard ML of New Jersey (hence the name). It is based in the concept of synchronous message passing between independently executing logical processing elements over channels [6] (see Figure 1).

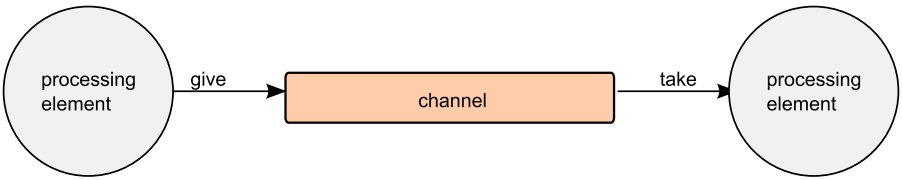


Fig. 1. In Concurrent ML, logical processing elements synchronously exchange values over channels. When one PE offers to give a value on a channel, and another offers to take on the same channel, they 'rendezvous' and exchange the value as a passed message.

We were interested in exploring Concurrent ML as another methodology to use for simulating P systems where communication is involved. Both Tissue P systems with symport/antiport and Simple Kernel P systems using communication appear to be logical choices to simulate with Concurrent ML. In all cases, communication is synchronous and involves a rendezvous between distinct processes/cells/compartments. We finally chose to implement Gheorghe *et al.*'s 3-colouring problem solution from [4], as it involves communication between compartments but is relatively low-complexity and thus appears to be suitable for a first attempt.

We chose to use the programming language F# with the library Hopac,[1] which is modelled on Concurrent ML and follows it closely.[2] Insofar as possible, the implementation has been created to faithfully follow the algorithm described in [4], and thus perhaps is not optimal in its efficiency, as an idiomatic specification of a Simple Kernel P system does not necessarily match to the idiomatic or efficient form of an F# program. 'Record' types are used to represent the individual compartments described in the paper. The program then advances through multiple steps, applying the rules (encoded as functions that operate on the record types) in accordance with the paper. Finally, once a solution is found, or found not to be possible, that is communicated to the environment.

Ultimately, however, this example is relatively trivial and involves little communication, and therefore does not test Concurrent ML significantly. Much

[1] `https://github.com/Hopac/Hopac`

[2] The final program can be found at `https://github.com/jcoo092/acmc2018`

of the operation of the algorithm in fact does not involve different compartments/processing elements at all. Instead, it is primarily based in the evolution of objects contained within the compartments, with minimal communication between compartments at the end. While that is highly effective in P systems [7], we think it would be interesting to see the results of using Concurrent ML for other problems where synchronous communication is a much bigger part the evolution of the system.

We do note here that, while our implementation is reasonably successful, it is the case that it is not particularly customisable, and it does not comport as well to the theoretical rules as the P-Lingua version created for the SKP system. Neither does it have any form of verfication or invariant detection, as provided by Spin and MeCoSim. We suggest that it may be worthwhile to pursue future work that seeks to improve one implementation/simulation by incorporating relevant parts of the other.

Concurrent ML seems to match to Simple Kernel P systems reasonably well, but also looks like it would fit well with Tissue P systems with symport/antiport [11] as well perhaps as Tissue P systems with Channel States [10], and Generalized Communicating P systems with minimal interaction rules [3]. Technically, the base form of Concurrent ML would only support antiport (i.e. one-way synchronous communication), but it is fairly simple to build two-way communication on top of it [9, Ch. 6].

2.1 Simulation results

We recorded the program's runtime on a number of differing graphs. Firstly, we used the graph shown in Figure 3 of [4], which took 2.6s to process. In keeping with that paper and following its definition of G(n,k), we also tested the graphs G(10, 1) and G(10,10), finding that it requires 0.3s for each. Here G(n,k) is used to represent a graph with n nodes, where all nodes are connected to node k in a hub-and-spoke formation.[3] We also decided to test our simulation using the classic Petersen graph, shown in Figure 2, and found that it required approximately 0.3s.

We further tried it on some complete graphs. For a complete graph of n = 10, the algorithm again requires 0.3s. We then tried it on a complete graph of n = 20, but the test had to be aborted after a matter of minutes due to a memory shortage. We then recorded running times of 0.83s, 2.6s, 8.88s, 26.6s and 84s for n = 11, 12, 13, 14 and 15, respectively. Given that we observed significant jumps in memory use in the course of running the larger graphs, it appears that the major cause of the rapid slowdown is likely to do with frequent ad-hoc memory allocations, though we have not as of yet profiled this fully.

Our implementation is entirely invariant to the edges of the graph, and the runing time depends only on the total number of nodes. This is to be expected, since it works as a simple brute-force exploration of the potential solution space.

[3] Note that this is different to the random graphs that are also commonly denoted by this notation.

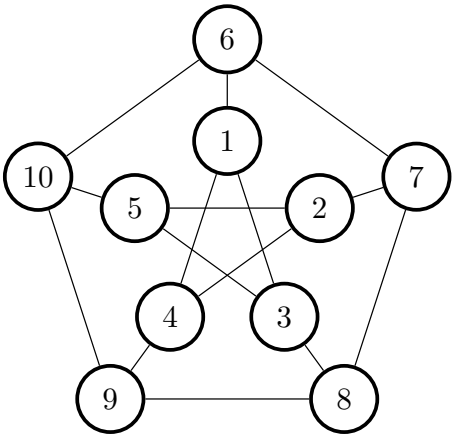


Fig. 2. The classic Petersen graph, with nodes labelled 1 to 10

In [4], a strategy of eliminating compartments for which there can be no further rule applications is mentioned. Using said strategy, results are typically achieved quite quickly. Our simulation does not have similar semantics as it does not select rules to apply at each step, but we note that applying the colour guard on rule $r_{2,2n+1}$ of the SKP system throughout the process results in the pruning of compartments which already contain invalid results (and so can safely be eliminated). Doing this reduces the evaluation of complete graphs to around 60 milliseconds or slightly more, since every compartment in them can always be entirely eliminated once four colours have been chosen. It also reduces the running time of Figure 3 from [4] to around 0.25s, and the Petersen graph, G(10, 1) and G(10,10) to 0.1s

3 cP systems solution to the Graph colouring problem

In working to create the Concurrent ML simulation of the problem, we noticed that in fact most of the work is performed by the instantiation of new objects, rather than communication. Thus, the problem appears to be an excellent fit to the pre-existing formulation of cP systems.

This problem, rather unsurprisingly, has a lot in common with the Hamiltonian Path Problem, and in fact our cP systems solution for the Graph Colouring problem is similar in some regards to our previous Hamiltonian Cycle Problem solution, as presented in [2]. Unfortunately, and unlike in [2], a lack of a time meant that as of yet we have been unable to construct simulations of the cP systems solution in software. We have a number of potential methods for this in mind, however.

We start construction of the system with a conceptual finite set $V \subset \mathbb{N}$ representing the nodes of the graph[4]. We further start with a top-level cell containing a set E of the edges in the graph where $E \subseteq \{e(n(i)n(j))\}_{i,j \in V;\ i \neq j}$, a starting node $s(S)$ where $S \in V$ is an arbitrary node in the graph, a functor v containing the labels of the nodes in V, and a set Δ of the potential colours – in the case of the three colour problem it might look like $\Delta = \{\delta(r), \delta(g), \delta(b)\}$. Using separate symbols in this fashion means that the algorithm can operate as an n-colour problem, without any other modification. As currently constructed, however, it makes no attempt to minimise the number of colours used.

3.1 Notation

A cP system can be described as a 6-tuple as shown below.

$$cP\Pi(T, A, O, R, S, s)$$

Where T is the set of top-level cells at the start of the evolution of the system, A is the alphabet of the system, $O \subseteq A$ is the sets of initial objects in the top-level cells, R is the sets of rules for each cell, S is the set of possible states of the sytsem, and s is the starting state of the system.

Our graph-colouring system can be formalised as

$$cP\Pi(1, A, O, R, S, s0)$$

where 1 is the single top-level cell of the system, $A = (\{b, c, e, l, m, n, s, v, \delta\} \cup \{\alpha\} \cup \{E\} \cup \{\Delta\})$ is all potential atoms, functors and other symbols of the system (α is used here to represent the counting symbol contained within objects that is typically represented using natural numbers), $O = (s(S) \cup \{v\} \cup \{E\} \cup \{\Delta\})$ is the set of initial objects contained within the top-level cell (where S is an arbitrarily chosen member of v), R is the ruleset in Section 3.2, $S = \{s0, s1, s2, s3, s4\}$ is the potential states of the system, and $s0$ is the initial state of the system.

3.2 Rules

Our entire ruleset is presented in Figure 3. It consists of seven rules, which are invariant to the problem graph at hand, listed in weak priority order. They are explained below:

1. This rule begins the evolution of the system. It converts the functor v into a functor b, which is used further to instantiate new objects with the possible colourings of the system. It merely selects one the node described by S, and assigns it a colour at random. The b object holds an l functor which keeps track of which iteration a given b belongs to; a multiset of m functors which track nodes and their assigned colours in a potential solution; and

[4] In fact, any finite set of arbitrary symbols could be used, but we use the natural numbers here for ease of reading.

$$
\begin{array}{lll}
s_0 & v(SZ) & (1)\\
& \quad \rightarrow_1 \ s_1 \ b(l(1)\ m(n(S)c(C))\ v(Z))\ l(1) & \\
& \qquad\qquad \mid \delta(C) & \\
& \qquad\qquad \mid s(S) & \\
& & \\
s_1 & b(M\ v(\lambda)\ \_) & (2)\\
& \quad \rightarrow_1 \ s_3 \ M\ !_{env} & \\
& & \\
s_1 & & (3)\\
& \quad \rightarrow_+ \ s_2 \ b(l(L1)\ m(n(X)c(C))\ m(n(Y)c(D))\ M\ v(Z)) & \\
& \qquad\qquad \mid b(l(L)\ m(n(X)c(C))\ M\ v(YZ)) & \\
& \qquad\qquad \mid e(n(X)n(Y)) & \\
& \qquad\qquad \mid \delta(D) & \\
& \qquad\qquad \neg\ \delta(C) = \delta(D) & \\
& & \\
s_1 & b(l(L)\ \_) & (4)\\
& \quad \rightarrow_+ \ s_2 \ \lambda & \\
& \qquad\qquad \mid l(L) & \\
& & \\
s_2 & b(m(n(X)c(C))\ m(n(Y)c(C))\ \_) & (5)\\
& \quad \rightarrow_+ \ s_1 \ \lambda & \\
& \qquad\qquad \mid e(n(X)n(Y)) & \\
& & \\
s_1 & & (6)\\
& \quad \rightarrow_1 \ s_4 & \\
& & \\
s_2 & l(L) & (7)\\
& \quad \rightarrow_1 \ s_1 \ l(L1) &
\end{array}
$$

Fig. 3. cP systems rules

a v functor which continues to track the reducing unexplored nodes of the graph. A separate global l functor is also instantiated at this point, which tracks what iteration number the system is currently at, which is used in later rules. This rule begins in state 0 and ends in state 1.

2. This rule is one of the potential end points of the evolution of the system. If there are no further remaining nodes to explore in at least one b object, then one of the bs is selected at random and the multiset of m functors containing a solution to the colouring problem is output to the environment. This rule begins in state 1 and ends in state 3. State 3 is used to indicate to the environment that a solution has been found and the evolution of the system has terminated. The change in state means that no other rule can be applied if this rule is applied.
3. This rule is arguably the heart of the process. For every pre-existing b object, a new one is generated with a random colouration assigned to an unexplored node, so long as there exists an edge between those nodes in the graph *and* the selected colours are not the same. It also increments the iteration counter within the b object. This rule begins in state 1 and ends in state 2.
4. This rule is a 'cleaning' rule, which removes all the existing b objects with the current iteration count, thus keeping the working space relatively clean. Recall that in P systems, all rules that can be applied are applied simultaneously at the end of a step, and thus this does not delete objects that are still being used in rule 3. This rule begins in state 1 and ends in state 2.
5. This rule is a complement to rules 3 and 4, and along with rule 7 occurs immediately after the application of rules 3 and 4. It detects existing b objects which have just been created where two connected nodes assigned the same colour are included in its multiset.[5] This rule is required because while rule 3 avoids choosing the same colour for the two nodes under consideration, it is possible that the newly included node is also connected with another pre-selected node with the same colour, which was not considered in a given application of rule 3. Thus, this rule removes any other illegal b instances before they can be operated upon further. This rule begins in state 2 and ends in state 1.
6. This rule is the other possible termination rule. It merely transitions to state 4, which is used to signal to the environment that no solution is possible. The key to this rule is that, because it transitions to state 4, it is only applicable if none of the other pre-existing rules are. This effectively means that it can only apply if all b objects have been removed from the system, indicating that every possible combination explored so far has already found a colouring conflict. This rule begins in state 1 and ends in state 4.
7. This rule merely increments the global iteration counter. It is placed last so that the termination rules can trigger before it, as otherwise it will always be applicable after the application of rule 1. This rule begins in state 2 and ends in state 1, and thus is applied simultaneously with rule 5.

[5] This is substantially similar to the guard on rule $r_{2,2n+1}$ of the Simple Kernel P systems solution.

3.3 Complexity

Our algorithm requires at most $2N$ steps, where N is the total number of nodes in the graph. In the successful case it will require that many, but for graphs where there is no possible n-colouring it may terminate early.

Rule 1 requires one application. Rules 2 and 6 between them require one application. Rules 3, 4, 5 and 7 all each require $N-1$ applications, but those are shared for 3 & 4 and 5 & 7, thus giving $2(N-1)$. That expands and simplifies to $2N$ overall, giving a complexity of $\mathcal{O}(N)$.

3.4 Comparison with the Simple Kernel P systems solution

Our cP systems solution requires only a handful of rules, which are *invariant* to the problem graph under study – there is *no* definition of a family of rules that are customised to the graph at hand. We require roughly equivalent starting objects to the SKP solution, but note that in fact we appear to require generally fewer symbols overall.

Table 1. skP systems vs cP systems

Type/specification	skP systems	cP systems
Alphabet	$n(n-1)/2+7n+10$	$15+\lvert\Delta\rvert+\lvert E\rvert$
Rules	$2n$ & $2n+7$	7
Maximum number of subcompartments	3^n+1	1
Number of steps	$2n+3$	$\leq 2n$

Table 1 provides a comparison of the Simple Kernel P systems solution presented in [4] compared with our solution presented above, and is based upon a table that appears in [4]. Note that the 'alphabet' and 'maximum number of subcompartments' entries are not completely comparable, however. Our alphabet consists of only 9 atoms and functors plus the unique counting symbol and the five potential states, as well as the colour symbols (Δ) used (therefore 18 symbols in the case of the 3-colouring problem), but with the potential nesting of atoms and functors these can combine into myriad forms – although as the semantic meaning of each never changes, we do not consider this to add a great deal of complexity. We note though that the specification of the edges of the graph are included in the alphabet for the SKP system, so we too count the magnitude of the edges set E in the size of our alphabet. While we only require one compartment, the top-level cell, it can potentially hold a large number of functors with their own sub-objects, with a maximum upper bound of $(n! \times |\Delta|) + |\Delta| + |E| + 1$ (though in practice it will never contain that many at once due to the prohibition on creating new objects which share a colour with the relevant connected edge).

4 Examples

To demonstrate the operation of the cP systems algorithm in a 3-colouring context, we provide a working example using the graph in Figure 4, and a failing example (where no successful 3-colouring is possible) using the graph in Figure 5.

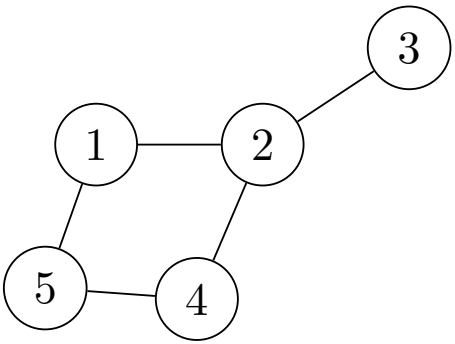


Fig. 4. Example undirected graph

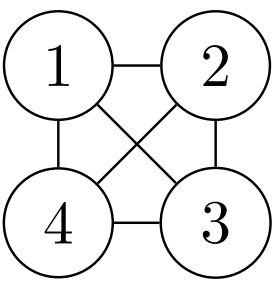


Fig. 5. Example undirected graph with no 3-colouring solutions

4.1 Successful example

For the graph in Figure 4, we begin a single top-level cell situated in the environment, with the hypothetical set of nodes $V = \{1, 2, 3, 4, 5\}$ from which we derive the object $v(n(1)\, n(2)\, n(3)\, n(4)\, n(5))$, and the set of objects

$$E = \{e(n(1)n(2))\, e(n(1)n(5))\, e(n(2)n(3))\, e(n(2)n(4))\, e(n(4)n(5))\},$$

as well as the set of colour objects $\Delta = \{\delta(r)\, \delta(g)\, \delta(b)\}$, in this case representing the colours red, green and blue. These latter sets of objects are all immediately

within the top-level cell, along with $s(1)$ to select node 1 at the beginning of the process. This beginning state (with the exception of the contents of the s object) is mandatory, but all following discussion in this subsection is merely one possible execution.

$$e(n(1)n(2)) \quad e(n(1)n(5)) \quad e(n(2)n(3)) \quad e(n(2)n(4)) \quad e(n(4)n(5))$$
$$\delta(r) \quad \delta(g) \quad \delta(b)$$
$$v(n(1)\, n(2)\, n(3)\, n(4)\, n(5)) \quad s(1)$$

Fig. 6. Initial set of objects inside the top-level cell for Figure 4

From this starting state, rule 1 is applied, choosing node 1 to start the process with, and selecting red as its colouring. This creates the standard b and l objects. No other rules are applicable at this point, as the system started in state 0. The application of this rule leaves the subcell in state 1. Rule 1 will not be applicable henceforth, as the rules provide no way to revert to state 0.

At the next step, rule 2 is checked but found inapplicable, as at this point v will not be empty, instead containing $v(n(2)\, n(3)\, n(4)\, n(5))$. Rule 3, conversely, is applicable and thus can generate further new b objects. In this instance, two edges exist from node 1 to other nodes (2 and 5 specifically), meaning that both executions can apply, generating new b objects containing chosen colourings. There are two other nodes that can be chosen to connect to, but only two other possible colourings to choose, as red is currently blocked. This leads to the creation of four b objects $b(l(2)\ m(n(1)c(r))\ m(n(2)c(g))\ v(n(3)\, n(4)\, n(5)))$, $b(l(2)\ m(n(1)c(r))\ m(n(5)c(b))\ v(n(2)\, n(3)\, n(4)))$ etc. (see Figure 7 for the full listing).

Simultaneously, rule 4 is applied to sweep away the pre-existing initial b object. Following that, at the next step, Rule 5 is not applicable because there is no current b object with two or more m objects inside, and rule 6 is not applicable because a state transition to state 1 has already been selected, and thus a transition to state 3 is invalid. Finally, rule 7 is also applied to remove the old l object and introduce a newly incremented one. Figure 7 lists the objects in the top-level cell at the end of this step.

The next two steps proceed similarly to that described in the above two paragraphs, with a key exception in that at this point there are b objects with at least two m objects inside, meaning that rule 5 is potentially applicable, though so far there are no matching b objects. In fact, these step proceed largely equivalently to the previous two, except that there are a greater number of new objects created (see Figure 8). At the previous steps, both the edges between node 1 and node 5, and node 1 and node 2, were explored. In these next steps, edges between node 1 and node 5, node 1 and node 2 (where these edges were

$$e(n(1)n(2)) \quad e(n(1)n(5)) \quad e(n(2)n(3)) \quad e(n(2)n(4)) \quad e(n(4)n(5))$$
$$\delta(r) \quad \delta(g) \quad \delta(b)$$
$$b(l(2)\ m(n(1)c(r))\ m(n(2)c(g))\ v(n(3)\,n(4)\,n(5)))$$
$$b(l(2)\ m(n(1)c(r))\ m(n(2)c(b))\ v(n(3)\,n(4)\,n(5)))$$
$$b(l(2)\ m(n(1)c(r))\ m(n(5)c(g))\ v(n(2)\,n(3)\,n(4)))$$
$$b(l(2)\ m(n(1)c(r))\ m(n(5)c(b))\ v(n(2)\,n(3)\,n(4)))$$
$$l(2)$$

Fig. 7. Set of objects inside the top-level cell after the third step (i.e. after application of rules 3, 4, 5 & 7) for Figure 4

not explored previously for a given b object), node 5 and node 4, and node 2 and node 3 are all explored with all objects that will not have direct colour conflicts as per rule 3 instantiated.

At the sixth step, a large number of further objects are created, some of which are listed in Figure 9. Note that some of those, however, contain instances where two connected nodes have been assigned the same colour, e.g. the first three b objects in Figure 9. They come into existence at the end of the sixth step, when rules 3 and 4 are applied and the system transitions to state 2. At that point, not only is rule 7 applicable, but here rule 5 will apply to every one of those illegal objects, and remove them at the end of the current step. Thus, at the end of the seventh step, those objects have been eliminated, and the working space is back in state 1 with only 'valid' objects in it, ready for further processing.

Figure 10 presents the objects inside the top-level cell at the end of the seventh step. Note that this is largely identical to the objects at the end of the sixth step, except with the invalid b objects removed by rule 5 (shown here as scratched out), and the l object incremented.

Steps 8 and 9 proceed in the same fashion as earlier. At the end of step 9 a number of objects which contain valid colourings will be present inside the top-level cell. Rule 2 will then select one of the solutions at random and emit it to the environment. For example, Figure 11 shows some of the potential solutions that could be generated, reflecting the state of the system at the end of the ninth step. The fourth and sixth of those listed in fact have contiguous colours, and so would have been removed by rule 5 at the end of the ninth step, but any of the others may be selected. In fact, the first solution shows that in this case it is possible to completely and validly colour this graph using only two colours. This solution may or may not be chosen at random.

At the end of the tenth step, rule 2 will select one of the possible solutions and emit it to the environment. The system will also transition to state 3, signalling that the process succeeded.

$$e(n(1)n(2)) \quad e(n(1)n(5)) \quad e(n(2)n(3)) \quad e(n(2)n(4)) \quad e(n(4)n(5))$$
$$\delta(r) \quad \delta(g) \quad \delta(b)$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(g))\ m(n(5)c(g))\ v(n(3)\,n(4)))$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(g))\ m(n(5)c(b))\ v(n(3)\,n(4)))$$
$$b(l(3)\ m(n(1)c(r))\ m(n(5)c(g))\ m(n(2)c(g))\ v(n(3)\,n(4)))$$
$$b(l(3)\ m(n(1)c(r))\ m(n(5)c(g))\ m(n(2)c(b))\ v(n(3)\,n(4)))$$
$$\vdots$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(b))\ m(n(3)c(r))\ v(n(4)\,n(5)))$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(b))\ m(n(3)c(g))\ v(n(4)\,n(5)))$$
$$l(3)$$

Fig. 8. Set of objects inside the top-level cell after the fifth step for Figure 4. Note that there are some identical objects here which have been created independently.

$$e(n(1)n(2)) \quad e(n(1)n(5)) \quad e(n(2)n(3)) \quad e(n(2)n(4)) \quad e(n(4)n(5))$$
$$\delta(r) \quad \delta(g) \quad \delta(b)$$
$$b(l(4)\ m(n(1)c(r))\ m(n(5)c(g))\ m(n(4)c(b))\ m(n(2)c(r))\ v(n(3)))$$
$$b(l(4)\ m(n(1)c(r))\ m(n(2)c(g))\ m(n(4)c(b))\ m(n(5)c(r))\ v(n(3)))$$
$$b(l(4)\ m(n(1)c(r))\ m(n(2)c(b))\ m(n(4)c(g))\ m(n(5)c(r))\ v(n(3)))$$
$$\vdots$$
$$b(l(4)\ m(n(1)c(r))\ m(n(5)c(g))\ m(n(4)c(r))\ m(n(2)c(g))\ v(n(3)))$$
$$b(l(4)\ m(n(1)c(r))\ m(n(5)c(g))\ m(n(4)c(r))\ m(n(2)c(b))\ v(n(3)))$$
$$\vdots$$
$$b(l(4)\ m(n(1)c(r))\ m(n(2)c(b))\ m(n(3)c(g))\ m(n(5)c(g))\ v(n(4)))$$
$$b(l(4)\ m(n(1)c(r))\ m(n(2)c(b))\ m(n(3)c(g))\ m(n(5)c(b))\ v(n(4)))$$
$$l(3)$$

Fig. 9. Set of objects inside the top-level cell after the sixth step for Figure 4

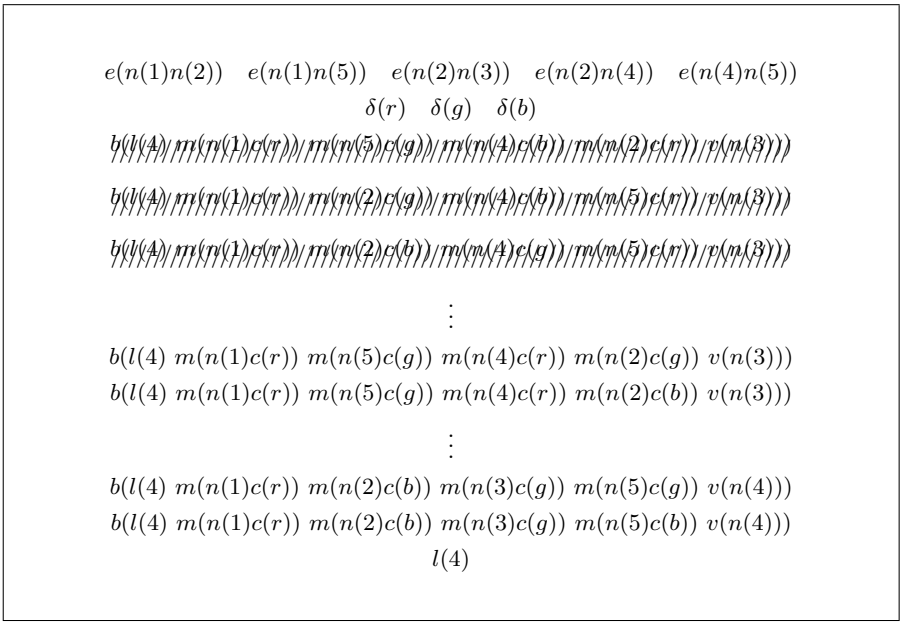


Fig. 10. Set of objects inside the top-level cell after the seventh step for Figure 4

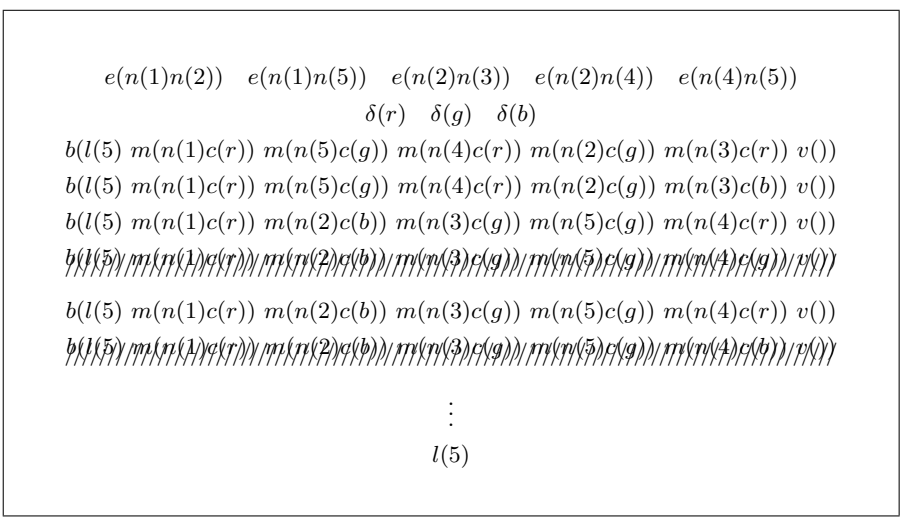


Fig. 11. Set of objects inside the top-level cell after the ninth step for Figure 4

4.2 Failure example

Here we step through the execution of the algorithm when there is no possible valid 3-colouring solution, using the graph depicted in Figure 5. The system begins with the objects depicted in Figure 12.

$$e(n(1)n(2)) \quad e(n(1)n(3)) \quad e(n(1)n(4)) \quad e(n(2)n(3)) \quad e(n(2)n(4)) \quad e(n(3)n(4))$$
$$\delta(r) \quad \delta(g) \quad \delta(b)$$
$$v(n(1)\, n(2)\, n(3)\, n(4)) \quad s(1)$$

Fig. 12. Initial set of objects inside the top-level cell for Figure 5

After the first step, the application of rule 1, the objects shown in Figure 13 are inside the top-level cell. This is not substantively different from the successful example above.

$$e(n(1)n(2)) \quad e(n(1)n(3)) \quad e(n(1)n(4)) \quad e(n(2)n(3)) \quad e(n(2)n(4)) \quad e(n(3)n(4))$$
$$\delta(r) \quad \delta(g) \quad \delta(b)$$
$$b(l(1)\ m(n(1)c(r))\ v(n(2)\, n(3)\, n(4))) \quad l(1)$$

Fig. 13. Set of objects inside the top-level cell at the end of step 1, for Figure 5

At the end of the third step, the objects shown in Figure 14 are in the top-level cell. None of those created in the second step will be removed by rule 5 in the third step, because they all have been derived from the only other node selected so far, and thus were selectively generated by rule 3 such that they would not conflict.

Figure 15 shows the objects in the top-level cell at the end of step 4. Note that in this case half of these, while having been validly generated by rule 3, in fact contain colouring conflicts between two connected nodes and thus will be removed by rule 5 at the end of the next step.

Finally, Figure 16 shows the objects in the top-level cell as at the end of step 6. Due to the fully-connected nature of the graph in Figure 5, every possible solution will contain at least one instance of proposed contiguous colouration,

$$e(n(1)n(2)) \quad e(n(1)n(3)) \quad e(n(1)n(4)) \quad e(n(2)n(3)) \quad e(n(2)n(4)) \quad e(n(3)n(4))$$
$$\delta(r) \quad \delta(g) \quad \delta(b)$$
$$b(l(2)\ m(n(1)c(r))\ m(n(2)c(g))\ v(n(3)\,n(4)))$$
$$b(l(2)\ m(n(1)c(r))\ m(n(2)c(b))\ v(n(3)\,n(4)))$$
$$b(l(2)\ m(n(1)c(r))\ m(n(3)c(g))\ v(n(2)\,n(4)))$$
$$b(l(2)\ m(n(1)c(r))\ m(n(3)c(b))\ v(n(2)\,n(4)))$$
$$b(l(2)\ m(n(1)c(r))\ m(n(4)c(g))\ v(n(2)\,n(3)))$$
$$b(l(2)\ m(n(1)c(r))\ m(n(4)c(b))\ v(n(2)\,n(3)))$$
$$l(2)$$

Fig. 14. Set of objects inside the top-level cell at the end of step 3, for Figure 5

$$e(n(1)n(2)) \quad e(n(1)n(3)) \quad e(n(1)n(4)) \quad e(n(2)n(3)) \quad e(n(2)n(4)) \quad e(n(3)n(4))$$
$$\delta(r) \quad \delta(g) \quad \delta(b)$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(g))\ m(n(3)c(g))\ v(n(4)))$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(g))\ m(n(3)c(b))\ v(n(4)))$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(g))\ m(n(3)c(r))\ v(n(4)))$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(g))\ m(n(3)c(b))\ v(n(4)))$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(g))\ m(n(4)c(g))\ v(n(3)))$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(g))\ m(n(4)c(b))\ v(n(3)))$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(g))\ m(n(4)c(r))\ v(n(3)))$$
$$b(l(3)\ m(n(1)c(r))\ m(n(2)c(g))\ m(n(4)c(b))\ v(n(3)))$$
$$\vdots$$
$$l(2)$$

Fig. 15. Set of objects inside the top-level cell at the end of step 4, for Figure 5

thus making every possible solution invalid. At the end of step 7, rule 5 will eliminate all of these, and leave the system empty of b objects.

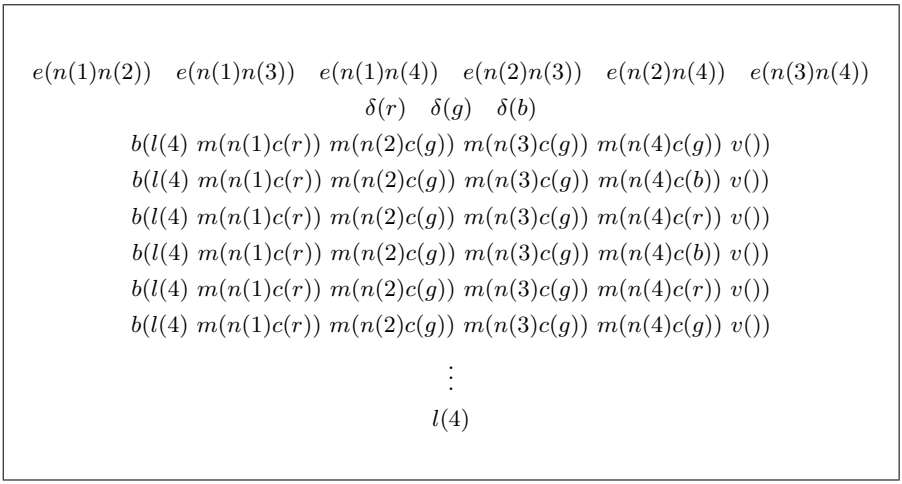


Fig. 16. Set of objects inside the top-level cell at the end of step 6, for Figure 5

After rule 7 finishes, the top-level cell will contain only those objects depicted in Figure 17. With no b or v objects available in the system, none of rules 1-5 will be applicable (in fact, at the end of rule 7 the system would be in state 1, so only rules 2, 3 and 4 would potentially be applicable anyway). Thus, the first rule selected is rule 6, which merely transitions the system to state 4, signalling to the environment that the evolution of the system has ceased after determining that there was no possible valid colouring using the colours provided.

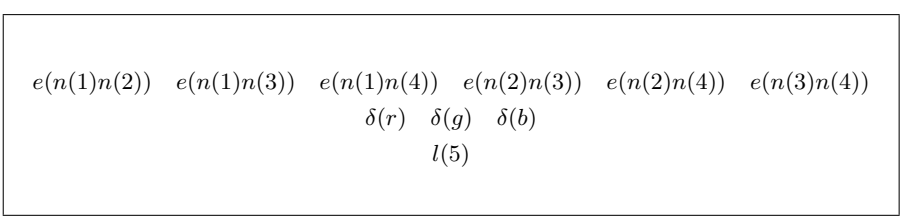


Fig. 17. Set of objects inside the top-level cell at the end of rule 7, for Figure 5

5 Conclusions

Firstly we implemented a version of Gheorghe *et al.*'s Simple Kernel P systems solution to the 3-colouring problem using F# and a Concurrent ML-derived library, Hopac. On the small sample of graphs tested this produces reasonable results for smaller graphs, not entirely dissimilar to those achieved by the P-Lingua simulation. Including a test for already-invalid compartments has the potential to significantly improve the outcomes, as it prunes the search space.

Concurrent ML appears to be a good fit in principle to P systems variants that use communication between cells/membranes, but this particular problem had fairly minimal communication and instead relied heavily upon cell division. Further testing with other problems is required to determine the efficacy of Concurrent ML for sure.

We secondly provided a cP systems solution to the graph colouring problem, which is capable of solving the problem for arbitrary graphs with an arbitrary number of potential colours in at most $2N$ steps. Its operation was demonstrated with two 3-colouring examples based on two different graphs, for which it was and was not possible, respectively, to find a valid 3-colouring.

References

1. Beigel, R., Eppstein, D.: 3-coloring in $O(1.3289^n)$ time. Journal of Algorithms **54**(2), 168–204 (Feb 2005). https://doi.org/10.1016/j.jalgor.2004.06.008
2. Cooper, J., Nicolescu, R.: The Hamiltonian Cycle and Travelling Salesman Problems in cP Systems. Fundamenta Informaticae **164**(2-3), 157–180 (2019). https://doi.org/10.3233/FI-2019-1760
3. Csuhaj-Varjú, E., Verlan, S.: On generalized communicating P systems with minimal interaction rules. Theoretical Computer Science **412**(1-2), 124–135 (Jan 2011). https://doi.org/10.1016/j.tcs.2010.08.020
4. Gheorghe, M., Ipate, F., Lefticaru, R., Pérez-Jiménez, M.J., Ţurcanu, A., Valencia Cabrera, L., García-Quismondo, M., Mierlă, L.: 3-Col problem modelling using simple kernel P systems. International Journal of Computer Mathematics **90**(4), 816–830 (Apr 2013). https://doi.org/10.1080/00207160.2012.743712
5. Lewis, R.M.R.: Introduction to Graph Colouring, pp. 1–25. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-25730-3_1
6. Panangaden, P., Reppy, J.: The Essence of Concurrent ML, pp. 5–29. Springer New York, New York, NY (1997). https://doi.org/10.1007/978-1-4612-2274-3_2
7. Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Tissue P Systems with Cell Division. International Journal of Computers Communications & Control **3**(3), 295 (Sep 2008). https://doi.org/10.15837/ijccc.2008.3.2397
8. Reppy, J.H.: CML. ACM SIGPLAN Notices **26**(6), 293–305 (Jun 1991). https://doi.org/10.1145/113446.113470
9. Reppy, J.H.: Concurrent Programming in ML. Cambridge University Press, New York, New York, USA (2007)
10. Song, B., Pérez-Jiménez, M.J., Păun, G., Pan, L.: Tissue P Systems With Channel States Working in the Flat Maximally Parallel Way. IEEE Transactions on NanoBioscience **15**(7), 645–656 (Oct 2016). https://doi.org/10.1109/TNB.2016.2594380

11. Verlan, S.: Tissue P Systems with Minimal Symport/Antiport. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) Developments in Language Theory. pp. 418–429. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30550-7_35

cP systems Solution of the Santa Claus Problem

(extended abstract)

Alec Henderson, Ocean Wu, Michael J. Dinneen, and Radu Nicolescu

Department of Computer Science
The University of Auckland
Auckland, New Zealand

Abstract. We briefly summarize a few known solutions to a classic and challenging concurrency problem, called the Santa Claus Problem. We then introduce cP systems and provide an elegant solution to this problem. Finally, we evaluate this novel solution and mention the importance of formal verification.

1 Introduction

As it is now often repeated, the free lunch is over—we can no longer make our code run faster by simply waiting for faster hardware. Modern high performance systems need to exploit concurrency, yet writing concurrent code can easily introduce bugs and complexities, making people spend more time worrying about and satisfying the computer rather than solving the domain problem.

The Santa Claus Problem is a classic concurrency challenge, introduced by Trono [11] in 1994:

> Santa Claus sleeps at the North Pole until awakened by either all of the nine reindeer, or by a group of three out of n elves (typically $n = 10$, or a larger number). He then performs one of two indivisible actions:
> - If awakened by the group of nine reindeer, Santa harnesses them to a sleigh, delivers toys, and finally unharnesses the reindeer who then go on vacation.
> - If awakened by a group of three elves, Santa shows them into his office, consults with them on toy R&D, and finally shows them out, so they can return to work on constructing toys.
>
> A waiting group of reindeer must be served by Santa before a waiting group of elves. Since Santas time is extremely valuable, marshaling the reindeer or elves into a group must not be done by Santa.

The problem seems simple at first sight, but has become a tough testbed for the expressiveness of concurrency constructs. As noted by Benton [3], attempts to solve this problem can easily introduce errors such as:

- Santa takes off to deliver toys, while one or more (possibly all nine) reindeer may still be waiting to be harnessed.

– Santa ends its consulting time and goes to sleep, while one or more (possibly all three) elves haven't yet asked their questions.
– One or more additional elves sneak into the consultation room after Santa invites the group of three who have initially registered for consultation.
– The priority rule is quite tough to fully implement. There is frequently a (possibly very narrow, but not null) opportunity for a group of three elves to get Santa's attention even when all reindeer have returned and are ready to be harnessed.

Trono's own first solution [11], based on semaphores, highlighted all the above problems. Subsequent solutions tried to remedy this, but even recent solutions may exhibit some of the above problems, or else are excessively complex and fragile (qualifying them as "messy" would not be a great exaggeration). While primitive constructs such as semaphores can successfully model simple scenarios, they seem less adequate for high-level models of complex scenarios.

Therefore, the Santa Claus Problem has become a great testbed for testing the expressiveness of numerous concurrency constructs and models, such as: semaphores, monitors, locks, barriers, select-type constructs, guards, message passing (cf. MPI [5]), software transactional memory (STM) [7], actors and actor extensions (multiple heads, multiple mailboxes [10]), rendezvous (cf. Ada [2]), communicating sequential processes (CSP, cf. occam [1]), join calculus (cf. Polyphonic C# [3]), etc. The next section offers a brief overview of some of these proposals.

Essentially, there are two distinct problems that must be properly solved, cf. Ben-Ari [2]: (1) how to synchronize a set of processes (here elves or reindeer) and release them as a group; (2) how one process (e.g. Santa) provides more than one distinct service, with different priorities.

2 Overview

The shared memory model implemented in the form of threads is usually adopted as the first attempt to solve any concurrency challenges, since the thread libraries are conveniently available in many popular programming languages such as C/C++, Java, and C#. In the shared memory model, the program is split into two or more tasks running on different threads which operate on shared data, and the underlying operating system is in charge of scheduling and dispatching these threads for execution. The first solution to the Santa Claus Problem given by Trono utilizes such construct with semaphores [11]. However, Trono's solution is only partially correct because it "assumes that a process released from waiting on a semaphore will necessarily be scheduled for execution" [2]. Hurt and Pedersen [6] implemented and compared a few different solutions using barriers, locks, semaphores, mutexes, and monitors. However, all these constructs suffer from various problems such as race conditions, deadlocks, and livelocks. Besides, it is a non-trivial task to prove the correctness, as all possible interleavings of program execution have to be considered.

Message Passing Interface (MPI) [5] is a message-passing standard for a distributed memory architecture e.g. a cluster. Most MPI implementations consist of a specific set of API directly callable from C\C++, Fortran and any languages able to interface with such libraries, including C#, Java or Python. In MPI, each of a program's tasks is separated onto a different process, mimicking a distributed memory model. The *MPI_Barrier* method can be used to synchronize a group of processes, and synchronous receives replaces wait/notify to simply the asynchronous logic. These techniques can be used to synchronize the individual Santa, elf, and Reindeer processes [6]. Due to the lack of shared memory in MPI, a separate process for each of the Reindeer and elf waiting groups are needed. When a process receives a pieces of data, it processes the data and then pass it onto one or multiple processes. Therefore the same piece of data is never operated on by more than one process at a time, eliminating the data race problem plagued by the shared memory model [6].

Jones [7] proposed another solution using the Software Transactional Memory (STM) model in Haskell. STM enables people to write programs in a more modular way i.e. building large programs by gluing smaller programs together, without needing to expose their implementations [7]. The STM solution to the Santa Claus Problem introduces the abstraction of a *Gate* and a *Group* for modularity and ease of action coordination. Santa creates two groups, one for the elves and the other for the reindeer. The elves and the reindeer try to join their group if needing to wake Santa up. Santa controls the Gates for marshalling the elves and the reindeer [7]. The shared memory model makes such modularity impossible: any newly added actions, which operate on the shared data, need to be carefully coordinated by synchronizing constructs, such as locks and conditional variables, to avoid data races.

The actor model is gaining momentum in solving concurrency challenges. It provides high-level concurrency abstractions via message passing. Each actor comes with a queue, called a mailbox, for storing any incoming messages. Actors communicate by sending and receiving messages. Sending is an asynchronous operation, meaning any actors can send any number of messages at a time. Receiving is a synchronous operation, meaning each of the actors can only process one message at a time. Sending and Receiving of a message are usually implemented by pattern matching. However, as noted by Sulzmann et al. [10], the classic actor model implemented in languages such as Erlang is restricted to a *single-headed* message pattern—that is, each receive only matches one message at a time. This restriction makes the priority in the Santa Claus Problem a non-trivial task to implement, since the messages coming from the three elves might come before the nine reindeer, and the receive procedure can only match the messages that arrive first. To address this limitation, Sulzmann et al. [10] proposed and designed an extension of Erlang style actors with receive clauses containing multi-headed message patterns i.e. matching multiple messages at a time, extending the ability to easily express complex synchronization patterns.

Ben-Ari [2] highlighted the errors in the original Trono's paper [11], discussed various problems that low-level semaphore-based solutions may have, and pro-

posed a high-level solution in Ada. This solution uses Ada's unique *rendezvous* construct, inspired from join calculus. It also uses the more recent *protected object* construct, essentially an extension of Hanson's monitors. The solution is fairly neat and covers all requirements well, except the last and tricky one, to ensure that reindeer have priority over elves. This priority requires a subtle interaction between more sophisticated concepts, such as *requeuing* tasks, using additional accept (i.e. select) *guards*, and finally still needs a *pragma* requiring a textual order queuing policy.

Benton [3] proposed a very neat solution, possibly the most elegant, based on Polyphonic C#, a fairly direct implementation of join calculus ideas [4] in the C# programming language. It solves most problems by extensively using the new *chord* concept, which allows one to associate the header and body of one *synchronous* function with one or more *asynchronous* function headers. Calls to a chord only succeed when all functions have been called, i.e. the synchronous base and all additional asynchronous headers. Calling the synchronous entry proceeds synchronously, as expected, possibly *waiting* for all associated asynchronous calls. Calling an asynchronous entry does *not block*; if needed, the parameters of the call are implicitly *queued* in system provided "message" queue (like in actor systems). Space does not allow us to delve deeper in this topic here, but we illustrate these concepts by the exceptionally simple implementation of a semaphore, cf. Fig. 1. Still, unless the compiler and runtime ensure a textual order for chord matching, implementing the required priority is not simple and requires a few non-trivial additions.

```
public async Signal() & public void Wait() {}
```

Fig. 1. Semaphore in Polyphonic C#.

3 A Membrane Computing Model with Structured Data

For this paper, we will use a structured model of P systems, called cP systems, that allow for "compound" structured data, like lists and annotated objects (functors) instead of being restricted to the traditional basic data type consisting of only multi-sets of objects. See [8] for a lengthy introduction to cP systems.

These cP systems are an extension of traditional cell-like (tree-based) and tissue (graph-based) P systems. They consist of state-based rewriting rules with inter-cell communication. Additionally, intra-cell parallelism is supported where the rules run in a weak (i.e. top-down) priority mode.

We now highlight from [8] the main features of cP systems that we use:

– Top-level cells (distributed nodes) are organised in digraph networks and are differentiated by unique IDs.

- Each arc represents an unidirectional communication channel and each has two (not necessarily distinct) labels for the source and target cells.
- Top-level cells contain nested (and labelled) sub-cells, with the sub-cells (i.e. *compound terms*) representing local data that can be processed either sequentially or in the max-parallel mode.
- The evolution is governed by multiset rewriting+communication rules, running in exactly-once ($\rightarrow_1$) or max-parallel ($\rightarrow_+$) modes.
- Only top-level cells have evolution rules.
- In a synchronous evolution, each internal step takes zero time, and each message takes exactly one time unit to transit from source to target.
- In an asynchronous evolution, each message may take any finite real time to transit from source to target.
- Top-level cells send messages—via the right-hand-side of rewriting rules, using symbol '!' —over outgoing arcs, to their structural neighbours.
- Top-level cells receive messages—via the left-hand-side of rewriting rules, using symbol '?'—over incoming arcs, from their structural neighbours.
- Messages which arrive at the target cell are not immediately inserted among the target's contents; instead, these messages are conceptually "enqueued" and there is one message "queue"—read "multiset"—for each incoming arc.
- Receiving cells have full control over the time and format of messages accepted from the message "queues".
- Messages not yet accepted remain in their message "queues".

The basic grammar of cP systems is shown in Fig. 2. We use standard conventions: the symbol λ denotes the empty multiset, numbers are represented in base 1 (lists of special symbol '1', with length representing its value), *atoms* denoted by lower case characters, *variables* denoted by uppercase letters, *functors* are sub-cell labels with their arguments enclosed in parentheses '()' and '$\ldots'$ represents zero or more repetitions.

In the grammar given in Fig. 2, we use $\alpha \in \{1, +\}$ to represent application of the rule is in *exactly-once* or *max-parallel* mode. The symbol δ (for communication channels of the rules) denotes either a specific node label, set of node labels, or *wildcard* symbol '_', where the data comes from/to.

Rules are evolved using pattern matching unification of atoms to variables. Only if all conditions of the left-hand-side '<lhs>' of a rule must match before a rule can be executed (in addition, to the promoter and inhibitor constraints). We want to point out that cP rules allow algorithm descriptions with *fixed-size alphabets* and *fixed-sized rulesets*, independent of the size of the problem and number of cells in the system.

Examples of rules

1. Change state from s_0 to s_1 and rewrite one pair of a and b into one c, provided that at least one p is present (and stays in the cells):

$$s_0 \;\; a \;\; b \;\; \rightarrow_1 \;\; s_1 \;\; c \;\; | \;\; p$$

```
<rule> ::= <lhs> →α <rhs> <promoters> <inhibitors>
<lhs> ::= <state> (<term> | ?δ '{' <term> '}')...
<rhs> ::= <state> (<term> | !δ '{' <term> '}')...
<state> ::= <atom>
<promoters> ::= ('|' <term-or-eq>)...
<inhibitors> ::= ('¬' <term-or-eq>)...

<term> ::= <simple-term> | <compound-term> | <number> | <list>
<term-or-eq> ::= <term> | '(' <term> '=' <term> ')'
<simple-term> ::= <atom> | <variable>
<compound-term> ::= <functor> '(' <argument> ')'

<functor> ::= <atom>
<argument> ::= <term>...
<number> ::= 1...
<list> ::= '[]' | '[' <head> '|' <list> ']'
<head> ::= <term>...
```

Fig. 2. Basic cP system rule grammar.

2. Change state from s_0 to s_1 and rewrite all a, b pairs into c's, in the max-parallel mode, provided that at least one p is present:

$$s_0\ a\ b \rightarrow_+ s_1\ c \mid p$$

3. If there is an a coming from channel ι, then accept it ("read" it), change state from s_0 to s_1, rewrite the incoming a and one local b into one c, while sending one d over channel ω:

$$s_0\ ?_\iota\{a\}\ b \rightarrow_1 s_1\ c\ !_\omega\{d\}$$

4. Change state from s_0 to s_1, rewrite one compound symbol $a()$ by adding one *1* to its contents; variable X is unified to the actual contents of a.

$$s_0\ a(X) \rightarrow_1 s_1\ a(X1)$$

If the current a already has two copies of *1*, i.e. $a(11)$, the result will be an updated copy with three *1*'s, i.e. $a(111)$—thereby *incrementing* its base 1 contents.

5. Change state from s_0 to s_1, rewrite one compound symbol $a()$ by removing one *1* of its contents, if there is at least one *1* among its contents.

$$s_0\ a(Y1) \rightarrow_1 s_1\ a(Y)$$

If the current a already has three copies of *1*, i.e. $a(111)$, the result will be an updated copy with two *1*'s, i.e. $a(11)$—thereby *decrementing* its base 1 contents.

6. A complex operation, highlighting the weak priority order, with resulting state depending on the current cell contents.

$s_0\ a \rightarrow_1 s_1\ e\ (1)$
$s_0\ b \rightarrow_1 s_2\ f\ (2)$
$s_0\ c \rightarrow_1 s_1\ g\ (3)$

(a) If the cell contains a and c, then rules (1) and (3) apply; new state: s_1, new contents: e and g.
(b) If the cell contains b and c, then only rule (2) applies; new state: s_2, new contents: f and c. Rule (3) is not applicable, because rule (2) has already set the target state to s_2.
(c) If the cell contains a, b and c, then only rules (1) and (3) apply; new state: s_1, new contents: e, b, and g. Rule (2) is not applicable, because rule (1) has already set the target state to s_1.

4 Santa System

The cP system used to solve the Santa Claus Problem is described using five cell types:

- One Santa cell, denoted κ.
- One elf registry cell, denoted $\bar{\epsilon}$.
- One reindeer registry cell, denoted $\bar{\rho}$.
- Nine reindeer cells, denoted $\rho_1, \rho_2, ...\rho_9$.
- n elf cells, denoted $\epsilon_1, \epsilon_2, ...\epsilon_n$.

An example graph of the system for two reindeer, and two elves is shown in Fig. 3. A diagram of the message senders and receivers is shown in Fig. 4.

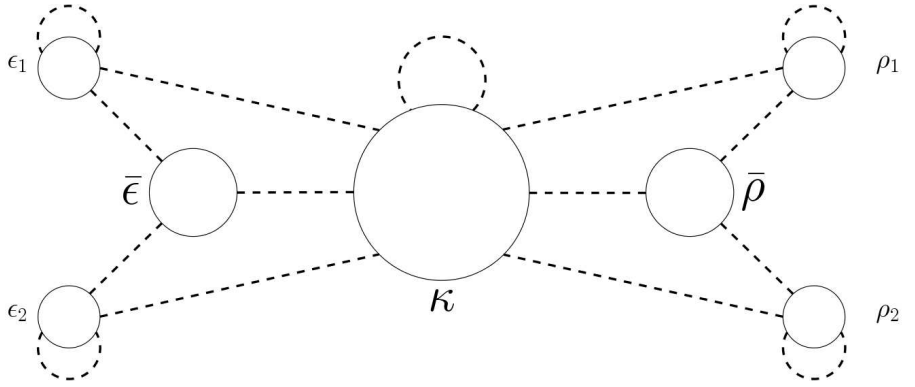


Fig. 3. cP system graph, for two reindeer and two elves. The dashed lines represent connections between the cells.

The system messages can be interpreted as:

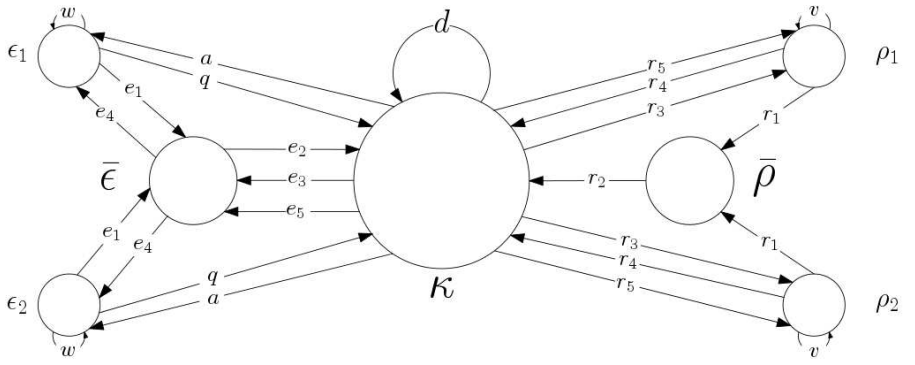


Fig. 4. An overview of the cP system and messages sent.

- w means that the elves are working. Due to the system being *asynchronous*, the message w can take any length for the different elves. For example Fig. 5 we show two elves sending messages, the two messages could arrive at very different times. This is why all arbitrary length jobs are simulated using self-addressed messages.
- v is a similarly self-addressed message which simulates the time the reindeer spend on vacation.
- d is a similarly self-addressed message which simulates the time that Santa spends delivering the presents.
- e_1 sent by an elf to tell the registry that they want to consult with Santa.
- e_2 sent from the elf registry, to Santa, to tell him a group of three elves are ready to consult.
- e_3 from Santa to the elf registry to say he is ready to consult the group of elves.
- e_4 which tells elves that they can now consult with Santa, sent from the elf registry.
- e_5 from Santa to the elf registry to tell the registry he is finished with the most recent group of elves.
- q from an elf to Santa representing the elf's part of the consulting process.
- a from Santa to the elf that asked a question represents his part of the consulting process.
- r_1 from a reindeer to the reindeer registry represents the reindeer finishing it's vacation.
- r_2 from the reindeer registry to Santa represents when all nine reindeer have finished vacation.
- r_3 represents Santa harnessing the reindeer.
- r_4 represents that a reindeer is harnessed and ready to deliver.
- r_5 represents Santa telling the reindeer to unharness and go on holiday once again.

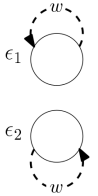


Fig. 5. Two elves sending their self-addressed w messages.

The rulesets for each cell type are in Figs. 6 to 10, respectively, where each rule is numbered by the cell symbol followed by a dot (".") and then the rule index. For example, the reindeer's second rule is listed as $\rho.2$.

$S_0\ ?_\rho\{r_2\}$	$\to_1$	S_1	$(\kappa.1)$
S_1	$\to_+$	$S_2\ !_R\{r_3\}\ \ \pi(9)\ \mid\ \rho(R)$	$(\kappa.2)$
$S_2\ \ ?_R\{r_4\}\ \ \pi(X1)$	$\to_1$	$S_2\ \ \pi(X)$	$(\kappa.3)$
$S_2\ \ \pi()$	$\to_1$	$S_2\ \ !_\kappa\{d\}$	$(\kappa.4)$
$S_2?_k\{d\}\ \ \pi()$	$\to_1$	S_3	$(\kappa.5)$
S_3	$\to_+$	$S_0\ \ !_R\{r_5\}\ \mid\ \rho(R)$	$(\kappa.6)$
$S_0\ ?_\epsilon\{e_2\}$	$\to_1$	$S_3\ \ !_\epsilon\{e_3\}\ \ \pi(3)$	$(\kappa.7)$
$S_3\ ?_E\{q\}\ \ \pi(X1)$	$\to_1$	$S_3\ \ !_E\{a\}\ \ \pi(X)$	$(\kappa.8)$
$S_3\ \pi()$	$\to_1$	$S_0\ \ !_\epsilon\{e_5\}$	$(\kappa.9)$

Fig. 6. Ruleset for the Santa cell, κ.

$S_0\ \ r(9)$	$\to_1$	$S_0\ \ !_\kappa\{r_2\}\ \ r()$	$(\bar{\rho}.1)$
$S_0\ \ ?_R\{r_1\}\ \ r(X)$	$\to_1$	$S_0\ \ r(X1)$	$(\bar{\rho}.2)$

Fig. 7. Ruleset for the reindeer registry, $\bar{\rho}$.

$$
\begin{array}{llll}
S_0 & \rightarrow_1 & S_1 \;\; !_\kappa\{e_2\} \;\mid\; e(3) & (\bar{\epsilon}.1) \\
S_0 \;\; ?_E\{e_1\} \;\; e(X) & \rightarrow_1 & S_0 \;\; \omega(E) \;\; e(X1) & (\bar{\epsilon}.2) \\
S_1 \;\; ?_\kappa\{e_3\} & \rightarrow_1 & S_2 & (\bar{\epsilon}.3) \\
S_2 \;\; \omega(E) & \rightarrow_+ & S_3 \;\; !_E\{e_4\} & (\bar{\epsilon}.4) \\
S_3 \;\; ?_\kappa\{e_5\} \;\; e(X) & \rightarrow_1 & S_0 \;\; e() & (\bar{\epsilon}.5)
\end{array}
$$

Fig. 8. Ruleset for the elf registry, $\bar{\epsilon}$.

$$
\begin{array}{llll}
S_1 \;\; ?_I\{v\} & \rightarrow_1 & S_2 \;\; !_\rho\{r_1\} & (\rho.1) \\
S_0 & \rightarrow_1 & S_1 \;\; !_I\{v\} \;\mid\; \theta(I) & (\rho.2) \\
S_2 \;\; ?_\kappa\{r_3\} & \rightarrow_1 & S_3 \;\; !_\kappa\{r_4\} & (\rho.3) \\
S_3 \;\; ?_\kappa\{r_5\} & \rightarrow_1 & S_0 & (\rho.4)
\end{array}
$$

Fig. 9. Ruleset for a reindeer cell, ρ.

$$
\begin{array}{llll}
S_1 \;\; ?_I\{w\} & \rightarrow_1 & S_2 \;\; !_\epsilon\{e_1\} & (\epsilon.1) \\
S_0 & \rightarrow_1 & S_1 \;\; !_I\{w\} \;\mid\; \theta(I) & (\epsilon.2) \\
S_2 \;\; ?_\epsilon\{e_4\} & \rightarrow_1 & S_3 \;\; !_\kappa\{q\} & (\epsilon.3) \\
S_3 \;\; ?_\kappa\{a\} & \rightarrow_1 & S_0 & (\epsilon.4)
\end{array}
$$

Fig. 10. Ruleset for an elf cell, ϵ.

Initial configuration We assume that all reindeer and elf cells contain $\theta(x)$ where x is the unique identifier of the cell (in our case numbers). We also assume that Santa contains $\rho(x_1), \rho(x_2), ...\rho(x_9)$, where $x_i's$ are the unique reindeer identifiers.

Example

We illustrate the system evolution with a simple scenario with *two* reindeer and *two* elves. Obviously, this does not fully describes all possible concurrency issues.

However, it presents an intuitive description of the rule set. See Fig. 3 for the cP graph and Fig. 4 for an overview of the message senders and receivers.

Initially the elves are working and the reindeer are on vacation, this is denoted in the rule set by rules $\epsilon.2$ and $\rho.2$. Where the time working and on vacation is represented by the time it takes for a self-addressed message to arrive. When an elf receives this self-addressed message, he knows that he needs to consult with Santa, and so tells the elf registry that he is ready to see Santa (rule $\epsilon.1$). Similarly when the reindeer receives the message, they know they are rested enough to deliver presents and inform the reindeer registry (rule $\rho.1$).

Once a group of three elves have all told the registry that they are ready, the registry tells Santa to wake up and help (rule $\bar{\epsilon}.1$). Santa then tells the elf registry that he is ready to see the elves (rule $\kappa.7$). He also sets his question and answer counter to be three. The registry then tells everyone in the group that Santa is ready for questions (rule $\bar{\epsilon}.4$).

When an elf receives the message that Santa is ready for questions, they send a question to Santa (rule $\epsilon.3$). Santa answers in turn each of the elf's questions, decrements his question and answer counter until it becomes zero (rule $\kappa.8$). When an elf gets an answer, they then return to work (rules $\epsilon.4$ and $\epsilon.2$). When Santa's question and answer counter is zero, he tells the elf registry that he is finished with the current group, and returns to sleeping (rule $\kappa.9$).

When the reindeer registry has been informed by all the reindeer that they are ready to deliver the presents, the registry tells to Santa wake up and deliver the presents (rule $\bar{\rho}.1$). Santa wakes up and then harness all of the reindeer (rules $\kappa.1$ and $\kappa.2$). When a reindeer is harnessed and ready, they confirm with Santa (rule $\rho.3$). When all of the reindeer are harnessed and ready to deliver, Santa delivers all of the presents (rule $\kappa.4$). Delivery time is represented by a self-addressed message. When Santa finishes delivery, he tells the reindeer to go on vacation (rules $\kappa.5$ and $\kappa.6$) and returns to sleeping. When the reindeer receive the message to go on vacation, they do as they are advised (rules $\rho.4$ and $\rho.1$).

Possible pitfalls

As mentioned previously there are four common pitfalls in solutions to the Santa Claus Problem.

- Santa takes off to deliver toys, while one or more (possibly all nine) reindeer may still be waiting to be harnessed.

Our solution to this is that Santa sends a message (r_3) to all of the reindeer to get harnessed. The reindeer once harnessed, respond with a message (r_4). Santa does not start to deliver presents until, he has received confirmation from all of the reindeer (counted nine r_4's).

- Santa ends his consulting and goes to sleep, while one or more (possibly all three) elves have not yet asked their questions.

When Santa starts consulting with the elves, he starts a question and answer counter. Santa will not finish consulting until, he has received questions from the entire group. This ensures that the entire elf group has asked there questions, before Santa will sleep again. Otherwise, his counter is greater than 0, and would still be consulting.

– One or more additional elves sneak into the consultation room after Santa invites the group of three who have initially registered for a consultation.

Elves are invited into the consulting room by the elf registry (rule $\bar{\epsilon}.4$). The elf registry is only able to add three elves to the set of elves before it changes state (rules $\bar{\epsilon}.1$ and $\bar{\epsilon}.2$), and no longer accepts more to the group.

– The priority rule is quite tough to fully implement. There is frequent opportunity (possibly very narrow, but not null) for a group of three elves to get Santa's attention even when all reindeer have returned and are ready to be harnessed.

Although difficult in many programming languages, implementing the preference in cP is achieved simply via the *weak order priority* of the rules of Santa.

5 Conclusions

We think that our cP solution compares very favourably with other extant solutions, in term of elegance and conciseness. We should also stress that our system is completely formal, directly executable (in principle), and – in contrast to all other solutions – is totally self contained (does not require specialised libraries). Table 1 from [6] summarizes a simple but useful complexity measure, lines of code (LOC). In our case, we have a total of 24 rules, roughly corresponding to 24 LOC.

Table 1. LOC for various Santa Claus solutions, cf. [6]

			SM		DM	PO
	C#	C	Java	Groovy	MPI	JCSP
Total	642	420	564	315	352	315
Synchronisation/Communication	48	49	46	46	34	27
Prevent Race Condition	14	8	8	8	N/A	N/A
Exception/Error Handling	35	0	177	18	41	0
Custom Barrier Implementation	42	35	N/A	N/A	N/A	55
GUI	145	N/A	N/A	N/A	N/A	N/A

The runtimes of different solutions varies very widely. Table 2 from [9] provides some interesting timing figures, but is far from complete. We are working

Table 2. Runtimes of various Santa Claus solutions, in the real/user/system format, cf. [9]

Repetitions of Santa	Lime(guards)	C(semaphores)	Go(channels)	Java (monitors)
10,000	0.04/0.04/0.00	0.87/0.26/1.18	0.08/0.12/0.01	6.38/2.48/5.30
100,000	0.30/0.30/0.00	8.82/2.50/12.0	0.77/1.18/0.06	60.3/21.6/52.0
1,000,000	2.91/2.90/0.01	93.0/24.8/234	75.1/11.6/0.55	≈534/159/509

to provide a more uniform fair testbed, which should also consider a direct translation of our model into an existing or adapted open source runtime.

Last, but not least, this exercise suggests that complex concurrency problems should be verified with formal methods or tools. The Santa Claus Problem highlights how easily one (even experts) can make mistakes while modelling complex concurrency scenarios; convincing intuitions and explanations are not enough. This is a theme for further research.

References

1. occam 2.1 reference manual. http://www.wotug.org/occam/documentation/oc21refman.pdf, 1995.
2. M. Ben-Ari. How to Solve the Santa Claus Problem. *Concurrency: Practice and Experience*, 10(6):485–496, 1998.
3. N. Benton. Jingle Bells: Solving the Santa Claus Problem in Polyphonic C#. *Unpublished manuscript*, Mar 2003.
4. N. Benton, L. Cardelli, and C. Fournet. Modern concurrency abstractions for c#. In B. Magnusson, editor, *ECOOP 2002 — Object-Oriented Programming*, pages 415–440, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
5. M. P. Forum. MPI: A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.
6. J. Hurt and J. B. Pedersen. Solving the Santa Claus Problem: A comparison of various concurrent programming techniques. *Communicating Process Architectures 2008: WoTUG-31*, 66:381, 2008.
7. S. P. Jones. Beautiful Concurrency. *Beautiful Code: Leading Programmers Explain How They Think*, pages 385–406, 2007.
8. R. Nicolescu and A. Henderson. An introduction to cP systems. In C. G. Díaz, A. Riscos-Núñez, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Enjoying Natural Computing - Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday*, volume 11270 of *Lecture Notes in Computer Science*, pages 204–227. Springer, 2018.
9. E. Sekerinski and S. Yao. Refining santa: An exercise in efficient synchronization. In J. Derrick, B. Dongol, and S. Reeves, editors, Proceedings *18th Refinement Workshop,* Oxford, UK, 18th July 2018, volume 282 of *Electronic Proceedings in Theoretical Computer Science*, pages 68–86. Open Publishing Association, 2018.
10. M. Sulzmann, E. S. Lam, and P. Van Weert. Actors with multi-headed message receive patterns. In *International Conference on Coordination Languages and Models*, pages 315–330. Springer, 2008.

11. J. A. Trono. A new exercise in concurrency. *ACM SIGCSE Bulletin*, 26(3):8–10, 1994.