Abstract Tissue-like P systems can be viewed as distributed and parallel bio-inspired computing devices, where the objects in each region can communicate using corresponding channels, and infinitely many objects in the environment can provide substances for cells. In this work, inspired by the biological phenomenon of polarization on cells of actual tissues, we propose a new variant of tissue-like P systems, namely, tissue-like P systems with polarizations ($\mathcal{TPP}$ systems), in which the execution of rules may be influenced by polarizations on cells, and such systems run under flat maximal parallelism. In addition, cell division rules with polarizations are introduced into this variant, and we conclude that the computational power of a $\mathcal{TPP}$ system is equivalent to that of a Turing machine in terms of generating numbers, where the maximum lengths of symport rules and antiport rules are 2 and 4, respectively, or the maximum length of symport rules is 4 (if only this type of rule is used). In both cases, only one cell and three types of polarizations are employed. Moreover, due to cell division rules, $\mathcal{TPP}$ systems can solve the SAT problem (a classical **NP**-complete problem) in polynomial time, where the maximum rule length is 4. These results show that the variant can be employed as a distributed parallel computing paradigm. Hence, it is theoretically feasible to apply this variant in some specific applications that require precise rule control.

Keywords Membrane computing, Tissue P system, Polarization, Universality, **NP**-complete problem

1 Introduction

Natural computing refers to a computational paradigm inspired by biological mechanisms in nature. To date, various essential algorithms of natural computing have been inspired by group behavior, such as particle swarm optimization, the artificial bee colony algorithm and the ant colony algorithm. In addition, well-known branches of natural computing include DNA computing [1], membrane computing [2], evolutionary computing [3] and neural networks [4]. As a new branch of natural computing, membrane computing was initiated in 1998 and is inspired by biological living cells. Membrane computing has been well developed, and it has formed a multidisciplinary research hotspot involving computer science, mathematics and biology. Theoretically, many variant of P systems are Turing universal [5–14]; namely, their computational power is equivalent to that of a Turing machine. Moreover, many variants of P systems can solve some **NP**-complete problems in polynomial time [15–19]. Many scholars have investigated the

application of membrane computing in diverse fields to address various real-world problems, e.g., optimization algorithms [20, 21], mobile robots [22], machine learning [23], biology [24, 25] and fault diagnosis [26, 27]. For additional details, some review papers including books (e.g., [28] and [29]) and the website http://ppage.psystems.eu/ can be viewed to obtain the latest information.

At present, membrane computing focuses primarily on the following categories: cell-like P systems [2] (inspired by living cells, and the structure can be abstracted as a rooted tree, where the root of the tree is the skin membrane), tissue-like P systems [30] (inspired by living cells, and the structure can be abstracted as a graph, where the nodes of the graph represent cells) and neural-like P systems [31] (inspired by neurons, and the structure can be abstracted as a graph, where the nodes of the graph represent neurons). This work primarily considers tissue-like P systems, which are motivated by cooperation between cells in biological organisms. In such systems, cells are viewed as vertices, and channels between different regions are viewed as edges, where objects can communicate between regions (including cells and the environment). Usually, such a paradigm comprises three elements: the cell structure, objects and rules, where the rules consist of antiport rules and symport rules. When antiport rules are used, objects in different regions can exchange their locations; for symport rules, in contrast, objects located in a region can directly change their own locations. During the execution of the above rules, objects located in the environment can be provided for cells with an arbitrary number. However, the structure of such a system is not changed. To solve computationally hard problems, cell division [32] and cell separation [33] are introduced to obtain an exponential workspace. Specifically, some **NP**-complete problems can be solved efficiently, e.g., the 3-coloring problem [16], the travelling salesman problem [34], and the SAT problem [7].

In recent years, based on tissue-like P systems, many new variants have been proposed [5, 7, 35, 36], which are inspired by biological phenomena or developed from the perspective of other areas (e.g., mathematics and computers). In [37], a novel variant of P systems with active membranes was proposed, where each membrane has a polarization. The application of rules in this variant is associated with the polarization on a corresponding membrane. In addition, inspired by the phenomenon that biological neurons also have polarizations, scholars have proposed a novel paradigm, namely, spiking neural P systems with polarizations [12], in which the concept of "polarizations" is introduced into spiking neural P systems, and rules are controlled by the polarizations of neurons. In these proposed variants, there are three categories of polarizations: positive (denoted by "+"), negative (denoted by "−"), and neutral (denoted by "0"). Since polarizations exist in cells and tissues consist of cells, it is natural to introduce polarizations into tissue-like P systems.

In [38], tissue P systems with cell polarity was presented; in such a variant, the term "polarity", however, does not mean the presence of polarizations on cells. In [39], a variant of tissue-like P systems was proposed, where each symbol and cell has a polarization, working on vesicles of multisets with the operations of insertion, deletion, and substitution of single objects. In this work, from the biological perspective, polarizations may exist in actual biological tissues, such as epithelial cells; hence, we introduce polarizations into the original tissue-like P systems, thereby constructing a novel

variant, namely, tissue-like P systems with polarizations.

The main contributions of this paper are as follows:

(i) Inspired by the phenomenon of cell polarization in biological tissues, we introduce the notion of polarization into tissue-like P systems, thereby constructing tissue-like P systems with polarizations, where polarizations have a crucial influence on the execution of rules; moreover, flat maximal parallelism is introduced as the strategy for applying rules;

(ii) Currently, the computational power of a tissue-like P system with a rule length of at most 2 is restricted; however, we deploy $\mathcal{TPP}$ systems to obtain the same computational power as a Turing machine, where antiport rules involve at most two objects and symport rules involve at most one object; besides, the Turing university of $\mathcal{TPP}$ systems is also obtained by employing symport rules involving at most two objects;

(iii) $\mathcal{TPP}$ systems cannot solve **NP**-complete problems (assuming $\mathbf{P} \neq \mathbf{NP}$) in polynomial time without using cell division. Hence, we introduce cell division rules with polarizations to solve the classical SAT problem, passing from non-efficiency to efficiency, by combining antiport rules and symport rules, where the total number of objects participating in the rules is 2;

(iv) The strategy of applying rules with polarizations is introduced into tissue-like P systems, which provides a more flexible and accurate approach for using rules in tissue-like P systems. Hence, in some sense, $\mathcal{TPP}$ systems can be more suitable for some applications.

The remainder of this paper is organized as follows. Section 2 reviews some basic knowledge involved in this study. Subsequently, in Section 3, a variant of tissue-like P systems ($\mathcal{TPP}$ systems) is proposed, and its computational universality is explored by simulating register machines in Section 4. In Section 5, we explore its computational efficiency by solving the SAT problem. In Section 6, we briefly analyze the computational complexity of these systems. Finally, we present our conclusions and discuss future research directions.

2 Foundations

2.1 Formal language theory

An alphabet Γ is a finite non-empty set of symbols. A symbol sequence from a set Γ is called a string, and the length of any string is finite. It must be emphasized that if there are multiple copies of a symbol in a rule (rules are applied to evolve and communicate objects located in a P system), and the length of the symbol should include the number of all the copies. A string that does not contain a symbol is denoted by λ. Obviously, the length of such a string is 0. Γ^+ represents all non-empty strings from Γ; that is, $\Gamma^+ = \Gamma^* - \{\lambda\}$, where, Γ^*, represents the set of all strings over Γ. Hence, Γ^+ can be viewed as a set of symbols composed by a string (do not include empty string). For arbitrary two strings α and β, the symbols in the strings are combined to form their concatenation, which can be denoted by $\alpha\beta$. If there is only one symbol in an alphabet Γ, for instance, $\Gamma = \{a\}$, then $\{a\}^*$ (resp., $\{a\}^+$) is abbreviated as a^* (resp., a^+).

Given an alphabet Γ, a multiset from the alphabet can be denoted as (Γ, f), where, $f : \Gamma \to \mathbb{N}$ is a mapping from Γ to natural numbers. If $\Gamma = \{a_1, \ldots, a_n\}, 1 \leq i \leq n$, $f(a_i)$ is the multiplicity of object a_i, that is, the number of a_i is $f(a_i)$. We represent all objects in the alphabet $\{a_1, \ldots, a_n\}$ with the multiset $a_1^{f(a_1)} a_2^{f(a_2)} \ldots a_n^{f(a_n)}$. Two multisets can

be connected as the union operation; for example, the union of multisets $m_1 = (\Gamma, f_1)$ and $m_2 = (\Gamma, f_2)$ is denoted by $m_1 + m_2 = (\Gamma, g)$, where $g(x) = f_1(x) + f_2(x)$ $(x \in \Gamma)$. The relative complement m_2 in m_1 can be denoted by $m_1 \backslash m_2 = (\Gamma, g)$, $g(x) = f_1(x) - f_2(x)$ can be obtained, where $f_1(x) \geq f_2(x)$ (otherwise, $g(x) = 0$).

2.2 Register machines

To prove Turing universal, in this research, we use register machines to characterize the computational power of P systems.

Definition 1. $M = (m, H, l_0, l_h, I)$ denotes a register machine with m registers, where, H is a set of instruction labels, l_0, $l_h \in H$ are initial and halting instruction labels respectively, I is the following three type of instructions:

- $l_i : (\mathcal{ADD}(r), l_j, l_k)$ (This instruction adds 1 to register r, and then the instruction labelled l_j or l_k is performed non-deterministically [37]);
- $l_i : (\mathcal{SUB}(r), l_j, l_k)$ (There are two cases for this instruction. If register r is non-zero, the register is subtracted by 1, and then the instruction labelled l_j is executed; otherwise, the instruction labelled l_k can be performed);
- $l_h : \mathcal{HALT}$ (This instruction is applied to stop the entire computing process).

In this work, we will prove that $\mathcal{TPP}$ systems can generate all computable sets of numbers by simulating register machines. Specifically, initially, a system M executes the instruction with label l_0 firstly. At this moment, all registers must be empty (for example, store the number 0). Subsequently, the system will continue to apply instructions until the last instruction l_h is applied, and the system reaches completion status. The system executes the instructions one by one, a number will appear in the register r, and we can say that the register machine M has generated the number. In this way, if a register machine M can generate all set of Turing computable numbers, NRE can be characterized by M, where NRE denotes the family of recursively enumerable sets computed by Turing machines.

3 Tissue-like P systems with polarizations

In this section, we introduce the notion of polarization into tissue-like P systems; thus, a new type of variant is constructed, where rules are applied with the strategy of flat maximal parallelism. We assume that the reader is already familiar with some basic knowledge of membrane computing, such as formal language and automata theory.

3.1 Model description

There are a series of cells (all of them at the same level) in $\mathcal{TPP}$ system, and outside of cells is called the environment. Each cell including the environment can define a region.

Definition 2. A $\mathcal{TPP}$ system (degree $m \geq 1$, $1 \leq i \leq m$) is defined as follows:

$$\Pi = (O, E, w_i, p_i, \mathcal{R}, i_{out}),$$

where

- O is an alphabet of objects;
- E represents a set of objects that are located in the environment with an arbitrary multiplicity $(E \subseteq O)$;
- $w_i \subseteq O (1 \leq i \leq m)$, denote the multiset of objects initially located in the cell with label i;
- $p_i \in \{+, -, 0\}$, indicate the polarization initially located on the cell with label i;
- i_{out} is the output region $(i_{out} \in \{0, 1, \ldots, m\})$;

- $\mathcal{R}$ represents the set of rules in such system, communication rules and division rules respectively, where, communication rules include symport rules and antiport rules. In what follows, we denote by $[u]_i^{e_1}$ a cell with label i that has polarization e_1 ($e_1 \in \{+, -, 0\}$). Additionally, the length of a symport or antiport rule is the total number of objects in the rule.

  (i) Symport rules:
  between two cells: $[u]_i^{e_1}[\ ]_j^{e_2} \rightarrow [\ ]_i^{e'_1}[u]_j^{e'_2}$,
  between cells and the environment: $u[\ ]_i^{e_1} \rightarrow [u]_i^{e'_1}$, or $[u]_i^{e_1} \rightarrow u[\ ]_i^{e'_1}$,
  where, $i, j \in \{1, ..., m\}, e_1, e_2, e'_1, e'_2 \in \{+, -, 0\}, i \neq j, u \in O^+$. The length of a symport rule is $2|u|$.

  At a given moment, if a multiset u exists inside cell i and this cell (resp., cell j) has polarization e_1 (resp., e_2), a rule $[u]_i^{e_1}[\ ]_j^{e_2} \rightarrow [\ ]_i^{e'_1}[u]_j^{e'_2}$ is activated. By applying the rule, the multiset in cell i appears in cell j, and the polarization on cell i (resp., cell j) changes to e'_1 (resp., cell e'_2). Similarly, a rule $u[\ ]_i^{e_1} \rightarrow [u]_i^{e'_1}$, or $[u]_i^{e_1} \rightarrow u[\ ]_i^{e'_1}$ can be applied between a cell and the environment. In this case, due to the polarization on the cell, this multiset enters the cell from the environment or transfers from this cell to the environment in the opposite direction.

  (ii) Antiport rules:
  between two cells: $[u]_i^{e_1}[w]_j^{e_2} \rightarrow [w]_i^{e'_1}[u]_j^{e'_2}$,
  between cells and the environment: $u[w]_i^{e_1} \rightarrow w[u]_i^{e'_1}$,
  where, $i, j \in \{1, ..., m\}, e_1, e_2, e'_1, e'_2 \in \{+, -, 0\}, i \neq j, u, w \in O^+$. The length of an antiport rule is $2(|u| + |w|)$.

  At a given moment, if a multiset u exists inside cell i and another multiset w exists inside cell j, and cell i (resp., cell j) has polarization e_1 (resp., e_2), a rule $[u]_i^{e_1}[w]_j^{e_2} \rightarrow [w]_i^{e'_1}[u]_j^{e'_2}$ is activated. By applying the rule, the two multisets are transferred to opposite regions, and the polarization on cell i (resp., cell j) changes to e'_1 (resp., cell e'_2). Similarly, a rule $u[w]_i^{e_1} \rightarrow w[u]_i^{e'_1}$ can be applied between a cell and the environment. In this case, due to the polarization on the cell, multisets in the cell and the environment are transferred in the opposite direction, and the polarization of this cell can be changed.

  When we apply communication rules, importantly, the polarizations on cells may not be changed; that is, $e_1 = e'_1$, $e_2 = e'_2$.

  (iii) division rules:

  $[a]_i^{e_1} \rightarrow [b]_i^{e_2}[c]_i^{e_3},\ e_1, e_2, e_3 \in \{+, -, 0\}$,

  where, $i \in \{1, 2, \ldots, m\}, a, b, c \in O, i \neq i_{out}$.

  When a cell has polarization e_1 and a corresponding object appears in the cell, two cells labeled i are generated; simultaneously, two objects are located in the two regions to replace the previous object a. Importantly, objects a, b, and c can represent the same or different objects. In addition, the polarizations on the generated cells may not be the same as that on the initial cell.

Definition 3. $NOP_m(sym_n, anti_k, polar_l)$ is the set of natural numbers generated by $\mathcal{TPP}$ systems, where m is the number of cells and l is the quantity of the polarizations on all the cells; in addition, *sym* (resp., *anti*) indicates symport rule (resp., antiport rule), and n (resp., k) denotes the maximal length of symport rule (resp., antiport rule).

3.2 System execution

In [2], the notion of maximal parallelism was proposed by Păun: at a given moment, all rules which can be applied have to be applied to all possible objects. And for flat maximal parallelism, in each step, between two regions, a maximal set of applicable rules is chosen and each rule in the set is applied exactly once [40]. In our work, we adopt the latter strategy to apply rules; that is, we introduce flat maximal parallelism into tissue-like P systems. At a given moment, each rule of $\mathcal{TPP}$ systems can be executed once instead of multiple times.

In the following, we consider the execution of $\mathcal{TPP}$ systems. At one step, with respect to two different regions, if there are multiple rules that can be executed, only rules of the same type can be executed, that is, only symport rules or antiport rules can be executed. Moreover, the cells associated with rules must have the same polarizations before and after execution; otherwise, even if more than one rule can be employed at a step, only the rules whose associated cells have the same polarizations before and after execution can be non-deterministically applied. In addition, symport rules that can be executed are activated only when objects communicate in one direction, or else only the rules in which objects are transferred in the same direction can be non-deterministically selected to be applied.

The *configuration* of a $\mathcal{TPP}$ system is influenced by the following factors: the cells, the multiset of objects in each region, and the polarizations on the cells. At any step, all applicable rules associated with different regions can be executed in parallel on different channels; moreover, rules that can be executed must be applied to the corresponding channels. Initially, the notion of an *initial configuration* is employed to characterize such a system. At each step, a configuration can be generated with the execution of rules, and a *transition* is employed to characterize the variation between two configurations. A sequence of transitions is characterized by a *computation*. Notably, rules are employed continuously based on flat maximal parallelism and follow the principle of *non-determinism* [2]. Finally, when the system halts, the execution of all the rules terminates, and no rules can be used, so we can obtain the *halting configuration*, and the computation result can be obtained in the corresponding region.

3.3 Two examples

Example 1. Let $\Pi_1 = (O, E, w_1, w_2, p_1, p_2, \mathcal{R}, i_{out})$ be a $\mathcal{TPP}$ system, where $O = \{a, b, c\}$, $E = w_2 = \varnothing$, $w_1 = w_2 = b$, $p_1 = p_2 = 0$, there are infinite copies of objects a and c in the environment. $\mathcal{R}$ denotes the rules in the system, and there are two rules associated with each cell respectively.

$R_1 : a[b]_1^0 \to b[a]_1^+.$
$R_2 : c[\ ]_1^0 \to [c]_1^+.$
$R_3 : a[\ ]_2^0 \to [a]_2^+.$
$R_4 : [b]_2^0 \to b[\ ]_2^+.$

Because rules can be executed in parallel on different channels, the rules associated with cell 1 and cell 2 can be applied in parallel. First, we consider rules R_1 and R_2 associated with cell 1. Obviously, the two rules are allowed to be executed; however, only one of the two rules can be used because only rules of the same type can be executed.

Next, we consider rules R_3 and R_4 associated with cell 2. Obviously, the two rules are allowed to be executed; however, only one of the two rules can be used because symport rules are used only when objects communicate in one direction. Hence, rule R_3 or R_4 can be non-deterministically applied.

Example 2. Let $\Pi_2 = (O, E, w_1, w_2, p_1, p_2, \mathcal{R}, i_{out})$ be a $\mathcal{TPP}$ system, where $O = \{a, b\}$, $E = w_2 = \varnothing$, $w_1 = ab$, $p_1 = p_2 = 0$, $i_{out} = 2$, $\mathcal{R}$ is the set of the following rules:

$R_1 : [a]_1^+[\ ]_2^0 \to [\ ]_1^0[a]_2^0$

$R_2 : [b]_1^+[\ ]_2^0 \to [\ ]_1^0[b]_2^-$

In the simple example, the system Π_2 halts after one step. The polarizations of cell 2 are not the same after these two rules are executed, so only one rule can be non-deterministically selected to be applied. If R_1 is used, object a will appear in the output region; if R_2 is used, object b will appear in the output region. Hence, obviously, the computation result is $\{a\}$ or $\{b\}$.

Let $\Pi_3 = (O, E, w_1, w_2, \mathcal{R}, i_{out})$ be a tissue-like P system (see [30]), where $O = \{a, b\}$, $E = w_2 = \varnothing$, $w_1 = ab$, $i_{out} = 2$, $\mathcal{R}$ is the following rules:

$R_1 : (1, a/\lambda, 2)$

$R_2 : (1, b/\lambda, 2)$

The system halts after one step. In this example, rules R_1 and R_2 are performed simultaneously. It is easy to obtain the computation result $\{a, b\}$.

As shown by these simple examples, when polarizations are introduced into a tissue-like P system, the parallelism of the associated membrane system may decline.

3.4 Recognizer $\mathcal{TPP}$ systems

In membrane computing, to solve decision problems, a general approach is to use recognizer P systems [41]. Next, recognizer $\mathcal{TPP}$ systems are employed.

Definition 4. A recognizer $\mathcal{TPP}$ system (degree $m \geq 1$, $1 \leq i \leq m$) is a construct:

$$\Pi = (O, \Sigma, E, w_i, p_i, \mathcal{R}, i_{in}, i_{out}),$$

where

- Σ represents an input alphabet ($\Sigma \subseteq O$);
- `yes`, `no` $\in O$;
- i_{in} (resp., i_{out}) represents the input (resp., output) region;

In such a system, the other parameters are defined as in Definition 2. A recognizer $\mathcal{TPP}$ system starts from an initial configuration with an input multiset. Eventually, the system must stop computing, generating object `yes` or `no` as the computation result.

Definition 5. We use $X = (I_X, \theta_X)$ to represent a decision problem, where I_X indicates the instances of this problem and θ_X represents a predicate based on the instances. When solving decision problems with $\mathcal{TPP}$ systems (denoted by $\mathbf{\Pi} = \Pi_n \mid n \in \mathbb{N}$) in polynomial time if the following holds:

(i) $\mathbf{\Pi}$ is polynomially uniform by Turing machines; namely, the system Π_n can be built by a deterministic Turing machine which works in polynomial time;

(ii) relative to I_X, there is a pair (cod, s) of polynomial-time computable functions such that:

- $u \in I_X$, such a system has an input multiset $cod(u)$;
- relative to $n \in \mathbb{N}$, $s^{-1}(n)$ can be a finite set;
- such a system is sound for the given (X, cod, s). Relative to $u \in I_X$, $\mathcal{TPP}$ systems have an accepting computation with input $cod(u)$, in this case, $\theta_X(u) = 1$;
- such a system is polynomially bounded for the given (X, cod, s). Given a decision problem, $\mathcal{TPP}$ systems have to halt after $p(|u|)$ steps (p represents a polynomial function);

- such a system is complete for the given (X, cod, s). The value of register r is the number of objects a_r in cell 1. The register machine stops computing when object l_h is generated in cell 1, and all rules stop running. At that moment, the number of object a_1 in the cell can be regarded as the computation result.

  If $\theta_X(u) = 1$, then all calculations with $cod(u)$ are accepting computations.

Definition 6. The maximum rule length of a $\mathcal{TPP}$ system is equivalent to that of symport rules and antiport rules in the system. We use $PMC_{\mathcal{TPP}(k)}$ to represent that the class of decision problems can be solved by a family of recognizer $\mathcal{TPP}$ systems in a uniform manner in polynomial time, where the maximal rule length is k.

4 University of $\mathcal{TPP}$ systems

In this section, to explore the computational power of $\mathcal{TPP}$ systems, we prove that this variant is equivalent to a Turing machine in terms of number generation.

$M = (m, H, l_0, l_h, I)$ is a register machine with m registers. The generated number will appear in register 1. When $\mathcal{TPP}$ systems halt, all registers (except register 1) can be empty. The reader is assumed to have mastered basic automata theory (see [**?**] for details).

Theorem 1. $NOP_1(sym_2, anti_4, polar_3) = NRE$.

Proof. A $\mathcal{TPP}$ system is constructed as follows, where the polarization on cell 1 is "0".

$$\Pi = (O, E, w_1, p_1, R, i_{out}),$$

where

- $O = \{l, l^{(1)}, l^{(2)}, l^{(3)}, l^{(4)}, l^{(5)} \mid l \in H\} \cup \{a_r \mid 1 \leq r \leq m\}$;
- $E = \{l^{(1)}, l^{(3)}, l^{(5)} \mid l \in H\} \cup \{a_r \mid 1 \leq r \leq m\}$;
- $w_1 = \{l_0\} \cup \{l^{(2)}, l^{(4)} \mid l \in H\}$;
- $p_1 = 0$;
- i_{out}=1.

(1) For the ADD instruction $l_i : (\mathcal{ADD}(r), l_j, l_k)$, rules are designed as follows:

$$R_1 : l_i^{(1)}[l_i]_1^0 \to l_i[l_i^{(1)}]_1^0.$$
$$R_2 : a_r[l_i^{(1)}]_1^0 \to l_i^{(1)}[a_r]_1^-.$$
$$R_3 : l_j[l_i^{(2)}]_1^- \to l_i^{(2)}[l_j]_1^-.$$
$$R_4 : l_k[l_i^{(2)}]_1^- \to l_i^{(2)}[l_k]_1^-.$$
$$R_5 : l_i^{(2)}[\ ]_1^- \to [l_i^{(2)}]_1^0.$$

A $\mathcal{TPP}$ system executes an ADD instruction as follows. Initially, the system runs with rule R_1, namely, object $l_i^{(1)}$ and object l_i in cell 1 exchange their regions, yet the polarization on the cell remains unchanged. Next step, due to object $l_i^{(1)}$, rule R_2 is executed, changing the polarization to "+", and object a_r enters cell 1. At step 3, rule R_3 or R_4 is executed randomly: l_j or l_k enters the cell and $l_i^{(2)}$ re-enters the cell; simultaneously, the polarization is changed to "0" (the same polarization as at the initial moment), and the system is ready for the next simulation. Therefore, three steps are required to complete the simulation of an ADD instruction.

(2) $l_i : (\mathcal{SUB}(r), l_j, l_k)$ is the SUB instruction, where the following are rules in the system:

$$R_6 : l_i^{(1)}[l_i]_1^0 \to l_i[l_i^{(1)}]_1^+.$$
$$R_7 : l_i^{(3)}[l_i^{(4)}]_1^+ \to l_i^{(4)}[l_i^{(3)}]_1^0.$$
$$R_8 : l_i^{(4)}[a_r]_1^0 \to a_r[l_i^{(4)}]_1^-.$$
$$R_9 : l_i^{(5)}[l_i^{(3)}]_1^0 \to l_i^{(3)}[l_i^{(5)}]_1^-.$$
$$R_{10} : l_i^{(3)}[l_i^{(4)}]_1^- \to l_i^{(4)}[l_i^{(3)}]_1^+.$$
$$R_{11} : l_j[l_i^{(3)}]_1^+ \to l_i^{(3)}[l_j]_1^0.$$
$$R_{12} : l_i^{(4)}[l_i^{(5)}]_1^+ \to l_i^{(5)}[l_i^{(4)}]_1^0.$$

$R_{13} : l_i^{(4)}[l_i^{(1)}]_1^- \to l_i^{(1)}[l_i^{(4)}]_1^0.$
$R_{14} : l_k[l_i^{(5)}]_1^0 \to l_i^{(5)}[l_k]_1^0.$

An SUB instruction is simulated with the rules designed above. Initially, the system runs by applying rule R_6, namely, object $l_i^{(1)}$ and object l_i exchange their regions, changing the polarization to "+". Next, due to the polarization on cell 1, rule R_7 is executed, changing the polarization to "0", and object $l_i^{(3)}$ including $l_i^{(4)}$ exchange its regions. At that moment, according to whether object a_r appears in the cell or not, two cases are possible:

1) If object a_r exists, rules R_8 and R_9 are executed at once. The emphasis is that the two rules have the same type, and the polarization before and after execution is the same; therefore, they are applied simultaneously. Due to rule R_8, object a_r comes into the environment so that the number of object a_r is decreased. In addition, due to rule R_9, $l_i^{(5)}$ and $l_i^{(3)}$ exchange their regions. At step 4, rule R_{10} will be executed, changing the polarization to "-", and $l_i^{(3)}$ including $l_i^{(4)}$ exchange their regions. Finally, R_{11} and R_{12} are executed simultaneously, where R_{11} is applied to generate object l_j in the cell, thereby simulating the SUB instruction, and R_{12} is applied to send $l_i^{(4)}$ to cell 1; therefore the next SUB instruction is performed. Overall, the rules are performed successively as follows.

$$R_6 \to R_7 \to \{R_8, R_9\} \to R_{10} \to \{R_{11}, R_{12}\}$$

2) If a_r does not exist, the system only executes rule R_9 at the next step, changing the polarization to "-", and $l_i^{(5)}$ including $l_i^{(3)}$ exchange their regions. At step 4, rule R_{13} starts to execute, changing the polarization to "-", and $l_i^{(4)}$ including $l_i^{(1)}$ exchange their regions. At the final step, R_{14} is applied, and object l_k appears in the cell; therefore, the next SUB instruction is performed. Overall, the rules are performed successively as follows.

$$R_6 \to R_7 \to R_9 \to R_{13} \to R_{14}$$

In general, 5 steps are required to complete the computation. Obviously, the above computing process can correctly simulate the SUB instruction. When the system eventually reaches completion, object l_h would be generated in the cell 1.

Theorem 2. $NOP_1(sym_4, polar_3) = NRE$.

Proof. A $\mathcal{TPP}$ system is constructed as follows, where only symport rules can be used, the polarization on cell 1 is "0".

$$\Pi = (O, E, w_1, p_1, R, i_{out}),$$

where

- $O = \{l, l^{(1)}, l^{(2)}, l^{(3)}, l^{(4)}, l^{(5)}, l^{(6)} \mid l \in H\} \cup \{a_r \mid 1 \le r \le m\}$;
- $E = \{l^{(3)} \mid l \in H\} \cup \{a_r \mid 1 \le r \le m\}$;
- $w_1 = \{l_0\} \cup \{l^{(1)}, l^{(2)}, l^{(4)}, l^{(5)}, l^{(6)} \mid l \in H\}$;
- $p_1 = 0$;
- i_{out}=1.

(1) $l_i : (\mathcal{ADD}(r), l_j, l_k)$ is the ADD instruction, where the following are rules in the system:

$R_1 : [l_i l_i^{(1)}]_1^0 \to l_i l_i^{(1)}[\ ]_1^0.$
$R_2 : a_r l_i^{(1)}[\ ]_1^0 \to [a_r l_i^{(1)}]_1^+.$
$R_3 : [l_i^{(2)}]_1^+ \to l_i^{(2)}[\ ]_1^+.$
$R_4 : l_j l_i^{(2)}[\ ]_1^+ \to [l_j l_i^{(2)}]_1^0.$
$R_5 : l_k l_i^{(2)}[\ ]_1^+ \to [l_k l_i^{(2)}]_1^0.$

A $\mathcal{TPP}$ system execute an ADD instruction as follows. At step 1, the system runs by applying rule

R_1, and objects l_i and $l_i^{(1)}$ are moved into the environment. Next, rule R_2 will be performed, changing the polarization from "0" to "+", and object a_r enters cell 1, which satisfies the requirement of adding 1 to the corresponding register. At step 3, due to the polarization "+", object $l_i^{(2)}$ will change its region. In the end, rule R_4 or R_5 is executed randomly: l_j or l_k enters the cell, changing the polarization from "+" to "0", and the system is ready for the next simulation. Therefore, four steps are required to complete the simulation process.

(2) $l_i : (\mathcal{SUB}(r), l_j, l_k)$ is the SUB instruction, where the following are rules in the system:

$$
\begin{aligned}
&R_6 : [l_i l_i^{(1)}]_1^0 \rightarrow l_i l_i^{(1)}[\ ]_1^0.\\
&R_7 : l_i^{(1)} l_i^{(3)}[\ ]_1^0 \rightarrow [l_i^{(1)} l_i^{(3)}]_1^+.\\
&R_8 : [l_i^{(3)} a_r]_1^+ \rightarrow l_i^{(3)} a_r[\ ]_1^0.\\
&R_9 : [l_i^{(4)} l_i^{(5)}]_1^+ \rightarrow l_i^{(4)} l_i^{(5)}[\ ]_1^0.\\
&R_{10} : l_i^{(3)} l_i^{(4)}[\ ]_1^0 \rightarrow [l_i^{(3)} l_i^{(4)}]_1^+.\\
&R_{11} : l_i^{(5)} l_j[\ ]_1^+ \rightarrow [l_i^{(5)} l_j]_1^0.\\
&R_{12} : [l_i^{(3)} l_i^{(6)}]_1^0 \rightarrow l_i^{(3)} l_i^{(6)}[\ ]_1^-.\\
&R_{13} : l_i^{(4)} l_i^{(5)}[\ ]_1^- \rightarrow [l_i^{(4)} l_i^{(5)}]_1^0.\\
&R_{14} : l_i^{(6)} l_k[\ ]_1^- \rightarrow [l_i^{(6)} l_k]_1^0.
\end{aligned}
$$

An SUB instruction is simulated with the rules designed above. Initially, the system runs with rule R_6, and objects l_i and $l_i^{(1)}$ are moved into the environment. Next, rule R_7 is executed, changing the polarization to "+", and objects $l_i^{(1)}$ including $l_i^{(3)}$ are moved to cell 1. At that moment, according to whether object a_r exists in cell 1 or not, we obtain the following two cases.

1) If object a_r exists, rules R_8 and R_9 are executed simultaneously, changing the polarization to "0"; due to rule R_8, object a_r comes into the environment, and the number of object a_r decreases; in addition, according to the rule R_9, objects $l_i^{(4)}$ and $l_i^{(5)}$ are moved into the environment. At step 4, rule R_{10} will be executed, changing the polarization to "+", and objects $l_i^{(3)}$ and $l_i^{(4)}$ enter the cell. Finally, R_{11} is applied to generate object l_j in the cell, thereby simulating the SUB instruction; moreover, object $l_i^{(5)}$ is sent to cell 1, changing the polarization to "0"; therefore, the next SUB instruction would be performed. Overall, the rules are performed successively as follows.

$$R_6 \rightarrow R_7 \rightarrow \{R_8, R_9\} \rightarrow R_{10} \rightarrow R_{11}$$

2) If a_r does not exist, the system only executes rule R_9 at the next step, changing the polarization to "0", and objects $l_i^{(4)}$ and $l_i^{(5)}$ are transferred to the environment. At step 4, rule R_{12} starts to execute, changing the polarization to "-", and objects $l_i^{(3)}$ and $l_i^{(6)}$ are moved into the environment. In the end, R_{13} and R_{14} are applied simultaneously, and object l_k appears in the cell. In addition, objects $l_i^{(4)}$, $l_i^{(5)}$ and $l_i^{(6)}$ enter the cell, changing the polarization to "0"; therefore, the next SUB instruction is performed. Overall, the rules are performed successively as follows.

$$R_6 \rightarrow R_7 \rightarrow R_9 \rightarrow R_{12} \rightarrow \{R_{13}, R_{14}\}$$

In general, 5 steps are required to complete the computation. Obviously, the above computing process can correctly simulate the SUB instruction. If object l_h has been generated and the execution of all rules has been completed, the system stops computing, and the number of object a_1 in this cell is regarded as the computation result. When the system eventually reaches completion, object l_h would be generated in the output region.

5 A uniform solution to the SAT problem based on $\mathcal{TPP}$ systems

5.1 Constructing $\mathcal{TPP}$ systems to solve the SAT

Theorem 3. $\mathtt{SAT} \in PMC_{\mathcal{TPP}(4)}$

Proof. A SAT, which has n Boolean variables and m clauses formula, can be denoted as follows:

$$C_j = y_{1,j} \vee \cdots \vee y_{p_j,j},$$

where $y_{i,j} \in \{x_l,\ \neg x_l \mid 1 \le l \le n\}$, $1 \le i \le p_j$, $1 \le j \le m$; $\neg x_l$ is the negation of a propositional variable x_l.

Relative to a SAT formula γ, it can be encoded by $cod(\gamma)$ as follows:

$$cod(\gamma) = \alpha_{1,1} \cdots \alpha_{n,1}\alpha_{1,2} \cdots \alpha_{n,2} \cdots \alpha_{1,m} \cdots \alpha_{n,m}.$$

Next we codify $\alpha_{i,j}$ in γ with the following multiset ($1 \le j \le m$, $1 \le i \le n$)

$$\alpha_{i,j} = \begin{cases} Y_{i,j}, \text{which denotes } x_i \text{ is in } C_j; \\ N_{i,j}, \text{which denotes } \neg x_i \text{ is in } C_j; \\ B_{i,j}, \text{which denotes that neither } x_i \text{ nor } \neg x_i \text{ is in } C_j. \end{cases}$$

To solve the SAT with n Boolean variables and m clauses formula, we build a recognizer P system $\Pi_{\mathcal{TPP}(m,n)}$:

$$\Pi_{\mathcal{TPP}(m,n)} = (O, \Sigma, E, w_1, w_2, p_1, p_2, \mathcal{R}, i_{in}, i_{out}),$$

where

- $O = \Sigma \cup \{a_i \mid 1 \le i \le n+1\} \cup \{t_{i,j}, f_{i,j}, T_{i,j}, F_{i,j} \mid 1 \le i \le n, 1 \le j \le m+1\} \cup \{r_j \mid 1 \le j \le m\} \cup \{c_j \mid 1 \le j \le m+1\} \cup \{d, \mathtt{yes}, \mathtt{no}\}$;
- $\Sigma = \{Y_{i,j}, N_{i,j}, B_{i,j} \mid 1 \le i \le n, 1 \le j \le m\}$;
- $E = \{t_{i,j}, f_{i,j}, T_{i,j}, F_{i,j} \mid 1 \le i \le n, 1 \le j \le m+1\}$ $\cup \{a_i \mid 1 \le i \le n+1\} \cup \{r_j \mid 1 \le j \le m\} \cup \{c_j \mid 1 \le j \le m+1\}$;
- $w_1 = \{a_1\}, w_2 = \{d, \mathtt{yes}, \mathtt{no}\}$;
- $p_1 = p_2 = 0$;
- $i_{in} = 1$, $i_{out} = 0$;
- R is the following set of rules:

(1) Generation phase

$$R_{1,i} : [a_i]_1^0 \to [t_{i,1}]_1^0[f_{i,1}]_1^0, 1 \le i \le n.$$

$$R_{2,i,j} : [t_{i,j}Y_{i,j}]_1^0[\ ]_2^0 \to [\ ]_1^+[t_{i,j}Y_{i,j}]_2^0, \quad 1 \le i \le n, 1 \le j \le m.$$
$$R_{3,i,j} : [t_{i,j}N_{i,j}]_1^0 \to t_{i,j}N_{i,j}[\ ]_1^+, \quad 1 \le i \le n, 1 \le j \le m.$$
$$R_{4,i,j} : [t_{i,j}B_{i,j}]_1^0 \to t_{i,j}B_{i,j}[\ ]_1^+, \quad 1 \le i \le n, 1 \le j \le m-1.$$

$$R_{5,i} : [t_{i,m}B_{i,m}]_1^0[\ ]_2^0 \to [\ ]_1^+[t_{i,m}B_{i,m}]_2^+, 1 \le i \le n.$$
$$R_{6,i} : t_{i,m+1}[t_{i,m}]_2^+ \to t_{i,m}[t_{i,m+1}]_2^+, 1 \le i \le n$$
$$R_{7,i} : [\ ]_1^+[t_{i,m+1}B_{i,m}]_2^+ \to [t_{i,m+1}B_{i,m}]_1^0[\ ]_2^-, 1 \le i \le n.$$

$$R_{8,i,j} : [f_{i,j}N_{i,j}]_1^0[\ ]_2^0 \to [\ ]_1^-[f_{i,j}N_{i,j}]_2^0, \quad 1 \le i \le n, 1 \le j \le m.$$
$$R_{9,i,j} : [f_{i,j}Y_{i,j}]_1^0 \to f_{i,j}Y_{i,j}[\ ]_1^-, \quad 1 \le i \le n, 1 \le j \le m.$$
$$R_{10,i,j} : [f_{i,j}B_{i,j}]_1^0 \to f_{i,j}B_{i,j}[\ ]_1^-, \quad 1 \le i \le n, 1 \le j \le m-1.$$

$$R_{11,i} : [f_{i,m}B_{i,m}]_1^0[\ ]_2^0 \to [\ ]_1^-[f_{i,m}B_{i,m}]_2^+, 1 \le i \le n.$$
$$R_{12,i} : f_{i,m+1}[f_{i,m}]_2^+ \to f_{i,m}[f_{i,m+1}]_2^+, 1 \le i \le n.$$
$$R_{13,i} : [\ ]_1^-[f_{i,m+1}B_{i,m}]_2^+ \to [f_{i,m+1}B_{i,m}]_1^0[\ ]_2^-, \quad 1 \le i \le n.$$

$$R_{14,i,j} : T_{i,j+1}N_{i,j}[\ ]_1^+ \to [T_{i,j+1}N_{i,j}]_1^0, \quad 1 \le i \le n, 1 \le j \le m.$$
$$R_{15,i,j} : T_{i,j+1}B_{i,j}[\ ]_1^+ \to [T_{i,j+1}B_{i,j}]_1^0, \quad 1 \le i \le n, 1 \le j \le m.$$

$$R_{16,i,j} : F_{i,j+1}Y_{i,j}[\ ]_1^- \to [F_{i,j+1}Y_{i,j}]_1^0, \quad 1 \le i \le n, 1 \le j \le m.$$

$$R_{17,i,j} : F_{i,j+1}B_{i,j}[\ ]_1^- \to [F_{i,j+1}B_{i,j}]_1^0, \quad 1 \le i \le n, 1 \le j \le m.$$

$$R_{18,i,j} : t_{i,j}[T_{i,j}]_1^0 \to T_{i,j}[t_{i,j}]_1^0, \quad 1 \le i \le n, 1 \le j \le m.$$

$$R_{19,i,j} : f_{i,j}[F_{i,j}]_1^0 \to F_{i,j}[f_{i,j}]_1^0, \quad 1 \le i \le n, 1 \le j \le m.$$

$$R_{20,i,j} : t_{i,j+1}[t_{i,j}]_2^0 \to t_{i,j}[t_{i,j+1}]_2^+, \quad 1 \le i \le n, 1 \le j \le m.$$

$$R_{21,i,j} : r_j[Y_{i,j}]_2^0 \to Y_{i,j}[r_j]_2^+, \quad 1 \le i \le n, 1 \le j \le m.$$

$$R_{22,i,j} : [\ ]_1^+[t_{i,j+1}r_j]_2^+ \to [t_{i,j+1}r_j]_1^0[\ ]_2^0, \quad 1 \le i \le n, 1 \le j \le m-1.$$

$$R_{23,i} : [\ ]_1^+[t_{i,m+1}r_m]_2^+ \to [t_{i,m+1}r_m]_1^0[\ ]_2^-, 1 \le i \le n.$$

$$R_{24,i,j} : f_{i,j+1}[f_{i,j}]_2^0 \to f_{i,j}[f_{i,j+1}]_2^+, \quad 1 \le i \le n, 1 \le j \le m.$$

$$R_{25,i,j} : r_j[N_{i,j}]_2^0 \to N_{i,j}[r_j]_2^+, \quad 1 \le i \le n, 1 \le j \le m.$$

$$R_{26,i,j} : [\ ]_1^-[f_{i,j+1}r_j]_2^+ \to [f_{i,j+1}r_j]_1^0[\ ]_2^0, \quad 1 \le i \le n, 1 \le j \le m-1.$$

$$R_{27,i} : [\ ]_1^+[f_{i,m+1}r_m]_2^+ \to [f_{i,m+1}r_m]_1^0[\ ]_2^-, 1 \le i \le n.$$

$$R_{28} : [d]_2^- \to [d]_2^0[d]_2^0.$$

$$R_{29,i} : a_{i+1}[t_{i,m+1}]_1^0 \to t_{i,m+1}[a_{i+1}]_1^0, 1 \le i \le n.$$

$$R_{30,i} : a_{i+1}[f_{i,m+1}]_1^0 \to f_{i,m+1}[a_{i+1}]_1^0, 1 \le i \le n.$$

The initial configuration of $\mathcal{TPP}$ systems is described as follows: There are two cells labeled 1 and 2 in the system, where cell 1 contains object a_1 and cell 2 contains objects d, `yes` and `no`. In addition, the two cells have the same polarization "0" on them. When solving the `SAT` problem, the computing process can be divided into three phases: 1) the generation phase; 2) the checking phase; and 3) the output phase.

In general, both cell 1 and cell 2 have division rules, where cell 1 can generate all assignments of n variables, and cell 2 can communicate corresponding objects with cell 1. The generation phase primarily corresponds to two parallel computing processes: one is between cell 1 and the environment, and the other is primarily between cell 1 and cell 2 (the environment is also involved as an auxiliary function). The computing process of this phase is shown in Figure 1.

This stage primarily corresponds to a multiple iterative computing process of n variables, where the process of each variable can be performed similarly. Related to the first iteration of the variables, rule $R_{1,i}$ can be applied at step 1; as a result, the number of cells labeled 1 is doubled, and objects $t_{1,1}$ (corresponding to the assignment of "true") and $f_{1,1}$ (corresponding to the assignment of "false") appear in the cells. At the second step, variable x_1 in m clauses is checked to determine whether the current assignment is satisfied with the execution of rules from $R_{2,i,j}$ to $R_{5,i}$ and of rules from $R_{8,i,j}$ to $R_{11,i}$, where rules from $R_{2,i,j}$ to $R_{5,i}$ are designed for the assignment of "true" and rules from $R_{8,i,j}$ to $R_{11,i}$ are designed for the assignment of "false". When corresponding rules are executed, for the "true" (resp., "false") assignment, the polarization of cell 1 will be changed to "+" (resp., "-"). There are two possible parallel computing processes according to whether the current assignment is satisfiable for clauses:

(i) The corresponding clause is satisfiable with the current assignment. In this case, rule $R_{2,i,j}$ or $R_{8,i,j}$ can be employed, but they would not be executed simultaneously. At the next step, $R_{20,i,j}$ and $R_{21,i,j}$ (or $R_{24,i,j}$ and $R_{25,i,j}$) are applied. These rules have the same rule type and have the same polarizations before and after execution, so they can be

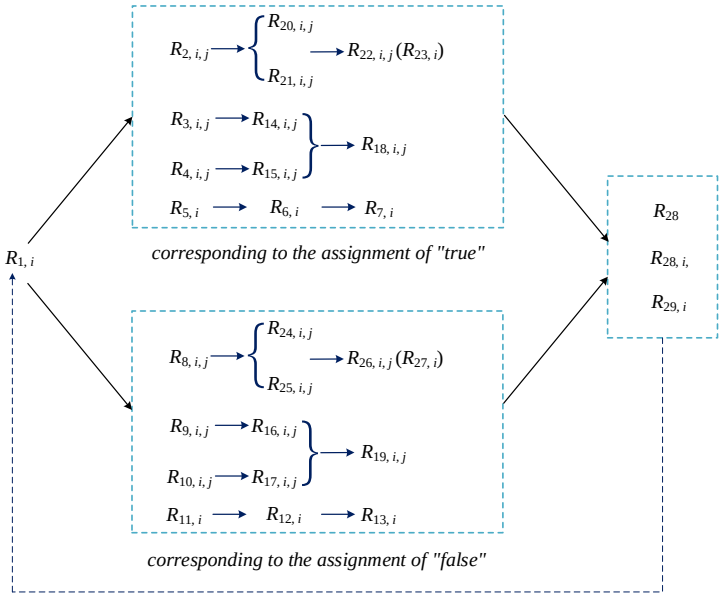

Fig. 1 Computing process during the generation phase.

used simultaneously. By applying these rules at this step, the polarization of cell 1 can be changed from "0" to "+". Subsequently, rule $R_{22,i,j}$ or $R_{26,i,j}$ is executed, sending multiset $t_{i,j+1}r_j$ back to cell 1 and changing the polarizations of the corresponding cell 1 and cell 2 to "0". In this way, the whole process increases the second subscript of $t_{i,j}$ and the satisfiable clauses are obtained (represented by r_j). If the current variable corresponds to the last clause in the formula, rule $R_{23,i,j}$ or $R_{27,i}$ is employed, changing the polarization of cell 1 to "0" for the next iterative variable and changing the polarization of cell 2 to "-" for the application of division rule R_{28} at the next step.

(ii) The corresponding clause is not satisfiable with the current assignment. In this case, rule $R_{3,i,j}$ or $R_{4,i,j}$ (resp., $R_{9,i,j}$ or $R_{10,i,j}$) can be employed. Subsequently, rules $R_{14,i,j}$ to $R_{17,i,j}$ are used. Specifically, if rule $R_{3,i,j}$ (resp., $R_{4,i,j}$) was executed at the last step, rule $R_{14,i,j}$ (resp., $R_{15,i,j}$) is applied at this step; similarly, if rule $R_{9,i,j}$ (resp., $R_{10,i,j}$) was executed at the last step, rule $R_{16,i,j}$ (resp., $R_{17,i,j}$) will be applied at this step. The polarization of cell 1 would be changed to "0". Next, rule $R_{18,i,j}$ (resp., $R_{19,i,j}$) is executed. In this way, the whole process can increase the second subscript of $t_{i,j}$ (resp., $f_{i,j}$). We consider a particular case, namely, if the last clause in the formula corresponding to the current variable is $B_{i,m}$, we design rules $R_{5,i}$ to $R_{7,i}$ (including rules $R_{11,i}$ to $R_{13,i}$) to perform division on cell 2. Hence, these rules can not only increase the second subscript of $t_{i,j}$ (resp., $f_{i,j}$) one by one but also change the polarization of cell 2 to "-". Obviously, a division rule can be employed on each cell 2.

With the execution of the above rules, the following computation continues in parallel on cell 1 and cell 2. For a cell with label 1, when the execution of m clauses is completed, object $t_{i,m+1}$ (resp., $f_{i,m+1}$) can be generated in each cell 1, and rule $R_{29,i}$ (resp., $R_{30,i}$) is used at this step, generating object a_{i+1} in the corresponding cell. Hence, rule $R_{1,i}$ is executed for the generated object a_{i+1}. Similarly, the assignment of the next variable is generated, and the computation of the next variable starts. In contrast, for a cell with label 2, due to the polarization "-" on cell 2 at this moment, division rule

R_{28} can be executed to double the number of cells labeled 2, thereby preparing for the next variable. Ultimately, the generation phase generates 2^n cells labeled 1 and 2^n cells labeled 2.

In particular, the system runs under flat maximal parallelism. For example, when rules $R_{22,i,j}$ or $R_{26,i,j}$ are activated, the same cells with label 1 do not receive multiple multiset $t_{i,j+1}r_j$ because a rule cannot be executed multiple times on the same cell at the same time. Hence, each cell with a label of 2 can only send objects to exactly one cell with a label of 1.

The entire generation phase is completed within $3mn + 2n$ steps.

(2) Checking phase

$$R_{31} : c_1[a_{n+1}]_1^0 \to a_{n+1}[c_1]_1^0.$$
$$R_{32,j} : [c_j r_j]_1^0[\ ]_2^0 \to [\ ]_1^+[c_j r_j]_2^+, 1 \le j \le m.$$
$$R_{33,j} : c_{j+1}[c_j]_2^+ \to c_j[c_{j+1}]_2^0, 1 \le j \le m.$$
$$R_{34,j} : [\ ]_1^+[c_j]_2^0 \to [c_j]_1^0[\ ]_2^0, 2 \le j \le m+1.$$

The main task of this stage is to detect whether a sequence of objects $r_1, r_2, \ldots, r_m$ exist in each cell 1. When the generation phase ends, object a_{n+1} is produced in each cell 1, and then rule R_{31} can be executed. As a result, object c_1 in the environment and object a_{n+1} in the cell exchange their own regions. Subsequently, by employing rule $R_{32,j}$, multiset $c_1 r_1$ in a cell labeled 1 arbitrarily enters a cell labeled 2, and then the system executes rule $R_{33,j}$, increasing the subscript of object c_1 in cell 2 to generate object c_2. Next, rule $R_{34,j}$ is executed, and object c_2 is sent to a cell labeled 1. Similarly, the above rules from $R_{32,j}$ to $R_{34,j}$ are performed repeatedly. The entire checking phase is completed within $3m + 1$ steps.

(3) Output phase

$$R_{35} : [\mathtt{no}]_2^0 \to \mathtt{no}[\ ]_2^0.$$
$$R_{36} : [c_m]_1^0[\mathtt{yes}]_2^0 \to [\mathtt{yes}]_1^+[c_m]_2^+$$
$$R_{37} : \mathtt{no}[\mathtt{yes}]_1^+ \to \mathtt{yes}[\mathtt{no}]_1^0.$$

When the system starts running, at step 1, rule R_{35} is used together with rule $R_{1,1}$. When the checking phase halts, if object c_m is generated on an arbitrary cell 1, obviously, the SAT has at least one satisfiable solution, and then rule R_{36} is activated; as a result, object c_m and object yes exchange their regions where they are located. Eventually, the system stops computing when the execution of rule R_{37} halts, and then object yes is located in the environment, indicating that the computing result of the SAT is satisfiable. In this case, the entire output phase is completed within 2 steps.

In contrast, when the checking phase halts, if we cannot find object c_m in any cell 1, the SAT has no satisfiable solution. In this case, neither rule R_{36} nor R_{37} will be performed. Therefore, the system reaches the halting configuration, and object no is located in the output region, indicating that the computing result of the SAT is not satisfiable. In this case, the entire output phase does not take one step.

5.2 Some formal details

Relative to Theorem 3, the computing resources of $\Pi_{\mathcal{TPP}(m,n)}$ are listed below.

- size of the set O: $7mn+2m+5n+5 \in O(mn)$;
- initial number of cells: $2 \in O(1)$;
- initial number of objects: $4 \in O(1)$;
- the types of polarizations: $3 \in O(1)$;

- the total number of rules: $18mn+3m+7n+5 \in O(mn)$;
- the maximal length of rules: $4 \in O(1)$.

Eventually, the system stops computing. At this point, no rules can be activated, and no rules are being applied. If the SAT is satisfiable, the system halts after $3mn + 3m + 2n + 3$ steps; in contrast, if the SAT is not satisfiable, then the system halts after $3mn + 3m + 2n + 1$ steps. Therefore, the $\mathcal{TPP}$ system is polynomially bounded.

6 Computational complexity of $\mathcal{TPP}$ systems

Next, we analyse analyze the computational complexity of $\mathcal{TPP}$ systems. In the following, we mainly consider it from the perspective of the maximum rule length, along with the types of rules.

For a membrane system, the shorter the maximum rule length is, the more optimized the constructed system. Therefore, if a membrane system can be constructed to solve the same problem with a shorter maximum rule length, the system provides more excellent computational properties. For instance, in [33], the SAT was solved with a maximum rule length of 8; however, in [42], the maximum rule length was reduced to 3 with the same model. Hence, the latter paper improved the previous research results. Obviously, in membrane computing, the rule length is an important factor affecting the computational efficiency.

In [43], in a maximally parallel manner, it was proven that only finite sets of non-negative integers can be generated by tissue-like P systems with symport rules of maximal length 1 and antiport rules of maximal length 2 or with only symport rules of maximal length 2. Recently, when rule synchronization was introduced [44], some scholars obtained the result of Turing universality by applying the same model with the same rule lengths. In this work, however, when polarizations are employed, we also prove that the computational power of $\mathcal{TPP}$ systems is equivalent to that of a Turing machine, where the maximum lengths of symport rules and antiport rules are 2 and 4, respectively, or the maximum length of only symport rules is 4 (only one cell and three types of polarizations are employed in both cases). Moreover, due to cell division rules, $\mathcal{TPP}$ systems can solve the SAT problem (a classical **NP**-complete problem) in polynomial time, where the maximum rule length is 4.

According to Theorem 1 and Theorem 2 in this paper, the conclusions can be denoted as follows:

$NOP_1(sym_1, anti_2) = NFIN$,

$NOP_1(sym_2) = NFIN$.

Hence, despite the computational power of tissue-like P systems, the following two theorems are still open problems: $NOP_1(sym_1, anti_2) = NRE$ and $NOP_1(sym_2) = NRE$. Moreover, if the maximal rule length is 2, it is difficult to solve NP-complete problems.

In this work, the maximum length of $\mathcal{TPP}$ systems (constructed in Theorem 1, Theorem 2 and Theorem 3) seems to be 4, yet they can be modified to a form with a maximum length of 2. For instance,

Symport rule: $[t_{i,j}Y_{i,j}]_1^0[\ ]_2^0 \rightarrow [\ ]_1^+[t_{i,j}Y_{i,j}]_2^0$,

Antiport rule: $t_{i,j+1}[t_{i,j}]_2^0 \rightarrow t_{i,j}[t_{i,j+1}]_2^+$,

We can denote these rules in other forms, such as

Symport rule: $(1^{\{0\rightarrow+\}}, t_{i,j}Y_{i,j}/\lambda, 2^{\{0\rightarrow0\}})$

Antiport rule: $(0, t_{i,j+1}/t_{i,j}, 2^{\{0\rightarrow+\}})$

However, to better describe the polarizations on cells, we used the former form to characterize the corresponding rules. Hence, although the maximum length of rules is 4 in this work, this is the reason that rules are denoted in different ways; that

is, the maximum rule length in this work can be regarded as 2. Actually, the total number of objects participating in these rules is 2 (if there are multiple copies of an object, the total number would be counted).

7 Conclusions

In this work, inspired by the phenomenon of cell polarization in biological tissues, we have proposed a new mechanism that employs polarizations to control the execution of rules and introduces the strategy of flat maximal parallelism, thereby constructing a variant called $\mathcal{TPP}$ systems. To explore the computational power of $\mathcal{TPP}$ systems, we have proven that the variant is equivalent to a Turing machine in terms of generating numbers by using one cell and applying antiport rules with at most two objects and symport rules with only one object or only symport rules with two objects. Moreover, we have introduced cell division rules with polarizations and solved the classic SAT problem efficiently, where the total number of objects participating in the rules is only 2. In brief, on the one hand, polarizations are considered, adding some power and control to the system; on the other hand, parallelism between symport and antiport rules is restricted to only one type, and only in one direction if using symport rules.

In this work, a cell in $\mathcal{TPP}$ systems may have three types of polarization. If we decrease the number of polarization types, e.g., to two types, it would be interesting to study whether such a variant can obtain the same results of this paper. In addition, we apply antiport rules and symport rules when proving the computational efficiency of $\mathcal{TPP}$ systems; where, although only two objects are used in the two types of rules, symport rules are not noncooperative. In terms of computational complexity, whether rules in the proof can be noncooperative or not merits further study.

Polarizations are introduced into tissue-like P systems, and the parallelism of the rules may be reduced. Nevertheless, this approach can precisely control the rules through polarizations. We can consider time-free P systems, where the rule execution time is uncertain. If $\mathcal{TPP}$ systems and the time-free approach are combined, rules can be controlled more precisely through polarizations on cells. Hence, such a variant is worthy of further study.

$\mathcal{TPP}$ systems operate under the mode of flat maximal parallelism. To date, inspired by biological phenomena, various other modes have been presented, e.g., asynchronism [45], minimal parallelism [46], local synchronization [47], and rule synchronization [44]. Maximal parallelism is the most widely used mode. It is challenging to solve the SAT problem using $\mathcal{TPP}$ systems with maximal parallelism.

Recently, Song et al. proposed a novel type of tissue-like P systems, namely, monodirectional tissue P systems with promoters [36], in which objects can only move in one specified direction. These systems realize Turing universality under the influence of promoters with a maximum rule length of 2. It would be interesting to construct a new monodirectional P system based on $\mathcal{TPP}$ systems.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (Grant No. 61806114), and by the Science and Technology Research Program of Chongqing Municipal Education Commission (Grant No. KJZD-K202003201).

References

1. Adleman L. Molecular computation of solutions to combinatorial problems. Science, 1994, 266(5187):1021–1024
2. Păun G. Computing with membranes. Journal of Computer and System Sciences, 2000, 61(1):108–143
3. Garis H. Introduction to evolutionary computing. Evolutionary Computation, 2003, 12(2):269–271
4. Cheng B, Titterington B. Neural networks: a review from a statistical perspective. Statistical Science, 1994, 9(1):2–30
5. Freund R, Păun G, Pérez-Jiménez M J. Tissue P systems with channel states. Theoretical Computer Science, 2005, 330(1):101–116
6. Gazdag Z, Kolonits G. A new method to simulate restricted variants of polarizationless P systems with active membranes. Journal of Membrane Computing, 2019, 1(4):251–261
7. Luo Y, Zhao Y, Chen C. Homeostasis tissue-like P systems. IEEE Transactions on NanoBioscience, 2021, 20(1):126–136
8. Pan L, Song B. P systems with rule production and removal. Fundamenta Informaticae, 2020, 171(1-4):313–329
9. Peng H, Wang J. Coupled neural P systems. IEEE Transactions on Neural Networks and Learning Systems, 2019, 30(6):1672–1682
10. Peng H, Bao T, Luo X, Wang J, Pérez-Jiménez M J. Dendrite P systems. Neural Networks, 2020, 127:110–120
11. Song X, Valencia-Cabrera L, Peng H, Wang J. Spiking neural P systems with autapses. Information Sciences, 2021, 570:383–402
12. Wu T, Păun A, Zhang Z, Pan L. Spiking neural P systems with polarizations. IEEE Transactions on Neural Networks and Learning Systems, 2018, 29(8):3349–3360
13. Wu T, Qiang L, Pan L. Evolution-Communication spiking neural P systems. International Journal of Neural Systems, 2020, 31(2):2050064
14. Wu T, Zhang L, Q. Lyu, Jin Y. Asynchronous spiking neural P systems with local synchronization of rules. Information Sciences, 2022, 588:1–12
15. Cooper J, Nicolescu R. Alternative representations of P systems solutions to the graph colouring problem. Journal of Membrane Computing, 2019, 1(2):112–126
16. Díaz-Pernil D, Christinal H A, Gutiérrez-Naranjo M A. Solving the 3-COL problem by using tissue P systems without environment and proteins on cells. Information Sciences, 2018, 430–431:240–246
17. Guo P, Zhu J, Chen H, Yang R. A linear-time solution to All-SAT problem based on P systems. Chinese Journal of Electronics, 2018, 27(2):367–373
18. Luo Y, Xiong Z, Zhang G. Time-free solution to SAT problem by tissue P systems. Mathematical Problems in Engineering, 2017, 2017:1–8
19. Luo Y, Tan H, Y. Zhang, Y Jiang. The computational power of timed P systems with active membranes using promoters. Mathematical Structures in Computer Science, 2019, 29(5):663–680
20. Singh G, K. Deep. Effectiveness of new multiple-PSO based membrane optimization algorithms on CEC 2014 benchmarks and Iris classification. Natural Computing, 2017, 16(3):473–496
21. 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, 2014, 24(05):1440006
22. Buiu C, George A G. Membrane computing models and robot controller design, current results and challenges. Journal of Membrane Computing, 2019, 1(4):262–269
23. Hu J, Peng H, Wang J, Yu W. kNN-P: A kNN classifier optimized by P systems. Theoretical Computer Science, 2020, 817:55–65
24. Frisco P, Gheorghe M, Pérez-Jiménez M J. Applications of membrane computing in systems and synthetic biology. 1nd ed. Germany: Berlin Heidelberg, 2014
25. García-Quismondo M, Levin M, Lobo D. Modeling regenerative processes with membrane computing. Information Sciences, 2017, 381:229–249
26. Wang J, Peng H, Yu W, Ming J, Pérez-Jiménez M J, Tao C, Huang X. Interval-valued fuzzy spiking neural P systems for fault diagnosis of power transmission networks. Engineering Applications of Artificial Intelligence, 2019, 82:102–109
27. Zhang G, Pérez-Jiménez M J, Gheorghe M. Real-life applications with membrane computing. 1nd ed. Germany: Berlin Heidelberg, 2017
28. Song B, Li K, Orellana-Martín D, Pérez-Jiménez M J, Pérez-Hurtado I. A survey of nature-inspired computing: membrane computing. ACM Computing Surveys, 2021, 54(1):1–31
29. Zhang G, Pérez-Jiménez M J, Riscos-Núñez A, Verlan S, Konur S, Hinze T, Gheorghe M. Membrane computing models: implementations. 1nd ed. Germany: Berlin Heidelberg, 2021
30. Martín-Vide C, Păun G, Pazos J, Rodríguez-Patón A. Tissue P systems. Theoretical Computer Science, 2003, 296(2):295–326
31. Ionescu M, Păun G, T. Yokomori. Spiking neural P sys-

tems. Fundamenta Informaticae, 2006, 71(2):279–308
32. Păun G, Pérez-Jiménez M J, Riscos-Núñez A. Tissue P systems with cell division. International Journal of Computers Communications and Control, 2008, 3(3):295–303
33. Pan L, Pérez-Jiménez M J. Computational complexity of tissue-like P systems. Journal of Complexity, 2010, 26(3):296–315
34. Aman B, Ciobanu G. Travelling salesman problem in tissue P systems with costs. Journal of Membrane Computing, 2021,3(2):97–104
35. Song B, Zhang C, Pan L. Tissue-like P systems with evolutional symport/antiport rules. Information Sciences, 2017, 378:177–193
36. Song B, Zeng X, Jiang M, Pérez-Jiménez M J. Monodirectional tissue P systems with promoters. IEEE Transactions on Cybernetics, 2020, 51(1):438–450
37. Păun G. P Systems with active membranes: attacking NP complete problems. Journal of Automata, Languages and Combinatorics, 2001, 6(1):75–90
38. Besozzi D, Busi N, Cazzaniga P, Ferretti C, Leporati A, Mauri G, PEscini D, Zandron C. (Tissue) P systems with cell polarity. Mathematical Structures in Computer Science, 2009, 19(6):1141–1160
39. Alhazov A, Freund R, Ivanov S, Verlan S. Tissue P systems with vesicles of multisets. International Journal of Foundations of Computer Science, 2022, 33:179–202
40. Pan L, Păun G, Song B. Flat maximal parallelism in P systems with promoters. Theoretical Computer Science, 2016, 623:83–91
41. Pérez-Jiménez M J, Romero-Jiménez A, Sancho-Caparrini F. A polynomial complexity class in P systems using membrane division. Journal of Automata, Languages and Combinatorics, 2006, 11(4):423–434
42. Pérez-Jiménez M J, Sosík P. An optimal frontier of the efficiency of tissue P systems with cell separation. Fundamenta Informaticae, 2015, 138(1–2):45–60
43. Păun G, Rozenberg G, Salomaa A. The Oxford handbook of membrane computing. 1nd ed. New York: Oxford University Press, 2010
44. Song B, Pan L. Rule synchronization for tissue P systems. Information and Computation, 2021, 281:104685
45. Frisco P, Govan G, Leporati A. Asynchronous P systems with active membranes. Theoretical Computer Science, 2012, 429:74–86
46. Ciobanu G, Pan L, Păun G, Pérez-Jiménez M J. P systems with minimal parallelism. Theoretical Computer Science, 2007, 378(1):117–130
47. Pan L, Alhazov A, Su H, Song B. Local synchronization on asynchronous tissue P systems with symport/antiport rules. IEEE Transactions on NanoBioscience, 2020, 19(2):315–320