A Survey of Nature-Inspired Computing: Membrane Computing

BOSHENG SONG and KENLI LI, Hunan University, China DAVID ORELLANA-MARTÍN, MARIO J. PÉREZ-JIMÉNEZ, and IGNACIO PÉREZ-HURTADO, Universidad de Sevilla, Spain

Nature-inspired computing is a type of human-designed computing motivated by nature, which is based on the employ of paradigms, mechanisms, and principles underlying natural systems. In this article, a versatile and vigorous bio-inspired branch of natural computing, named *membrane computing* is discussed. This computing paradigm is aroused by the internal membrane function and the structure of biological cells. We first introduce some basic concepts and formalisms of membrane computing, and then some basic types or variants of *P systems* (also named *membrane systems*) are presented. The state-of-the-art computability theory and a pioneering computational complexity theory are presented with P system frameworks and numerous solutions to hard computational problems (especially **NP**-complete problems) via P systems with membrane division are reported. Finally, a number of applications and open problems of P systems are briefly described.

Key Words : Nature-inspired computing, membrane computing, distributed systems, computational complexity

1 INTRODUCTION

Natural sciences are influencing the area of information sciences, and the meaning of computation is modified. Up to now, there are many computing paradigms inspired by a large number of

This work was supported by National Natural Science Foundation of China (61972138), the Fundamental Research Funds for the Central Universities (531118010355), Hunan Provincial Natural Science Foundation of China (2020JJ4215), and the Key Research and Development Program of Changsha (kq2004016). The authors from the Universidad de Sevilla also acknowledge the support from research project TIN2017-89842-P, cofinanced by Ministerio de Economia, Industria y Competitividad (MINECO) of Spain, through the Agencia Estatal de Investigacion (AEI), and by Fondo Europeo de Desarrollo Regional (FEDER) of the European Union.

Authors' addresses: B. Song and K. Li (corresponding author), College of Information Science and Engineering, Hunan University, Changsha, China; emails: {boshengsong, lkl}@hnu.edu.cn; D. Orellana-Martín, M. J. Pérez-Jiménez, and I. Pérez-Hurtado, Department of Computer Science and Artificial Intelligence, Universidad de Sevilla, Sevilla, Spain; emails: {dorellana, marper, perezh}@us.es.

natural phenomena, such as the functioning of the brain, group behavior, cell membranes, and the immune system. Natural computing is a highly interdisciplinary area, which primarily encompasses computer science and natural sciences along with an indirect association with all other possible scientific fields.

A key component of natural computing is to explore suitable physical substrates for implementing computations. Two prime instances of nature-inspired paradigms are quantum computing and molecular computing. Quantum computing exploits the amazing laws of quantum mechanics to process information, and computations are achieved by using physical methods such as superconductors, nuclear magnetic resonance techniques, and ion-traps [64]; while in molecular computing, a large amount of information can be encoded with biomolecules (for instance, DNA strands), and various bio-operations, such as splicing, self-assembly, are employed to implement computations [1].

The most established "classical" nature-inspired computing models are *cellular automata*, aroused by self-reproducing/self-replicating biological organisms; *neural computation*, the goal of this paradigm is to understand and compute, motivated by the function and nature of the brain [127]; and *evolutionary computation*, which proposes a strategy for finding optimal or near-optimal solutions inspired by natural evolution and adaptation [34]. Some recent computing paradigms motivated from natural phenomena include *artificial life*, which is a cutting-edge research discipline that links the field of computational intelligence with the study of living systems [59]; *swarm intelligence*, motivated by the behavioral models of social animals [54], for instance, bees, fish, ants, or birds; *artificial immune systems*, which are motivated by the processes and principles of the biological immune systems [18]; *membrane computing*, abstracted from the partition of cell structure, and the cooperation of cells in organs and tissues to investigate the computing power, computing efficiency, applications, and implementations [78].

In this article, we focus on an active branch of nature-inspired computing paradigm: membrane computing, which is motivated from the function and structure of biological cells, abstracting from biological processes of chemicals interact and cross membranes, in other words, by the action of membranes in domain of cells into "reactors". All computational models studied in membrane computing framework are named *P systems*. In view of the difference of the underlying structure, P systems are classified into two types: *neural-like P systems* [49] or *tissue-like P systems* [61] (an arbitrary graph structure), and *cell-like P systems* [78] (a tree structure). A representative monograph of membrane computing is consulted in [80] and [84], and the state-of-the-art additional information of this research field is available on http://ppage.psystems.eu.

2 ELEMENTS OF MEMBRANE COMPUTING

2.1 Elements of Formal Language Theory

In this subsection, some elements from languages theory are presented in the classic form, dealing with strings, and also in the multiset form [101].

An *alphabet* Γ is defined by a non-empty set of symbols, and then the set of all strings generated by concatenating arbitrary number of symbols is indicated as Γ^*. $\Gamma^* \setminus \{\lambda\}$, denoted by Γ^+, is a set of string excluding empty string λ (there is no symbol in a string). We indicate as $|u|$ the *length* of u, which is defined by all symbols in u.

A *multiset* $\mathcal{M}$ over Γ is designed by two tuples (Γ, f), where a function f is defined from an alphabet Γ to the natural numbers set. In addition, $\mathcal{M}^+(\Gamma)$ (respectively, $\mathcal{M}(\Gamma)$) indicates the set of all non-empty multisets (respectively, the set of all multisets). If $\Gamma = \{a_1, \ldots, a_k\}$, then multiset $\mathcal{M}$ is represented by $\{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}$. For instance, let $\mathcal{M}_1 = (\Gamma, f_1)$, $\mathcal{M}_2 = (\Gamma, f_2)$ be two multisets, so the union of these two multisets is represented by $\mathcal{M}_1 + \mathcal{M}_2$, which is indicated as

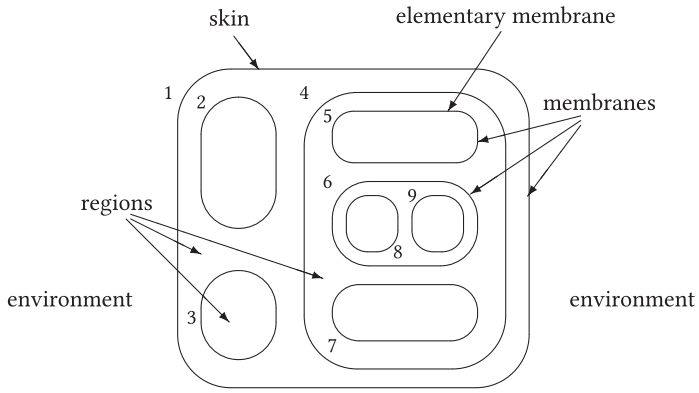

Fig. 1. A membrane structure.

$(\Gamma, f_1(x) + f_2(x))$ such that every $x \in \Gamma$; besides, the relative complement of $\mathcal{M}_2 = (\Gamma, f_2)$ in $\mathcal{M}_1 = (\Gamma, f_1)$ is indicated as $\mathcal{M}_1 \backslash \mathcal{M}_2 = (\Gamma, f_1(x) - f_2(x))$ if $\mathcal{M}_2$ is included in $\mathcal{M}_1$; otherwise, $\mathcal{M}_1 \backslash \mathcal{M}_2 = 0$.

2.2 Membrane Structure

Motivated from the function and structure of a biological cell, we consider hierarchical arrangements of a finite number of membranes; that is, membranes nested within other membranes. Membranes delimit *compartments* (also named *regions*) where various chemicals (also called *objects*) evolve on the basis of local reaction rules; moreover, objects are transferred across membranes by applying specific rules. A membrane with no lower neighbors is called *elementary*; otherwise, a membrane is named *non-elementary*. A membrane with no upper neighbor is called a *skin membrane*. The space "outside" the skin membrane is called the *environment*. Depending on the model, this environment will play an active or a passive role during the evolutionary process of the system. A simple membrane structure is shown in Figure 1. We remark that a global clock which marks the time for all regions exists for every membrane system, and each rule (of whatever type) is executed in exactly one time-unit.

A membrane structure can also be expressed by parentheses, thus the membrane structure of Figure 1 can be described as a parentheses expression

$$[\ [\ ]_2 [\ ]_3 [\ [\ ]_5 [\ [\ ]_8 [\ ]_9\ ]_6 [\ ]_7\ ]_4\ ]_1.$$

2.3 Multiset Rewriting Rules

Multiset rewriting rules are an abstraction of biochemical reactions taking place in a cell, which have form $u \to v$ (here u, v are two multisets), located in membranes. A rule in a region is available if multiset w of objects presented in that membrane includes multiset u, i.e., $u \subseteq w$. For instance, the use of the rule $abc^2 \to bd^3$ transforms one object a, one b, and two c into three d, while b is reproduced (here it plays the role of a catalyst). This rule can be applied to the multiset $a^2bc^3d^2$, but not to $abcd^2$, because $abc^2 \subseteq abcd^2$ does not hold.

The *weight* of a multiset rewriting rule $u \to v$ (denoted by $|u|$) is defined by all the occurrences of objects in u; a rule with weight at least two is named *cooperative*, while a rule with only one object is called *noncooperative*.

The communication between regions is provided by passing/exchanging objects between regions through membranes. Communication is a basic mechanism of cooperation among regions, and it is incorporated in rewriting rules by *target indications*, which have form $p \to q$, where each

element of q has the form (a, tar), tar is either *out*, *here,* or *in*. The meaning of these target indications is described as follows:

If $tar = here$, then object a stays in membrane where such rule resides.

If $tar = out$, then object a is sent to its parent region.

If $tar = in$, then object a is randomly sent to one of the child regions. Note that elementary regions do not have child regions, so a rule including $tar = in$ residing in an elementary region cannot be applied.

For instance, $ab \rightarrow (c, here)(d, out)(e, in)$ works as follows: a reacts with b, being removed from the region. Besides, an object c (that remains in the region where the rule resides), an object d (sent to the parent region), and an object e (sent in one of the child regions) are produced.

2.4 Communication Rules

Except for rewriting rules, another significant style of rules is called *communication rules*, which regulate the shift of objects (placed in neighboring regions) among regions. A typical instance of communication rules is antiport and symport rules, which are motivated by the coupled cross-membrane transport of ions and molecules.

There are two forms for symport rules: (u, out) and (u, in); while only one form exists for antiport rules: $(u, out; v, in)$. If these communication rules are associated with region h, then the meaning of antiport rules and symport rules is described as follows:

- for a symport rule (u, in), the multiset u placed in its parent of region h is moved to region h;
- for a symport rule (u, out), the multiset u placed in region h is moved to the parent of region h;
- for an antiport rule $(u, out; v, in)$, the multiset u placed in region h is moved to the parent of region h, and meanwhile the multiset v placed in the parent of region h is moved to region h.

Obviously, for communication rules (symport/antiport rules) to be applicable it is required that the multisets u and v of objects are available in corresponding regions.

2.5 Other Variants of Rules

Furthermore, there exist rules that are capable of changing the structure of the P system (creating, dividing, separating, merging, etc.). Besides, inspired by a mathematical or a biological motivation, inhibitors or promoters are associated with rules, and the employ of such rules is regulated by priority relations, objects among regions are moved under the control of the permeability (if a membrane is non-permeable, then all objects cannot pass through such membrane).

2.6 Strategies of Using Rules

There are various possible strategies of applying the available sets of rules to multisets of objects appeared in regions at a given time unit.

- *Maximal Parallelism.* At every computation step, a multiset of rules chosen at the current configuration is maximal.
- *Flat Maximal Parallelism.* A maximal set of valid rules is selected and every rule in this set is employed exactly once in each region in a computation step.
- *Minimal Parallelism.* If there exist some application rules in a membrane, then at least one rule must be employed.
- *Asynchronous.* Arbitrary number of available rules is employed in a computation step.

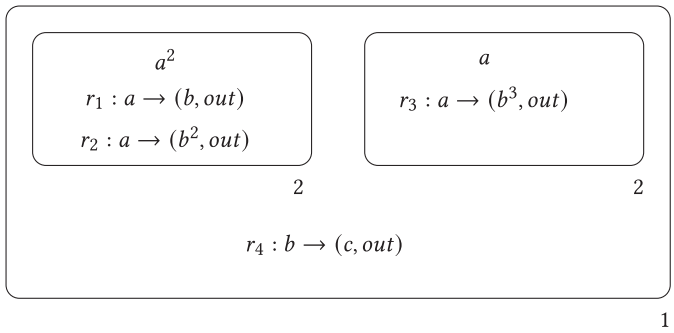

Fig. 2. An example system Π working in a flat maximally parallel way.

- *Sequential.* Exactly one rule is employed in a computation step.
- *Bounded Parallelism.* At most a certain number of rules is employed in each region in a computation step.

Next, we present two examples to illustrate the mode of application multiset rewriting rules in a maximally parallel and in a flat maximally parallel.

Example 1. Assume that currently the multiset $w = a^3b^2c^2$ of objects is present in region h, and such region contains the set of rules $R = \{r_1, \ldots, r_5\}$, with

$$r_1 : ab \to v_1,\ r_2 : c \to v_2,\ r_3 : bc \to v_3,\ r_4 : a^3c^2 \to v_4,\ r_5 : ad \to v_5,$$

for some $v_1, \ldots, v_5$.

Note that applying r_1 removes ("consumes") the multiset ab, while two parallel applications of r_1 remove the multiset a^2b^2. The remaining "still available" multiset ac^2 does not allow one more parallel application of r_1, but it allows for two parallel applications of r_2, which removes the multiset c^2. The remaining multiset a does not allow an application of any rule at all, and therefore we say that the maximally parallel use of the *multiset of rules* is $\{(r_1, 2), (r_2, 2)\}$. Besides, we can find that the application of $\{(r_3, 2)\}$ is maximally parallel (for the remaining multiset a^3 of objects, no rule can be used), and the application of $\{(r_1, 1), (r_2, 1), (r_3, 1)\}$ is maximally parallel (for the remaining multiset a^2 of objects, no rule can be used). Note that the application of $\{(r_1, 2), (r_2, 1)\}$ is not maximally parallel, because for the remaining multiset ac of objects, r_2 can be still used.

Example 2. Now we present how a P system Π works in a flat maximally parallel way, the system contains two membranes 1 and 2 located in skin membrane 1, the initial multisets of objects and the rewriting rules of the system are located in membranes, and the environment is the output region (see Figure 2).

The P system works as follows: At step 1, the maximal set of available rules in the left membrane 2 is $\{r_1, r_2\}$, hence rules r_1, r_2 are applied once, sending three copies of b to membrane 1; simultaneously, the maximal set of available rules in the right membrane 2 is $\{r_3\}$, hence rule r_3 is applied once, sending three copies of b to membrane 1. Therefore, after the first step, both membranes 2 do not have objects, and six copies of b appear in membrane 1. At each of the next six steps, rules in both membranes 2 are not available, and the maximal set of applicable rules in membrane 1 is $\{r_4\}$; by applying rule r_4 once at every step, six copies of b in membrane 1 are revised to c, which are sent to environment. Hence, after seven steps, six copies of c appear in environment, and the system halts. The set of numbers produced by system Π is $\{6\}$.

2.7 Halting Conditions

The standard way to define successful computation is by means of halting. However, some other variants of halting conditions have been studied in membrane computing.

- *Unconditional Halting*. The result of a computation can be considered from any configuration starting from the initial one.
- *Halting with States*. The system reaches a configuration in which a final state appears.
- *Partial Halting*. R is a set of rules, which is partitioned into disjoint subsets $R_1, \ldots, R_h$. A computation stops when there are no rules available to the current configuration, which has a rule from every set R_i, $1 \leq i \leq h$.

3 VARIANTS OF P SYSTEMS

In this section, some basic types of P systems are presented in membrane computing, which gives an insight to the most important models in the theory.

3.1 Transition P Systems

The notion of transition P systems (the generic variant of P systems) was raised in [78]. A simplified variant of this construct, called *symbol-object P system,* is presented here.

Definition 3.1. A transition P system (with degree $q \geq 1$) is a framework of the form

$$\Pi = (\Gamma, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, (\mathcal{R}_1, \rho_1), \ldots, (\mathcal{R}_q, \rho_q), i_{out}),$$

where:

- Γ is an alphabet, and every element in such alphabet is called an object;
- μ is a hierarchical membrane structure;
- $\mathcal{M}_i, 1 \leq i \leq q$, are multisets of objects, which are placed in membranes $1, 2, \ldots, q$;
- $\mathcal{R}_i, 1 \leq i \leq q$, are sets of rewriting rules combined with corresponding membranes $1, 2, \ldots, q$, ρ_i is a priority relation among rules of R_i ($1 \leq i \leq q$). A rewriting rule has form $u \to v$, where a multiset u is from Γ, and a multiset $v = v'$ or $v = v'\delta$ (δ is a special symbol not in Γ), and a multiset v' is over

$$(\Gamma \times \{out, here\}) \cup (\Gamma \times \{in_j \mid 1 \leq j \leq q\}).$$

- $i_{out} \in \{1, 2, \ldots, q\}$ refers to output membrane.

If a rewriting rule $u \to v \in R_i$ is employed, all objects in multiset u are depleted, besides, all objects in multiset v are generated; besides, each object from v are transferred on the basis of instructions $here, out, in_j$ combined with them: for instance, if multiset v have (b, tar) and $tar = here$, then b stays in current region; in case of $tar = out$, b moves from the current region to the upper region; if $tar = in_j$, then b gets into the immediately lower membrane j (note that if membrane j is an elementary membrane, then a rule of form (b, in_j) is forbidden to be employed). Note that the instruction *here* is usually not written.

A priority relation among rules considered here is in a *strong* sense: if a rule with a higher priority is applied, then no rule of a lower priority can be applied, even if the two rules do not compete for objects. For instance, if $r_1, r_2 \in \mathcal{R}_i$, $r_1 > r_2$, and both rules r_1, r_2 are available, then at that step only rule r_1 will be applied.

There exists an special object δ such that, if a rule $u \to v : \mathcal{R}_h$ is applied, and $\delta \in v$, then in that step the membrane is dissolved, more specifically, the evolved membrane disappears and the contents release to its parent membrane. This changes the membrane structure in such a way that,

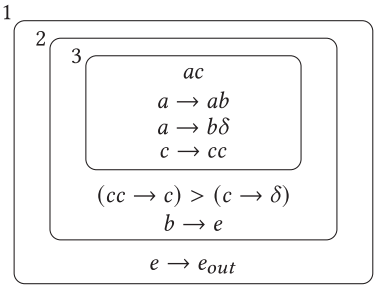

Fig. 3. An example of a transition P system.

if $child(h) = \{h_1, \ldots, h_s\}$ is a set of lower neighbors of membrane h, and the immediately upper membrane of h is i, then at the next configuration $child(i) = \{h_1, \ldots, h_s\}$.

A transition P system is worked in a maximally parallel mode. Note that when a membrane is dissolved, a computational step is divided in two microsteps: in the first one, all the objects evolve in the usual way; in the second step, the membranes that must be dissolved will be dissolved.

A *configuration* of a transition P system at some instant consists of all multisets of objects placed in every region at that moment and the current membrane structure. Each passage from a configuration to a next configuration is named a *transition*. A *computation* is defined by a sequence of transitions starting from initial configuration. A system reaches a *halting* configuration if the system does not have rule to apply. A transition P system has a result only when it reaches halting configurations.

Transition P systems as one of basic P system models have been investigated widely. In [78], an extended transition P systems, named *catalytic P systems* was proposed, where evolution rules have the forms $a \to v$ or $ca \to cv$. c, a are objects with $c \neq a$, v is a multiset, and c does not appear in v (c only helps a to evolve into v, but c itself does not undergo any transformation). c is known as the *catalyst* of the rule. The reader is referred to [35], [48], [57], and [111] for further details. However, the computation power of purely catalytic P systems as language-generating devices with one or two catalysts remains as an open problem. In [51], the relation of evolution-communication P systems with energy (a special object called "energy" is produced through evolution rules and consumed in communication rules) and non-cooperative transition P systems (all evolution rules are non-cooperative) was explored. In [52], it is known that cooperative transition P systems are simulated by evolution-communication P systems with energy.

The following instance is used to illustrate how a transition P system works:

Let Π be a transition P system, where sets of rules, initial multisets of objects and membrane structure are presented in Figure 3, the output region of Π is environment.

Owing to the non-deterministic feature of transition P systems, several different paths of computation are existed. The following is one possible path of computation for the P system described. Initially, only membrane 3 has objects a, c, at step 1, if rules $a \to ab$ and $c \to cc$ are used, then membrane 3 has objects a, b, c, c. At step 2, if rules $c \to cc$ and $a \to b\delta$ are applied, then membrane 3 has objects b, b, c, c, c, c (because of maximal parallelism), and such membrane is dissolved, every object placed in membrane 3 is released to membrane 2, and the rules in membrane 3 are disappeared. At step 3, rules $b \to e$ and $cc \to c$ (because of priority relation) are used in parallel, objects e, e, c, c are produced in membrane 2. At step 4, only rule $cc \to c$ can be applied, and membrane 2 has objects e, e, c. At step 5, only rule $c \to \delta$ can be used, membrane 2 is dissolved and two copies of object e are released to membrane 1, all rules in membrane 2 disappear. At step 6, rule $e \to e_{out}$ is applied, the environment will receive two copies of object e, the system halts.

3.2 P Systems with Active Membranes

Inspired from a mathematical and a biological point of view, it is natural to consider to increase the number of membranes, so the model of P systems with active membranes was proposed in [79], where an important feature that every membrane is associated with one of three possible charges $(+, -, 0)$ is considered.

Definition 3.2. A P system with active membranes (with degree $q \geq 1$) is a framework of the form

$$\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out}),$$

where:

- Γ is an alphabet, and every element in such an alphabet is called objects;
- H is a set of membrane labels;
- μ is a hierarchical membrane structure, and each membrane has a neutral polarization;
- $\mathcal{M}_i, 1 \leq i \leq q$, are multisets of objects over Γ initially appeared in corresponding membranes;
- $\mathcal{R}$ is a set of rules with seven types of forms (s represents the label of skin membrane):
  (a) $[\ a \to v\ ]_h^e$, for $h \in H, e \in \{+, -, 0\}, a \in \Gamma, v \in \Gamma^*$
  (evolution rules; under the control of the label of membrane and its polarization, a multiset of objects is produced, and both the label of membrane and its polarization are not changed in this process);
  (b) $a[\ \ ]_h^{e_1} \to [\ b\ ]_h^{e_2}$, for $h \in H \setminus \{s\}, e_1, e_2 \in \{+, -, 0\}, a, b \in \Gamma$, being s the skin membrane
  (*send-in* communication rules; under the control of the label of membrane and its polarization, an object enters the membrane, both the evolved object and membrane charge may be revised in this process);
  (c) $[\ a\ ]_h^{e_1} \to [\ \ ]_h^{e_2} b$, for $h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in \Gamma$
  (*send-out* communication rules; under the control of the label of membrane and its polarization, an object exits the membrane, both the evolved object and membrane charge may be revised in this process);
  (d) $[\ a\ ]_h^e \to b$, for $h \in H \setminus \{s\}, \alpha \in \{+, -, 0\}, a, b \in \Gamma$, where s is the skin membrane
  (dissolving rules; under the control of an object, the label of the membrane, and its polarization, a membrane with label h is dissolved, the evolved object may be revised; all the other objects and its inner membranes (may not exist) are released to its parent membrane; meanwhile, all rules in the membrane vanish);
  (e) $[\ a\ ]_h^{e_1} \to [\ b\ ]_h^{e_2} [\ c\ ]_h^{e_3}$, for $h \in H \setminus \{s\}, e_1, e_2, e_3 \in \{+, -, 0\}, a, b, c \in \Gamma$, being s the skin membrane
  (division of elementary membranes; under the control of an object, the label of the membrane, and its polarization, the membrane is split into two membranes, the polarizations of two generated membranes may be changed; the object arousing the rule may be changed in the generated membranes; all the rest of objects are reproduced in both of the generated membranes);
  (f) $[\ [\ \ ]_{h_1}^{e_1} [\ \ ]_{h_2}^{e_2}\ ]_h^e \to [\ [\ \ ]_{h_1}^{e_3}\ ]_h [\ [\ \ ]_{h_2}^{e_4}\ ]_h^{e'}$, $h, h_1, h_2 \in H$, $h \neq s$, $e, e', e_1, e_2, e_3, e_4 \in \{+, -, 0\}$, being s the skin membrane
  (division of non-elementary membranes; when two membranes with labels h_1, h_2 and polarizations e_1, e_2, respectively, appear in a membrane h with polarization e, then that membrane will be divided in two new membranes, the first one with membrane h_1, the second membrane with h_2 and the rest of the contents, both membranes and objects will be replicated in the generated membranes).

(g) $[\ a\ ]_h^{e_1} \to [\ \Gamma_1\ ]_h^{e_2}[\ \Gamma_2\ ]_h^{e_3}$, for $h \in H \setminus \{s\}, e_1, e_2, e_3 \in \{+, -, 0\}, a \in \Gamma, \Gamma_1 \cup \Gamma_2 = \Gamma$ and $\Gamma_1 \cap \Gamma_2 = \emptyset$, being s the skin membrane

(separation rules for elementary membranes; with the appearance of an object, the label of membrane and its polarization, a membrane is split into two membranes, the polarizations of two generated membranes may be changed; the object a is depleted, and the first generated membrane contains the objects from Γ_1, the second generated membrane contains the objects from Γ_2).

- $i_{out} \in \{0, 1, \ldots, q\}$ refers to output region (0 indicates the environment).

Rules in P systems with active membranes are employed in a maximally parallel mode with the following agreements: any membrane can be subject of only one rule of types (b) – (g); if a rule of type (a) is employed, the membrane can also evolve by a rule of other type; if a rule of types (e), (f), or (g) is used, we assume that the inner membranes and objects are evolved first, then the division rule is applied, and this two processes take only one step (that is, a bottom-up manner is considered for applying rules). Note that the rules that evolved a membrane h are applied to all membranes h. The skin membrane cannot be divided, dissolved, or separated.

A *configuration* of a P system with active membranes Π at some instant consists of all multisets of objects located in every region at that moment and the current membrane structure. The notions of transition, computation, and halting computation are analogous to the transition P systems as described above.

Since P systems with active membranes were first raised in [79], several varieties of such P systems were considered. Inspired by the fact that electrical charges placed on membranes are not very reasonable from a biological perspective, the idea for trading polarizations for labels was proposed by Alhazov et al. [8]. Motivated by the biological phenomenon of separation, Alhazov et al. introduced membrane separation into P systems with active membranes [4]. Motivated from the biological fact that when a compartment is large enough, new membranes will be generated inside it, Mutyam and Krithivasan [62] proposed P systems with membrane creation. The conception of strong (non-elementary) division was proposed by Zandron et al. [133], where two inner membranes control the division. For further varieties of P systems with active membranes, readers are referred to [7], [21], [38], and [50].

Besides the investigated maximal parallelism, some other strategies of applying rules were also explored in P systems with active membranes: time-freeness (the result achieved by the given P system is not related to the running time of rules) [102, 109]; minimal parallelism (each membrane which can evolve in a given step should do it by using at least one rule must be evolved) [25]; and asynchronous (any number of applicable rules is employed at a time). A challenging issue is to discuss the computational efficiency of polarization-less P systems with active membranes using elementary membrane division and working in an asynchronous pattern or in a flat maximally parallel pattern.

We give an example to illustrate how a P system with active membranes works.

Let $\Pi = (\{a, b, c, d, e, f\}, \{1, 2\}, [\ [\ ]_2^0\ ]_1^0, \{a\}, \emptyset, \mathcal{R}, 1)$ be a P system with active membranes, where R contains rules as shown below:

$r_1 : [\ a \to bc\ ]_1^0;$
$r_2 : b[\ ]_2^0 \to [\ d\ ]_2^+;$
$r_3 : [\ d\ ]_2^+ \to [\ e\ ]_2^+[\ f\ ]_2^-;$
$r_4 : [\ e\ ]_2^+ \to c;$
$r_5 : [\ f\ ]_2^- \to c.$

The computation for the P system is depicted as follows: Initially, the skin membrane has an object a; hence, at step 1, objects b and c are produced by using rule r_1. Next, rule r_2 is applied,

object b, outside of a membrane 2 might be rewritten into d and sent inside to the membrane, the charge of membrane 2 is transformed to positive. At the next step, by using rule r_3, two membranes with label 2 are generated and object d is evolved to object e and object f, respectively. By using rules r_4, and r_5 at step 4, two membranes with label 2 are dissolved, and objects e and f are evolved to object c. The system halts, and membrane 1 contains three copies of object c.

3.3 Communication P Systems

In transition P systems, objects are allowed both to change and move across the membranes, while in communication P systems, objects can not be evolved, and they are moved between the neighboring regions (including the environment). Symport/antiport P systems were raised in [72], which were inspired by phenomena (called symport and antiport) of biochemical ingredients via membrane channels.

Definition 3.3. A symport/antiport P system (with degree $q \geq 1$) is a framework of the form

$$\Pi = (\Gamma, \mathcal{E}, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}_1, \ldots, \mathcal{R}_q, i_{out}),$$

where:

- Γ is an alphabet, and every element in such alphabet is named objects;
- $\mathcal{E} \subseteq \Gamma$ is a set of alphabet of the environment, and it is assumed that the number of each object initial located in $\mathcal{E}$ is large enough;
- μ is a hierarchical membrane structure containing q membranes, and 1 represents the outermost membrane (skin membrane);
- $\mathcal{M}_i$, $1 \leq i \leq q$, are multisets of objects initially situated in corresponding membrane i;
- $\mathcal{R}_i$ is a set of antiport and symport rules placed in membrane i, $1 \leq i \leq q$. The antiport rules have form $(u, out; v, in)$, where $u, v \in \Gamma^*$; while symport rules have forms (u, in) and (u, out), where $u \in \Gamma^*$. Note that if rules $(u, in) \in R_1$ exist, then $u \cap (\Gamma \setminus \mathcal{E}) \neq \emptyset$,
- $i_{out} \in \{1, \ldots, q\}$ refers to the output membrane.

The meaning of antiport rules and symport rules has been given in Section 2. Here we emphasize the case of symport rules, if the environment is the parent region of region i, then the imported multiset should contain at least one symbol not in the environmental alphabet $\mathcal{E}$ (otherwise unlimited numbers of objects will be sent into the system in a single computational step).

In what follows, we give a simple example to demonstrate how a symport/antiport P system works.

Let $\Pi = (\{a, b\}, \{a, b\}, \{a, b\}, [\ [\ ]_2]_1, \{a\}, \emptyset, \mathcal{R}_1, \mathcal{R}_2), 1)$ be a symport/antiport P system, $\mathcal{R}_1 = \{(a, out; abb, in)\}$, $\mathcal{R}_2 = \{(a, in)\}$.

At the first step, by applying rule $(a, out; abb, in)$, in every computation step, the skin region obtains two copies of object b. Only when rule (a, in) is applied, the computation will halt, the numbers of object b presented in cell 1 is considered as the value of Π. Therefore, Π generates the set of numbers $\{2n \mid n \geq 0\}$.

In [72], it is shown that symport/antiport P systems are computationally complete. Furthermore, the register machine is simulated by antiport/symport P systems with single membrane [36, 37, 41]. As an extensive investigations, the computational power of such P systems using small or relatively small size parameters (for instance, the number of membranes, the length of rules, and symbols in multisets of rules) are equivalent to the register machines [3, 10, 12]. An interesting issue is to study the computational power of symport/antiport P systems with minimal parameters (number of objects or number of membranes).

3.4 Tissue-Like P Systems with Symport Rules and Antiport Rules

Definition 3.4. A tissue P system with symport rules and antiport rules (with degree $q \geq 1$) is a framework of the form

$$\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out}),$$

where:

- Γ is an alphabet, and every element in such alphabet is named objects;
- $\mathcal{E} \subseteq \Gamma$ is a set of alphabet of the environment, and it is assumed that the number of each object initial located in $\mathcal{E}$ is large enough;
- $\mathcal{M}_i, 1 \leq i \leq q$, are multisets of objects initially situated in corresponding cell i;
- $\mathcal{R}$ is a set of symport rules and antiport rules with the forms below:
  —Antiport rules: $(i, m/n, j)$, where $0 \leq i \neq j \leq q$, $m, n \in \Gamma^+$;
  —Symport rules: $(i, m/\lambda, j)$, where $0 \leq i \neq j \leq q$, $m \in \Gamma^+$;
- $i_{out} \in \{1, 2, \ldots, q\}$ refers to output cell and $i_{out} = 0$ refers to the environment.

An antiport rule $(i, m/n, j)$ is employed if cell i includes multiset m of objects and cell j includes multiset n of objects. When such an antiport rule is employed, region j receives multiset m from region i; meanwhile, region i receives multiset n from region j.

A symport rule $(i, m/\lambda, j)$ is employed if cell i includes multiset m of objects. When such a symport rule is employed, region j receives multiset m from region i.

A tissue P system with symport rules and antiport rules is worked in a maximally parallel mode. A *configuration* of a tissue P system with symport rules and antiport rules Π identify the multisets of objects present in its regions. The notions of transition, computation, and halting configuration are analogous to transition P systems as described above.

If cell division rules are introduced into tissue P systems with symport rules and antiport rules, then such systems are called *tissue P systems with symport rules and antiport rules and cell division*; when cell separation instead of cell division is incorporated into such P systems, then tissue P systems with symport rules and antiport rules and with cell separation were raised.

The form of division rules is $[\ a\ ]_i \rightarrow [\ b\ ]_i [\ c\ ]_i$ $(a, b, c \in \Gamma)$. Such a rule is applicable if the output cell is not cell i and cell i includes an object a. When a division rule is used, cell i is split into two cells i: object a triggered the rule is changed to b in one cell, while in the other cell object a is changed to c , and all the rest of objects are copied in these new produced cells.

Separation rules have form $[\ a\ ]_i \rightarrow [\ \Gamma_0\ ]_i [\ \Gamma_1\ ]_i$ ($a \in \Gamma$, Γ_0, Γ_1 are non-empty sets such that $\Gamma_0 \cup \Gamma_1 = \Gamma$ and $\Gamma_0 \cap \Gamma_1 = \emptyset$). Such a rule is applicable if cell i cannot be the output cell and it includes an object a. When a separation rule is used, the evolved cell i is split into two cells i, an object a triggered the rule is depleted, all the other objects are distributed in these new produced cells.

The model of tissue P systems with symport rules and antiport rules was raised in [73], and then in [39] such a basic model has been developed by introducing a conception of the state to communication channels, and the state can be revised if the corresponding communication rule is used. If rules involve one or two regions, tissue P systems with evolution-communication [11, 19] or with conditional uniport [115] were explored. Inspired by both the way transitions of the Petri nets and the symport/antiport paradigm, rules involve four regions (two regions are considered as inputs and the other two regions are considered as outputs) are studied, and generalized communication P systems were explored [26, 27, 56, 116], where such P systems simultaneously move objects from two regions to the other two regions. Such a purely communicating P system is a network of cells where the nodes are labeled and at any step of functioning contain a finite multiset of objects.

Due to the biological fact that molecules across a membrane are usually transferred from high concentration to low concentration, monodirectional tissue P systems were raised [106, 107], where only symport rules are allowed, and only one direction is allowed for two given regions when communication happens. Motivated by the communication rules (object is evolved when its position is changed) in P systems with active membranes, in [108], evolutional symport/antiport rules were investigated in tissue P systems such that objects may be changed during the movement process among regions.

A concept similar to tissue P system with evolutional symport rules and antiport rules is the P colony, introduced in [53]. For P colony, the *cells* (also called *agents*) have only one region that can be interacted with their joint environment by means of programs. There is a multiset of objects in each agent, and these objects are operated by a finite set of programs. Note that the number of objects in every agent remains unchanged during the computational process of the P colony. The agents place in the environment, where each object in such region is supposed to be in a countably infinite number of copies.

In [65] and [105], flat maximal parallelism of applying rules was explored, i.e., at every step, a maximal applicable set of rules in each membrane is selected, and every rule in the set can only be used once. In [70], the mechanism of flat maximal parallelism was studied in tissue P systems with promoters. Motivated from the mechanization of biological cell, many methods of producing new cells have been investigated, such as cell division and cell separation, where an exponential workspace is created, and a plenty of **NP**-complete problems have been solved via space-time tradeoff method. More specifically, in [82], cell division is incorporated into tissue P systems, such kind of new P systems was applied to solve the SAT problem (see [28] and [29] for more details). In [67], cell separation rules were employed in tissue P systems, and a complexity theory was studied in the tissue P systems with cell separation framework.

In [68], cell separation was incorporated into tissue P systems with evolutional symport rules and antiport rules, and a borderline between non-efficiency and efficiency is raised. According to different biological fact, many other varied tissue P systems were proposed, for instance, by introducing energy [2], promoters [103], proteins [24, 104], and the like. An obvious open problem is to introduce the biological features into tissue P systems, and study the computation power of such variant of P systems.

An example is given to illustrate the working of a tissue P system with symport rules and antiport rules.

Let $\Pi = (\{a, b\}, \{a, b\}, \{a\}, \emptyset, \mathcal{R}, 1)$ be a tissue P system with symport rules and antiport rules (having two cells with labels 1 and 2, respectively, with the environment being labeled 0), where R contains two rules

$$r_1 : (1, a/ab^2, 0);$$
$$r_2 : (1, a/\lambda, 0).$$

The computation for the tissue P system works as follows: Initially, cell 1 includes an object a, the environment includes the initial multiset $\{a, b\}$ (note that we assume that every initial object in the environment is large enough). At step 1, rule r_1 or r_2 is chosen non-deterministically. If rule r_2 is chosen , cell 2 receives an object a and the system halts; if rule r_1 is chosen , the environment receives an object a; meanwhile, cell 1 obtains an object a and two copies of b, so cell 1 increases two copies of b. Therefore, two copies of b are increased in cell 1 for each step whenever rule r_1 is selected. At some step, if rule r_2 is used, cell 2 receives an object a and the system halts. Hence, $2n$ ($n \geq 0$) copies of object b are produced in cell 1.

3.5 P Systems with Proteins on Membranes

Definition 3.5. A P system with proteins on membranes of degree $q \geq 1$ is a tuple

$$\Pi = (O, P, \mu, w_1/z_1, \ldots, w_q/z_q, \mathcal{E}, \mathcal{R}_1, \ldots, \mathcal{R}_q, i_{out}),$$

where:

- O and P are finite non-empty alphabets such that $O \cap P = \emptyset$;
- μ is a membrane structure with q membranes labeled by $1, \ldots, q$;
- $w_i, 1 \leq i \leq q$, are multisets over O;
- $z_i, 1 \leq i \leq q$, are multisets over P;
- $\mathcal{E} \subseteq O$ is a set of the alphabet of the environment, and it is assumed that the number of each object initial located in $\mathcal{E}$ is large enough;
- $i_{out} \in \{0, 1, \ldots, q\}$ refers to output region;
- $\mathcal{R}_i, 1 \leq i \leq q$, are sets of evolution rules with the forms below:
  (1) $[\, p \mid a \,]_i \rightarrow [\, p' \mid b \,]_i, p, p' \in P, a, b \in O, 1 \leq i \leq q$
  (The rule is available if membrane i contains object a and protein p is located on such membrane. By using such a rule, object a evolves to b, and protein p is revised to p'. Note that other objects and proteins associated with the evolved membrane remain unchanged.);
  (2) $a\,[\, p \mid \,]_i \rightarrow b\,[\, p' \mid \,]_i, p, p' \in P, a, b \in O, 1 \leq i \leq q,$
  (The rule is available if the father of membrane i contains object a and protein p is located on membrane i. By using such a rule, object a evolves to b, and protein p is revised to p'.);
  (3) $[\, p \mid a \,]_i \rightarrow b\,[\, p' \mid \,]_i, p, p' \in P, a, b \in O, 1 \leq i \leq q,$
  (The rule is available if membrane i contains object a and protein p is located on such membrane. By using such a rule, object a is sent out of membrane i, such object is revised to b during this process, and protein p is revised to p'.);
  (4) $a\,[\, p \mid \,]_i \rightarrow [\, p' \mid b \,]_i, p, p' \in P, a, b \in O, 1 \leq i \leq q,$
  (The rule is available if the father of membrane i contains object a and protein p is located on membrane i. By using such a rule, object a is sent into membrane i, such object is revised to b during this process, and protein p is revised to p'.);
  (5) $a\,[\, p \mid b \,]_i \rightarrow c\,[\, p' \mid d \,]_i, p, p' \in P, a, b, c, d \in O, 1 \leq i \leq q,$
  (The rule is available if the father of membrane i contains object a, membrane i has object a, and protein p is located on membrane i has object b. By using such a rule, object a is sent into membrane i, such object is revised to d; object b is sent out of membrane i, such object is revised to c, and protein p is revised to p'.).

A *configuration* of a P system with proteins on membranes is depicted by the current membrane structure, all multisets of objects in every membranes, and all multisets of proteins located on every membranes. The notions of transition, computation, and halting computation are analogous to transition P systems as described above.

The computational power of P systems with proteins on membranes was studied widely [74, 75, 112], result shown that P systems with proteins on membranes are universal by combining various types of rules. In [55] and [77], flip-flop membrane proteins are considered, where a protein has at most two states, and this kind of P systems were proved to be universal. In [76], membrane division rules are incorporated into P systems with proteins on membranes, and a uniform solution to the SAT problem was presented by a family of P systems with proteins on membranes and

membrane division. If division rules are allowed for non-elementary, then P systems with proteins on membranes can solve **PSPACE**-complete problems [112].

3.6 Spiking Neural P Systems

Neurons are one of the most interesting cell types in the human body. A large number of neurons working in a cooperative manner are able to perform complex tasks. Inspired from the way of neurons communicate by means of electrical impulses, spiking neural P systems were proposed in [49]

Definition 3.6. A *spiking neural P system* (for short, *SN P system*) (with degree $m \geq 1$) that was introduced in [23] is a framework of the form

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

where:

- $O = \{a\}$ is an alphabet with singleton element a, which is called spike;
- $\sigma_1, \ldots, \sigma_m$ are neurons with form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where
  (a) $n_i \geq 0$ is a natural number, which indicates the number of spikes initially placed in σ_i;
  (b) R_i is a set of rules that have form: $E/a^c \rightarrow a^p; d$, where E is a regular expression, $c \geq 1$, $c \geq p \geq 0$, and $d \geq 0$ are integer numbers; specifically, d is the delay, which represents the duration between employing firing rule and releasing the spike;
- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ are synapses (if two neurons are connected), and $(i, i) \notin syn$ for $1 \leq i \leq m$;
- $out, in \in \{1, 2, \ldots, m\}$ refer to output and input neurons, respectively.

For rules of form $E/a^c \rightarrow a^p; d$, if $p = 1$, then such a rule is called a *standard rule*; and if $p = 0$, then such a rule is called a *forgetting rule*, hence forgetting rules and standard rules are two special cases.

A rule $E/a^c \rightarrow a^p; d$ is employed in neuron σ_i if the following two requirements are satisfied: (1) there exists k spikes in neuron σ_i, $a^k \in L(E)$; (2) $k \geq c$, the number of spikes depleted by the rule must be less than the number of spikes in neuron σ_i. When such a rule is employed, neuron σ_i consumes c spikes, and after d steps, p copies of spikes are created. Especially when $d = 0$, the generated p spikes are duplicated and released immediately to all its syndetic neurons; otherwise, neuron σ_i is closed in these d computation steps, that is, if at step t, a firing rule is employed, then neuron σ_i is shut from step t to step $t + d - 1$, hence new spikes cannot be received from the closed neuron. Note that, if any spike is sent to the closed neuron, then that spike will be lost.

A forgetting rule $E/a^c \rightarrow \lambda$ is used in neuron σ_i if $a^k \in L(E)$, $k \geq c$ and firing rules cannot be employed. By employing such a forgetting rule, neuron σ_i decreases c spikes.

A configuration of an SN P system at some instant consists of the number of spikes in every neuron and the state of each neuron (the number of steps before the neuron opens; if the state of a neuron is already open, then the number of steps is zero). Therefore, a configuration of Π is expressed as follows: $\langle r_1/t_1, \ldots, r_m/t_m \rangle$ for $r_i \geq 0$ and $t_i \geq 0$, where r_i spikes are included in neuron σ_i, and after t_i ($i = 1, 2, \ldots, m$) steps it will become open. The notions of transition, computation, and halting computation are analogous to transition P systems as described above.

The result of an SN P system is represented in several ways: (1) the interval of time steps (distance) between the first two spikes [49, 83]; (2) the output neuron emits all spikes to the environment when system halts [20]; (3) the interval of time steps between all consecutive spikes. Thus, an SN P system is regarded as a generator over the alphabet $\{0, 1\}$ [22].

With the inspirations of different biological phenomena, and mathematical motivations, various types of SN P systems were explored, for instance, SN P systems with polarizations by taking the

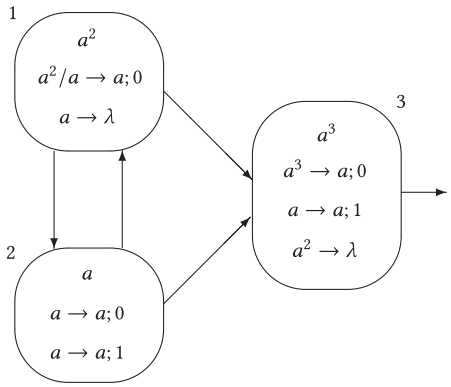

Fig. 4. An example of SN P system.

polarized membrane of a neuron as inspiration [128]; motivated by the biological fact that, if the membrane threshold potential is inferior to its membrane potential, then such neurons can fire; SN P systems with weights were investigated [117]; SN P systems with astrocytes inspired by the excitatory and inhibitory functioning of astrocytes on synapses [69]; aroused by Eckhorn's neuron model, coupled neural P systems were designed in [87]; axon P systems motivated by the information processing of axons in the nervous system, which have a powerful computational power, and such P systems were proved to be universal as both function computing devices and number generator devices [139]; SN P systems with structural plasticity were considered, inspired by the idea from the self-organizing and self-adaptive feature of artificial neural networks [17, 123]; motivated by the feature of request-response in grammar systems in [66], SN P systems with communication on request were investigated; cell-like SN P systems were described in [129] by incorporating the feature of hierarchical arrangement of membrane structure; dendrite P systems [86] aroused by the way dendrites lead the computation over neurons.

In [16], the notion of scheduled synapses was considered by spiking neural P systems, where a duration is combined with every synapse, which is valid only in the duration of its schedule. Therefore, it is worth studying the computational power of various SN P systems with scheduled rules, where each synapse is associated with a duration.

An example is given to illustrate how an SN P system works. Let Π be an SN P system (see Figure 4), which includes neurons σ_1, σ_2, and σ_3, and neuron σ_3 is the output one.

First of all, all neurons σ_1, σ_2, σ_3 fire and the two rules in neuron σ_2 are chosen non-deterministically. At step 1, the output neuron σ_3 obtains one spike from neuron σ_1 and neuron σ_2, respectively; these two spikes will be depleted later. The spikes between neuron σ_1 and neuron σ_2 are also exchanged; therefore, so long as the rule $a^2 \to a; 0$ in neuron σ_2 is employed, neuron σ_1 gets one spike, thus two spikes are obtained, and neuron σ_1 can fire again.

Whenever the rule $a \to a; 1$ in neuron σ_2 is chosen and applied, then neuron σ_2 cannot receive the spike of neuron σ_1, and only neuron σ_3 receives one spike; thus, if the number of spikes stays empty in neuron σ_2, it will no longer work. Next, only forgetting rule $a \to \lambda$ in neuron σ_1 can be used, simultaneously rule $a^2 \to a; 1$ in neuron σ_3 fires; meanwhile, neuron σ_2 emits one spike, but neuron σ_3 cannot receive this spike because it is shut at this step; neuron σ_1 receives one spike, which will be forgotten later. Finally, neuron σ_3 sends one spike to the environment, and the system halts. Consequently, the SN P system Π produces the set of natural numbers greater than 1.

3.7 Enzymatic Numerical P Systems

An enzymatic numerical P system (shortly, ENP system) of degree $q \geq 1$ that was introduced in [85] is a framework of the form

$$\Pi = (\mu, (Var_1, E_1, Pr_1, Var_1(0)), \ldots, (Var_q, E_q, Pr_q, Var_q(0))),$$

where

—μ is a hierarchical membrane structure;
—every membrane i is characterized by a four-tuple $(Var_i, E_i, Pr_i, Var_i(0))$, $1 \leq i \leq q$, where:
(a) Var_i is a set of variables placed in membrane i;
(b) $E_i \subseteq Var_i$ is a finite set of enzyme variables placed in membrane i;
(c) Pr_i is the set of programs placed in corresponding membrane i, which are the following two cases:
(i) non-enzymatic form

$$F_{l,i}(x_{1,i}, \ldots, x_{k_i,i}) \rightarrow c_{l,1} \mid v_1 + \cdots + c_{l,n_i} \mid v_{n_i},$$

where $F_{l,i}(x_{1,i}, \ldots, x_{k_i,i})$ refers to production function, $c_{l,1} \mid v_1 + \cdots + c_{l,n_i} \mid v_{n_i}$ refers to repartition protocol;
(ii) enzymatic form

$$F_{l,i}(x_{1,i}, \ldots, x_{k_i,i}) \mid_{e_{j,i}} \rightarrow c_{l,1} \mid v_1 + \cdots + c_{l,n_i} \mid v_{n_i},$$

where $e_{j,i}$ refers to an enzyme variable from Var_i, but they are different from $v_1, \ldots, v_{n_i}$ and from $x_{1,i}, \ldots, x_{k_i,i}$;
(d) $Var_i(0)$ is a set of original values of variables in corresponding membrane i.

If all programs are non-enzymatic form, then such P systems are called *standard numerical P systems* (shortly, *NP systems*) [81].

A non-enzymatic program is executed as follows: At a moment $t \geq 0$, the system calculates a production value $F_{l,i}(x_{1,i}(t), \ldots, x_{k_i,i}(t))$ by taking the current values of variables $x_{1,i}, \ldots, x_{k_i,i}$ belonging to compartment i. Afterwards, according to the distribution coefficients $c_{l,1}, \ldots, c_{l,n_i}$, the production value $F_{l,i}(x_{1,i}(t), \ldots, x_{k_i,i}(t))$ is distributed to each variable $v_1, \ldots, v_{n_i}$ from compartment i, the upper compartment, or the immediately inner compartments. Specifically, each variable v_s is associated a value $q_{l,i}(t) \cdot c_{l,s}$, $1 \leq s \leq n_i$, where $q_{l,i}(t)$ is computed as follows:

$$q_{l,i}(t) = \frac{F_{l,i}(x_{1,i}(t), \ldots, x_{k_i,i}(t))}{\sum_{s=1}^{n_i} c_{l,s}}.$$

Note that when the production value has been computed, the variables in the corresponding production function will be reset to zero; otherwise, they retain the current value. After repartition, the quantities assigned to each variable from several repartition protocols are added to the current value of these variables. If there are multiple programs that can be executed in a compartment at some moment, then the common strategy adopted is that all these applicable programs are simultaneously executed, called the *all-parallel mode*. The programs are also worked in one-parallel mode (using programs in the all-parallel mode under the restrictive condition that one variable can present in only one of the applied programs; in the case of multiple choices, the programs to be used are chosen in a non-deterministic way) or sequential mode (only one program is used in a step in every membrane; if more than one program in a membrane is available, then one of them is non-deterministically chosen). For more details, please refer to [81] and [114].

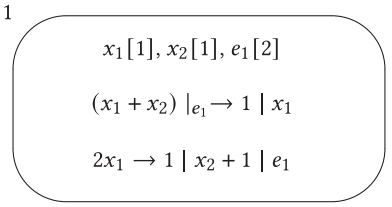

Fig. 5. An ENP system Π generating Fibonacci sequence.

An enzymatic program is employed at an instant t if $e_{j,i} > \min\{x_{1,i}(t), \ldots, x_{k_i,i}(t)\}$. In other words, enzyme variables are employed to control the use of programs. The execution of enzymatic programs is identical to that of non-enzymatic programs.

A configuration of an ENP system Π at an instant $t \geq 0$ consists of the values of all variables of the system, represented by $C_t = \langle x_{1,1}(t), \ldots, x_{k_1,1}(t), \ldots, x_{1,m}(t), \ldots, x_{k_m,m}(t)\rangle$, where the coordinate $x_{j,i}(t) \in \mathbb{R}$ indicates the value of variable $x_{j,i}$ from compartment i at current time t, for $1 \leq i \leq m$, $1 \leq j \leq k_i$, and $t \geq 1$. From one configuration to another, one is viewed as a transition, that is, at instant t, given the values of variables $x_{j,i}(t)$, the values of variables $x_{j,i}(t+1)$ at instant $t+1$ is calculated through the use of the corresponding programs. A sequence of transitions is called a computation. If all programs cannot be used, then the system is said to achieve a halting configuration. The calculation results of an ENP system are the values of the designated variables during the computation.

An example is given to illustrate how an ENP system works. To be specific, an ENP system Π_1 is defined in Figure 5. The ENP system Π_1 has one membrane, and the initial values for each variable are specified in square brackets, and the variable x_1 is designated as the output .

At each step $t \geq 0$, the value of enzyme variable e_1 is increased by value $x_1(t)$, thus $e_1(t) > \min\{x_1(t), x_2(t)\}$ is always true; in this way, programs $(x_1 + x_2) \mid_{e_1} \rightarrow 1 \mid x_1$ and $2x_1 \rightarrow 1 \mid x_2 + 1 \mid e_1$ are applied all the time. Hence, at each step t, the value of variable x_2 becomes the value $x_1(t)$, while the value of variable x_1 becomes the value $x_1(t) + x_2(t)$. As variable x_1 is an output one, the result of such ENP system is defined by the value of variable x_1 in the computation process, thus the ENP system Π generates Fibonacci sequence.

Several variants of NP systems have been proposed. For example, by considering the idea of the threshold control adopted in SN P systems, several other control strategies of the execution of programs have also been introduced in NP systems, such as NP systems with the variable threshold control [140], NP systems with the production threshold control [71], and NP systems with boolean condition control [60]. Inspired by objects can pass through the membranes, NP systems with migrating variables were put forward [141].

4 THEORY OF MEMBRANE COMPUTING

4.1 Computing Power

The initial motivation of membrane computing aimed at model computation inspired by membrane structure of biological cells. After 20 years of development, plenty of models were presented and their computing power was investigated. It turned out that universality holds for most of P systems both for symbol and string objects, working in an accepting or in a generative ways. Universality holds for these classes of P systems in their most general form, but also for their quite restricted forms, with restrictions on the number of membranes, form of the rules, and the like. In most cases, trade-offs (retaining the computing power) were found among the complexity of rules, the number of membranes or objects used. Thus, on this very abstract level one can conclude that

"the cell is a powerful computer", both when it works alone and in tissue-like or neural-like network configurations. In what follows, some basic results with regard to the computational power of transition P systems are presented.

The results of all the computations produced by transition P system are denoted by $N(\Pi)$. We denote by $NOP_m(\alpha, tar)$ the family of all sets of numbers produced by transition P systems, where m represents the number of membranes, α represents a type of rules, if $\alpha = coo$, it means cooperating rules are applied; if $\alpha = ncoo$, it means all rules are non-cooperating; if $\alpha = cat$, then rules with catalysts are applied. If the number of membranes is not bounded for transition P systems, we can substitute $*$ for the symbol m.

Due to the definitions, the following relations are obvious:

Lemma 4.1.
(1) For all $m \geq 1$, we have $NOP_m(ncoo, tar) \subseteq NOP_m(cat, tar) \subseteq NOP_m(coo, tar)$;
(2) $NOP_*(ncoo, tar) \subseteq NOP_*(cat, tar) \subseteq NOP_*(coo, tar)$;
(3) For all $m \geq 1$ and $\alpha \in \{coo, cat, ncoo\}$, we have $NOP_m(\alpha, tar) \subseteq NOP_{m+1}(\alpha, tar)$.

We denote by Σ a set of terminal objects, and the computation result of transition P systems is counted by the number of the symbols of $\Sigma \subseteq \Gamma$ in the specified output membrane at the end of the halting computations. An interesting case of transition P systems is when we consider $\Sigma = \Gamma - C$ (C is a alphabet of catalysts), thus the set of nutural numbers or vectors produced by transition P system are denoted by $N_{-C}(\Pi)$ or $Ps_{-C}(\Pi)$, and the families of all sets of numbers or sets of vectors computed by transition P systems with at most m membranes and the set of catalysts having at most k elements are denoted by $NO_{-C}P_m([p]cat_k)$ or $PsO_{-C}P_m([p]cat_k)$.

The following results based on transition P systems with (purely) catalysts are obtained (the proof of the following theorems can be found in Chapter 4 in [84]).

Theorem 4.2. *$NO_{-C}P_1(cat_2) = NO_{-C}P_1(pcat_3) = NRE$.*

Corollary 4.3. *For any $m \geq 1$ and $k \geq 2$,*

$$PsRE = PsO_{-C}P_m(cat_k) = PsO_{-C}P_m(pcat_{k+1}).$$

We obtain the following general result as a counterpart to Corollary 4.3.

Corollary 4.4. *For any $m \geq 2$ and $k \geq 2$,*

$$PsRE = PsOP_m(cat_k) = PsOP_m(pcat_{k+1}).$$

Next, we show that transition P systems with no catalyst and purely catalytic P systems with one catalyst can only generate semilinear sets.

Theorem 4.5. *$PsO_{-C}P_1(cat_0) = PsO_{-C}P_1(pcat_1) = PsREG$.*

4.2 Generation of Languages

Catalytic P systems and purely catalytic P systems can also be used as language generators: During a successful computation, all the symbols sent out through the skin membrane are taken as the symbols forming a string in just that sequence from which the symbols are sent out. (We take all possible sequences of symbols that are sent out in one transition step as possible substrings to be concatenated with the string already generated by the preceding transition steps—thus, not only one string may be the result of a successful computation.) The language generated by a (purely) catalytic P system Π in that way is denoted by $L(\Pi)$. The family of languages generated by [purely] catalytic P systems with at most m membranes and at most k catalysts is denoted by $LOP_m([p]cat_k)$.

Lemma 4.6. *For any partially recursive function $f : \mathbf{N} \to \mathbf{N}$ there exists a register machine M with two registers computing f in such a way that, when starting with 2^n in register 1 and 0 in register 2,*

M computes $f(n)$ by halting with $2^{f(n)}$ in register 1 and 0 in register 2. Moreover, in no configuration both registers are empty.

The following results based on (purely) catalytic P systems are obtained (see chapter 4 in [84]).

THEOREM 4.7. *$LOP_1(cat_2) = LOP_1(pcat_3) = RE$.*

THEOREM 4.8. *$LOP_1(bicat_1) = RE$, where bicat represents bistable catalysts, i.e., the catalytic may switch between two states.*

Symport/antiport P systems as language generators were also studied. More specifically, during a successful computation, all the terminal objects sent out by means of the skin membrane are taken as the objects forming a string in just that sequence the objects are sent out. If there is more than one object is sent out in one step (either by only one copy of a rule or by several rules from the applied multiset of rules), then we take all possible sequences of objects that are sent out in one transition step as possible substrings to be concatenated with the string already generated by the preceding transition steps, as we take all possible substrings from each transition step. So not only one string may be the result of a successful computation. The language generated by a symport/antiport P system Π in the above-mentioned way is denoted by $L(\Pi)$. The family of languages generated by symport/antiport P systems with at most m membranes using antiport/symport rules of type α ($\alpha \in \{sym_k, anti_k\}$), using the transition mode $X \in \{max, amin\}$, are denoted by $LOP_m(\alpha, X)$, where sym_k (respectively, $anti_k$) represents symport rules (respectively,, antiport rules) of length at most k are employed.

The following results based on symport/antiport P systems are obtained (the proof of the following theorems can be found in chapter 5 in [84]).

THEOREM 4.9. *$RE = LOP_1(anti_3, max) = LOP_1(sym_3, max)$.*

For the minimally parallel mode we also get a characterization of *RE* with the minimal number of membranes being two.

THEOREM 4.10. *$RE = LOP_2(anti_3, amin) = LOP_2(sym_3, amin)$.*

4.3 Computational Efficiency

The first foundations of a complexity theory in Membrane Computing were given in [97] and [98]. In the seminal paper of Membrane Computing discipline, the models defined are (cell-like) P systems with output membrane but without input membrane, where a single initial configuration given by the initial multisets over the working alphabet is associated with such P systems. In [98], *P systems with input membrane* have been designed, where an input alphabet Σ of such kinds of P systems is included in working alphabet Γ, and initial multisets located in input membrane are over $\Gamma \setminus \Sigma$.

4.3.1 Recognizer Membrane Systems. Bearing in mind that the solvability of decision problems is associated with the recognition of languages, so recognizer P systems are considered in the Membrane Computing framework.

Definition 4.11. A *recognizer membrane system* is a membrane system (with or without input membrane) that satisfies the following conditions: (a) two special objects (`no` and `yes`) are included in working alphabet Γ; (b) the initial multisets of a P system without input membrane are over Γ; however, the initial multisets of a P system with input membrane are over $\Gamma \setminus \Sigma$. Σ is an input alphabet of the system; (c) all computations of a recognizer membrane system halt; (d) if a computation C exists, then only one of the objects *no, yes* should have been delivered to the output region at the final step.

A computation C of recognizer P systems is named an *rejecting (respectively, accepting) computation* when object no (respectively, yes) appears in the output region when systems halt. Note that every computation in a recognizer membrane system is a halting computation.

These concepts are extended to tissue P systems motivated by intercellular communication in organs or tissues. Specifically, if Γ, Σ, and $\mathcal{E}$ represent the working alphabet, input alphabet. and environment alphabet, respectively, then $\mathcal{E} \subseteq \Gamma \setminus \Sigma$ and the initial multisets of a tissue P system are over $\Gamma \setminus \Sigma$.

4.3.2 Polynomial Time Complexity Classes. In [58], [79], and [132], P systems without input membrane were employed as an effective tool to offer efficient solutions to **NP**-complete problems. This kind of solution is considered as *special-purpose* solutions; for every instance of the problem, a corresponding P system is designed such that the syntax of this given instance is a portion of the designed P system.

Semi-Uniform Solutions. Next, following [98], the *special-purpose* solutions are defined in a mathematical way, called *semi-uniform solutions.*

Definition 4.12. A decision problem $X = (I_X, \theta_X)$ is efficiently solved in a *semi-uniform* manner by means of a family $\{\Pi(u) \mid u \in I_X\}$ of recognizer P systems without input membrane from $\mathcal{R}$ (denoted by $X \in \mathbf{PMC}^*_{\mathcal{R}}$) if the following prerequisites are satisfied:

- The family Π is polynomially uniform by Turing machines, more specifically, a deterministic Turing machine, which can design the system $\Pi(u)$ on the basis of the instance $u \in I_X$ processing in polynomial time exists.
- The family Π is polynomially bounded, i.e., there is a natural number $k \in \mathbb{N}$, and every computation of $\Pi(u)$ implements at most $|u|^k$ steps for every specific instance $u \in I_X$.
- The family Π is complete (for every specific instance $u \in I_X$, if $\theta_X(u) = 1$, then allcomputations of $\Pi(u)$ are accepting computations) and sound (for every instance $u \in I_X$, when an accepting computation of $\Pi(u)$ exists, then $\theta_X(u) = 1$) in reference to X.

According to Definition 4.12, the following statements are achieved:

—A *semi-uniform solution* to the decision problem X is offered by the family $\{\Pi(u) \mid u \in I_X\}$.
—A system $\Pi(u)$ is designed to process each instance $u \in I_X$. Moreover, according to the notions of completeness and soundness of the family in reference to the decision problem X, clearly, the system $\Pi(u)$ is *confluent*, i.e., the same result is obtained for all computations (either all computations are rejecting computations or all computations are accepting computations).

Uniform Solutions. Next, another kind of solution to decision problems by families of recognizer P systems is introduced. More specifically, all instances (with the *same size*) of the problem are solved via a system combined with an appropriate input.

Definition 4.13. A decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time and in a *uniform* way by a family $\Pi = \{\Pi(n) \mid n \in \mathbb{N}\}$ of membrane systems from $\mathcal{R}$ (denoted by $X \in \mathbf{PMC}_{\mathcal{R}}$) if the following conditions hold:

- *The family* Π *is polynomially uniform by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system* $\Pi(n)$ *with input membrane from* $n \in \mathbb{N}$*;*
- *There exists a pair* (cod, s) *of polynomial-time computable functions over* I_X *such that:*

—*For each instance* $u \in I_X$, $s(u)$ *is a natural number and* $cod(u)$ *is an input multiset of the system* $\Pi(s(u))$*; we denote by* $\Pi(s(u)) + cod(u)$ *the system obtained by adding* $cod(u)$ *to the multiset in the region* i_{in} *of* $\Pi(s(u))$*;*
—*For each* $n \in \mathbb{N}$, $s^{-1}(n)$ *is a finite set;*
—*The family* Π *is polynomially bounded with regard to* (X, cod, s)*, that is, there exists a polynomial function* p*, such that, for each* $u \in I_X$*, every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is halting and it performs at most* $p(|u|)$ *steps;*
—*The family* Π *is sound with regard to* (X, cod, s)*, that is, for each* $u \in I_X$*, if there exists an accepting computation of* $\Pi(s(u))$ *with input* $cod(u)$*, then* $\theta_X(u) = 1$*;*
—*The family* Π *is complete with regard to* (X, cod, s)*, that is, for each* $u \in I_X$*, if* $\theta_X(u) = 1$*, then every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting one.*

According to Definition 4.13, the following statements are achieved:

—A *uniform solution* to the decision problem X is offered by the family $\{\Pi(n) \mid n \in \mathbb{N}\}$. The ordered two-tuples (cod, s) is a *polynomial encoding* of X in $\{\Pi(n) \mid n \in \mathbb{N}\}$;
—A system $\Pi(s(u)) + cod(u)$ is designed to process every instance $u \in I_X$. Moreover, the system $\Pi(s(u)) + cod(u)$ is *confluent*, i.e., the same result is obtained for all computations (either all computations are rejecting computations or all computations are accepting computations).

On the basis of the definitions of recognizer P systems, the aforementioned complexity classes are closed both under polynomial-time reductions and under complement [99].

From Definition 4.12 and Definition 4.13, it is easy to know that each uniform solution of a decision problem can be viewed as a semi-uniform solution applying the same number of computing resources, i.e., for any class $\mathcal{R}$ of recognizer membrane systems, $\mathbf{PMC}_{\mathcal{R}} \subseteq \mathbf{PMC}^{*}_{\mathcal{R}}$. It has been proved that the concept uniformity solution is strictly weaker than semi-uniformity solution for P systems [63].

4.3.3 Limits on Efficient Computations. In this subsection, the limitations of polynomial time computations in membrane systems is analyzed. With respect to cell-like membrane systems, two interesting results were established [46]:

—A family of recognizer transition P systems can simulate every deterministic Turing machine processing in polynomial time;
—If a family of recognizer transition P systems can solve a decision problem, then a deterministic Turing machine is existed, which can efficiently solve such decision problem.

Consequently, the computational power of basic recognizer transition P systems is bounded, where only the class of problems in **P** is solved. A similar result is also available for recognizer tissue P systems. Actually, recognizer tissue P systems are simulated via recognizer transition P systems (see [30] for details).

4.3.4 Solving Computationally Hard Problems. In accordance with the previous section, for the sake of offering efficient solutions to computationally hard problems, therefore, it is necessary to raise the number of processor units (cells or membranes) during a computation in membrane systems. Specifically, the membrane system should have the ability of trading space for time by providing an exponential workspace generated in a reasonable time. This capability has been implemented by using different mechanisms inspired by the cellular mitosis (division rules), autopoiesis (creation rules) or membrane fission (separation rules), among others.

Cell-like membrane systems. P systems with active membranes are a kind of basic cell-like P systems [79], where each membrane is combined with an electrical charge. From Definition 3.2, it is known that each type of rules is non-cooperative, and by applying a rule, both the evolved membrane charge and object can be revised.

The class of recognizer P systems with active membranes (respecively, division of both non-elementary and elementary membranes or only of elementary membranes) is represented by $\mathcal{AM}$ (resp., $\mathcal{AM}(+ne)$ or $\mathcal{AM}(-ne)$). Many strongly **NP**-complete problems (`Bin Packing` [95], `Clique` [6], `SAT` [97], `Common Algorithmic Problem` [96]) and weakly **NP**-complete problems (`Partition` [44], `Knapsack` [93], `Subset Sum` [92]) were solved in the framework of $\mathcal{AM}(-ne)$ in polynomial time. In [5], a family of recognizer P systems from $\mathcal{AM}(+ne)$ can solve the *quantified Boolean formula* (`QBF-SAT`) problem.

In [100], it is shown that a computation of any confluent recognizer P system with active membranes can be simulated by means of a deterministic and efficient algorithm. Moreover, a deterministic Turing machine taking a time of the order $O(2^{p(n)})$ ($p(n)$ is a polynomial) can simulate any confluent recognizer P system with active membranes. Hence the result **PSPACE** $\subseteq$ $\mathbf{PMC}_{\mathcal{AM}(+ne)} \subseteq \mathbf{PMC}^{*}_{\mathcal{AM}(+ne)} \subseteq$ **EXP** is achieved. In [113], the complexity class **PSPACE** has been characterized by $\mathbf{PMC}_{\mathcal{AM}(+ne)}$.

According to the previous results, we know that P systems with active membranes are too powerful for solving decision problems from a perspective of computational complexity. Thus, polarizationless P systems with active membranes are proposed, and the class of all recognizer polarizationless P systems with active membranes is denoted by $\mathcal{AM}^{0}(\alpha, \beta)$, (a) if $\alpha = +d$ (respectively, $\alpha = -d$), it means the systems allow (respectively, forbid) the use of dissolution rules; (b) if $\beta = +ne$ (respectively, $-ne$), it means the systems allow the use of division of both non-elementary and elementary membranes (respectively, only of elementary membranes). It is worth pointing out the relevant role played by dissolution rules in the framework of $\mathcal{AM}^{0}(\alpha, \beta)$ from a complexity view. The result from [45] shows that $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^{0}(-d,-ne)} = \mathbf{PMC}_{\mathcal{AM}^{0}(-d,+ne)}$; besides, a solution to the `QBF-SAT` problem via membrane systems from $\mathcal{AM}^{0}(+d, +ne)$, has been given [9]; i.e., $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\mathcal{AM}^{0}(+d,+ne)}$.

Tissue-Like Membrane Systems. The class of recognizer tissue P systems with cell division (respectively, without environment) is denoted by $\mathcal{TDC}$ (respectively, $\widetilde{\mathcal{TDC}}$). We represent by $\mathcal{TDC}(k)$ (respectively, $\widetilde{\mathcal{TDC}(k)}$) the class of such tissue P systems (respectively, without environment), where communication rules of length at most k are allowed. It is worth pointing out that by applying the dependency graph method ([45]), it was shown that recognizer tissue P systems from $\mathcal{TDC}(1)$ can only solve tractable problems, $\mathbf{P} = \mathbf{PMC}_{\mathcal{TDC}(1)}$ [43]. However, in [42], a solution to the `HAM-CYCLE` problem was presented by a family of recognizer systems from $\mathcal{TDC}(2)$.

In [94], it is proved that any tissue P system with environment and with cell division can be efficiently simulated by a tissue P system without environment and with cell division, that is, we have $\mathbf{PMC}_{\mathcal{TDC}(k)} = \mathbf{PMC}_{\widetilde{\mathcal{TDC}(k)}}$ (for each $k \geq 1$). Hence, the function of the environment in recognizer tissue P systems with cell division framework is irrelevant from a perspective of computational complexity.

5 APPLICATIONS OF MEMBRANE COMPUTING

The applications to computer science are quite different: computer graphics, sorting/ranking, simulating and modeling circuits, cryptography, parallel architectures, and so on. In the past two decades, models of P systems have been employed to process a wide spectrum of application problems [137]. Here we only focus on the brief introduction of autonomous mobile robots path planning, fault diagnosis with spiking neural P systems, and other applications.

5.1 Path Planning and Control of Mobile Robots

Robots are physical devices acting in the real world and interacting with human beings and environment elements by means of sensors, motors, and computation units. In summary, the motion planning problem is defined below: *Given a geometric description of the robot, two states (a goal state and a start state), and the location of the obstacles in the environment, find a sequence of commands that controls to move the robot from the start state to the goal state.* The classical solution divides the problem in three subproblems: *global planning*, *local planning*, and *PID control*, where:

- The global planning considers the location of static obstacles in the environment (a precomputed map should be given as input), as well as the starting and goal positions. It computes a path avoiding obstacles before the robot starts to move.
  There are two versions of the global planning problem:
  — *The Feasibility Problem.* The goal is to discover a feasible path, if one exists.
  — *The Optimality Problem.* The goal is to find a feasible path with minimal cost where the cost is given by a computable function.
- The local planning takes the given path and tries to move the robot following the route while considering kinematic and physical constraints as well as avoiding both static and dynamic obstacles. The output of the local planner is a sequence of velocity references in an open loop.
- The PID control takes each velocity reference and commands the motors in order to maintain a constant velocity as close as possible to the velocity reference until the next reference is given.

One of the most important features of any control system in robotics is the need to provide real-time solutions to hard problems such as navigation with kinematic constraints in enviroments populated with static and dynamic obstacles. Therefore, most of the classical algorithms in robotics can only provide approximate solutions given a fixed time of computation. Several authors have studied how to accelerate such algorithms by using parallel hardware, i.e., GPU or FPGA, but adapting the algorithms to such a hardware is a hard task. On the other hand, instead of accelerating the conventional algorithms with parallel hardware, an intermediate computing paradigm can be used. In this sense, membrane computing is an inherently parallel computing paradigm with a large variety of simulators able to reproduce computations over parallel hardware. Thus, membrane computing can be used to model solutions to hard problems in robotics and well-studied hardware simulators can be used to apply the solutions to real case studies. The main advantage of this approximation is to apply the computational power of membrane computing which is intrisically parallel, as well as, to use well-studied and robust simulation algorithms for parallel hardware. On the other hand, the main disadvantage is the complication of adapting such algorithms to membrane computing, which requires an expert knowledge both in robotics and membrane computing.

Membrane computing provides a new point of view for implementing in parallel architectures such algorithms. However, due to the difficulty of working in a continuous space state in classical membrane system models, enzymatic numerical P systems (ENP systems) [85] were selected to model PID controllers (note that ENP systems use variables containing real numbers instead of multisets of objects).

Up to now several *membrane controllers* have been designed. In [15], the membrane controllers for mobile robots is formulated. In [124], a multi-behavior robot coordination controller is designed based on ENP systems. In [14], some current results and challenges about membrane controllers are presented. In [125], an original trajectory tracking control method based on ENP systems was

employed to solve the nonholonomic wheeled mobile robots problem. The control of robot swarms was also studied in [31], [32], and [33] in the framework of P colonies.

The global planning problem has also been attacked by using P systems. In [126], a membrane algorithm based on particle swarm optimization has been employed to solve the path planning problem for mobile robots. In [91], the rapidly exploring random tree (RRT) strategy was used for solving the feasibility problem in global planning together with random enzymatic numerical P systems with shared memory. The RRT is a classical approach in robotics, which includes the RRT algorithm (often used to solve the feasibility problem) and the RRT* algorithm (often used to solve the optimality problem). In [90], the RRT and RRT* algorithms were modeled and simulated by using ENP systems, which include parallel implementations in CUDA and OpenMP with several comparisons.

5.2 Fault Diagnosis with Spiking Neural P Systems

The fuzzy spiking neural P systems (FRSN P systems) was introduced in [89] and it can deal with the representation of fuzzy knowledge and complete fuzzy reasoning. Moreover, it is an ideal model to solve the problem of fault diagnosis having properties such as parallel computing advantage, high understandability, dynamic feature, synchronization, non-determination, and non-linearity. The structure of FSN P systems is described as follows:

Definition 5.1. An FRSN P system of degree $m \geq 1$, is a construct of the form

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

where

- $A = \{a\}$ is the singleton alphabet (the object a is called a *spike*);
- $r_1, \ldots, r_m$ are neurons having the form $r_i = (\alpha_i, \tau_i, r_i)$ with $i \in \{1, \ldots, m\}$, where
  - (i) $\alpha_i \in [0, 1]$ represents the (potential) value of spike contained in neuron σ_i (also called the *pulse value*);
  - (ii) $\tau_i \in [0, 1]$ represents the truth value associated with neuron σ_i;
  - (iii) r_i is a firing/spiking rule contained in neuron σ_i, and it is of the form $E/a^\alpha \rightarrow a^\beta$, where $a, b \in [0, 1]$.
- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $i \neq j$ for all $(i, j) \in syn, 1 \leq i, j \leq m$ (synapses between neurons);
- I and O represent the input neuron set and output neuron set, respectively.

Neurons in FRSN P system are classified into three classes: proposition neuron, *AND*-type rule neuron, and *OR*-type rule neuron. The reasoning algorithm in [89] is based on the firing mechanism of the neurons and it also uses matrix operations. Moreover, the model in [89] is capable of representing the fuzzy production rules of a fuzzy diagnosis knowledge base visually. It can also effectively model the corresponding dynamic reasoning behavior. In 2013, an approach for the fault diagnosis of power systems was proposed in [131] based on fuzzy reasoning spiking neural P systems (FRSN P systems). Moreover, the proposed method can diagnose different single and multiple faults along with failed malfunctioned protective devices. Only simple matrix operations are required for these models. Also the FRSN P systems based diagnostic model in [131] have good fault tolerance capacity. These models can be constructed in advance and can be stored in files and the diagnostic results can be obtained in no more than five reasoning steps. These inherent properties of this model make it an ideal model for online application.

In [89] and [131], fuzzy reasoning P systems with real numbers were presented. In [121], fuzzy reasoning spiking neural P systems with trapezoidal fuzzy numbers (tFRSN P systems) were used

for fault diagnosis of power systems. In this model, in order to develop the inference ability of tFRSN P systems from classical reasoning to fuzzy reasoning, a matrix-based fuzzy reasoning algorithm based on the dynamic firing mechanism was proposed. Moreover, the neurons in tFRSN P systems can be divided into four types, i.e., proposition neurons and three kinds of rule neurons: *general*, *AND,* and *OR*. Also the pulse value contained in each neuron is represented by a trapezoidal fuzzy number in $[0, 1]$ instead of a real number.

In [119], fuzzy reasoning spiking neural P systems with real numbers (rFRSN P systems) have been proposed for fault diagnosis of electric locomotive systems. Moreover, using fuzzy production rules, the relationships among breakdown signals and faulty sections in subsystems of electric locomotive systems have been investigated and, according to these rules. the fault diagnosis models are constructed for these subsystems. Then the fault diagnosis models based on rFRSN P systems for these subsystems are built according to these rules. Moreover, in [122], the fault diagnosis models for Shaoshan4 (SS4) electric locomotive systems based on rFRSN P systems were discussed. Also the rules neurons of this model are of three types, i.e., *GENERAL, AND,* and *OR*.

In 2015, a fault diagnosis method was introduced, based on fuzzy reasoning spiking neural P systems (FDSNP systems) [119] with trapezoidal fuzzy numbers. This model was used to model the faulty section along with an algebraic fuzzy reasoning algorithm which further helped the power system identify the faults. Moreover, from the diagnosis, it was clear that this model can effectively identify the faults in power transmission network with single and multiple fault section irrespective of incomplete and uncertain data from SCADA. Similarly, in [120], a weighted fuzzy reasoning spiking neural P system (WFSN P system) was proposed for diagnosis of the faults occurring in a traction power supply system of high-speed railways. A modified fuzzy reasoning spiking neural P system (MFRSN P system) [47] was also introduced in 2015 to solve the fault diagnosis problems in metro traction power systems. This model is also a rFRSN P system and it contains three rule neurons.

In [118], a new variant of spiking neural P systems known as fuzzy reasoning spiking neural P systems with interval-valued fuzzy numbers (ivFRSN P systems) was introduced for fault diagnosis of power systems. Such a model is a combination of interval-valued fuzzy numbers and spiking neural P systems. Furthermore, ivFRSN P systems can handle the incomplete and uncertain messages from the SCADA systems. In [88], intuitionistic fuzzy spiking neural P (IFSNP) systems were introduced by integrating intuitionistic fuzzy logic into spiking neural P systems. Such a model is very effective in the identification of fault in power systems which receives incomplete and uncertain messages from SCADA.

5.3 Other Applications

Membrane computing models are also useful in solving problems related to engineering optimization. It is well known that there exist a huge amount of computationally hard engineering problems which are intractable. Membrane algorithms are a popular heuristic approach to solve computationally hard problems because of its parallel distributive architecture, flexible evolution rules, and the like. Moreover, these approaches have been used to solve engineering problems in areas such as digital image processing, radar emitter signal analysis, and constrained manufacturing parameter optimization problems. The optimization of the time-frequency atom decomposition process of radar emitter signals has been done by quantum-inspired evolutionary algorithms (QIEA) in membrane algorithm framework and P systems (MQEPS) [136]. In [135], the image sparse decomposition problem has been solved by the membrane algorithm with quantum-inspired systems (MAQIS). The membrane algorithm based on tissue P systems and differential evolution (DETPS) [134] was applied to solve manufacturing parameter optimization problems. Moreover, an adaptive membrane evolutionary algorithm having dynamic membrane structure (AMEA) was introduced

in [130], and the differential evolution with the adaptive mutation factor has been used for solving CEOPs (constrained engineering optimization problems).

6 CONCLUSIONS AND FURTHER WORKS

In this article, we focus on a branch of natural computing called membrane computing, where membranes play an essential role in numerous biochemical responses which occur in compartments of cells. Some basic notions of such a paradigm were introduced, several classic types of P systems were given, and results both in theory and applications were presented.

Inspired by various biological facts, except the maximally parallelism of using rules, various parallel way of applying rules were proposed, such as only one rule is employed in a P system at every step (*sequential mode* [110, 138]); any number of applicable rules is employed in a region (*asynchronous mode* [40]); a appointed number of rules in each region or in the system (*bounded parallelism* [13]); if there exist some application rules in a membrane, then at least one rule must be employed (*minimal parallelism* [25]), and so on.

The applications to computer science are diverse, for instance, parallel architectures, modeling/simulating circuits, computer graphics, sorting/ranking, cryptography, and the like, which demonstrate that P systems are suitable for several research areas, actually, most of the applications are based on software for simulating P systems. There are also software products developed for didactic reasons, distributed implementations, as well as hardware attempts to implement P systems. Under development, there is also a specialized programming language (*P-Lingua*), as well as a plan to implement a P system in a biolab. In addition, perhaps a promising direction of applications is the use of membrane computing ideas in evolutionary computing.

REFERENCES

[1] Leonard M. Adleman. 1994. Molecular computation of solutions to combinatorial problems. *Science* 266, 5187 (1994), 1021–1024.

[2] Artiom Alhazov, Rudolf Freund, Alberto Leporati, Marion Oswald, and Claudio Zandron. 2006. (Tissue) P systems with unit rules and energy assigned to membranes. *Fundamenta Informaticae* 74, 4 (2006), 391–408.

[3] Artiom Alhazov, Rudolf Freund, and Marion Oswald. 2005. Symbol/membrane complexity of P systems with symport/antiport rules. In *Lecture Notes in Computer Science, Vol.* 3850 (2005), 96–113.

[4] Artiom Alhazov and Tseren-Onolt Ishdorj. 2004. Membrane operations in P systems with active membranes. In *Proceedings of the 2nd Brainstorming Week on Membrane Computing, Sevilla, ETS de Ingeniería Informática, 2-7 de Febrero, 2004 37-44.* (2004).

[5] Artiom Alhazov, Carlos Martín-Vide, and Linqiang Pan. 2003. Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundamenta Informaticae* 58 (11 2003), 67–77.

[6] Artiom Alhazov, Carlos Martín-Vide, and Linqiang Pan. 2004. *Solving Graph Problems by P Systems with Restricted Elementary Active Membranes.* Springer Berlin, Berlin, 1–22. DOI: https://doi.org/10.1007/978-3-540-24635-0_1

[7] Artiom Alhazov and Linqiang Pan. 2004. Polarizationless P systems with active membranes. *Grammars* 7, 1 (2004), 141–159.

[8] Artiom Alhazov, Linqiang Pan, and Gheorghe Păun. 2004. Trading polarizations for labels in P systems with active membranes. *Acta Informatica* 41, 2–3 (2004), 111–144.

[9] Artiom Alhazov and Mario J. Pérez-Jiménez. 2007. Uniform solution of QSAT using polarizationless active membranes. In *Machines, Computations, and Universality*, Jérôme Durand-Lose and Maurice Margenstern (Eds.). Springer Berlin, Berlin,, 122–133.

[10] Artiom Alhazov and Yurii Rogozhin. 2006. Towards a characterization of P systems with minimal symport/antiport and two membranes. In *Lecture Notes in Computer Science, vol.* 4361, Springer, (2006), 135–153.

[11] Francesco Bernardini and Marian Gheorghe. 2005. Cell communication in tissue P systems: Universality results. *Soft Computing* 9, 9 (2005), 640–649.

[12] Francesco Bernardini and Andrei Păun. 2003. Universality of minimal symport/antiport: Five membranes suffice. In *Lecture Notes in Computer Science,* vol. 2933. Springer, (2003), 43–54.

[13] Francesco Bernardini, Francisco J. Romero-Campero, Marian Gheorghe, and Mario J. Pérez-Jiménez. 2006. A modeling approach based on P systems with bounded parallelism. In *Lecture Notes in Computer Science,* vol. 4361. Springer, (2006), 49–65.

[14] Cătălin Buiu and Andrei George Florea. 2019. Membrane computing models and robot controller design , current results and challenges. *Journal of Membrane Computing* 1, 4 (2019), 262–269.

[15] Catalin Buiu, Cristian Vasile, and Octavian Arsene. 2012. Development of membrane controllers for mobile robots. *Information Sciences* 187 (2012), 33–51. Issue 1.

[16] Francis George C. Cabarle, Henry N. Adorna, Min Jiang, and Xiangxiang Zeng. 2017. Spiking neural P systems with scheduled synapses. *IEEE Transactions on Nanobioscience* 16, 8 (2017), 792–801.

[17] Francis George C. Cabarle, Henry N. Adorna, Mario J. Pérez-Jiménez, and Tao Song. 2015. Spiking neural P systems with structural plasticity. *Neural Computing and Applications* 26, 8 (2015), 1905–1917.

[18] Leandro Nunes Castro, Leandro Nunes De Castro, and Jonathan Timmis. 2002. *Artificial Immune Systems: A New Computational Intelligence Approach.* Springer Science & Business Media.

[19] Matteo Cavaliere. 2002. Evolution–communication P systems. In *Lecture Notes in Computer Science, vol.* 2597. Springer (2002), 134–145.

[20] Matteo Cavaliere, Oscar H. Ibarra, Gheorghe Păun, Omer Egecioglu, Mihai Ionescu, and Sara Woodworth. 2009. Asynchronous spiking neural P systems. *Theoretical Computer Science* 410, 24–25 (2009), 2352–2364.

[21] Rodica Ceterchi, Radu Gramatovici, Nataša Jonoska, and K. G. Subramanian. 2003. Tissue-like P systems with active membranes for picture generation. *Fundamenta Informaticae* 56, 4 (2003), 311–328.

[22] Haiming Chen, Rudolf Freund, Mihai Ionescu, Gheorghe Păun, and Mario J. Pérez-Jiménez. 2007. On string languages generated by spiking neural P systems. *Fundamenta Informaticae* 75, 1–4 (2007), 141–162.

[23] Haiming Chen, Mihai Ionescu, Tseren-Onolt Ishdorj, Andrei Păun, Gheorghe Păun, and Mario J. Pérez-Jiménez. 2008. Spiking neural P systems with extended rules: Universality and languages. *Natural Computing* 7, 2 (2008), 147–166.

[24] A. Hepzibah Christinal, Daniel Díaz-Pernil, and T. Mathu. 2018. A uniform family of tissue P systems with protein on cells solving 3-coloring in linear time. *Natural Computing* 17, 2 (2018), 311–319.

[25] Gabriel Ciobanu, Linqiang Pan, Gheorghe Păun, and Mario J. Pérez-Jiménez. 2007. P systems with minimal parallelism. *Theoretical Computer Science* 378, 1 (2007), 117–130.

[26] Erzsébet Csuhaj-Varjú and Sergey Verlan. 2011. On generalized communicating P systems with minimal interaction rules. *Theoretical Computer Science* 412, 1–2 (2011), 124–135.

[27] Erzsébet Csuhaj-Varjú and Sergey Verlan. 2017. Computationally complete generalized communicating P systems with three cells. In *Lecture Notes in Computer Science,* vol. 10725. Springer (2017), 118–128.

[28] Daniel Díaz-Pernil, Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez. 2008. A uniform family of tissue P systems with cell division solving 3-COL in a linear time. *Theoretical Computer Science* 404, 1–2 (2008), 76–87.

[29] Daniel Díaz-Pernil, Mario J. Pérez Jiménez, Agustín Riscos Núñez, and Álvaro Romero-Jiménez. 2008. Computational efficiency of cellular division in tissue-like membrane systems. *Romanian Journal of Information Science and Technology* 11, 3 (2008), 229–241.

[30] Daniel Díaz-Pernil, Mario J. Pérez-Jiménez, and Álvaro Romero-Jiménez. 2009. Efficient simulation of tissue-like P systems by transition cell-like P systems. *Natural Computing* 8, 4 (2009), 797–806.

[31] Andrei George Florea and Cătălin Buiu. 2017. *Membrane Computing for Distributed Control of Robotic Swarms: Emerging Research and Opportunities.* Pennsylvania: IGI Global.

[32] Andrei George Florea and Cătălin Buiu. 2017. Modelling multi-robot interactions using a generic controller based on numerical P systems and ROS. In *IEEE Proceedings of 2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI).* IEEE Press, 1–6.

[33] Andrei George Florea and Cătălin Buiu. 2018. A distributed approach to the control of multi-robot systems using XP colonies. *Integrated Computer-Aided Engineering* 25 (2018), 15–29. Issue 1.

[34] David B. Fogel. 2006. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* Vol. 1. John Wiley & Sons.

[35] Rudolf Freund, Lila Kari, Marion Oswald, and Petr Sosík. 2005. Computationally universal P systems without priorities: Two catalysts are sufficient. *Theoretical Computer Science* 330, 2 (2005), 251–266.

[36] Rudolf Freund and Marion Oswald. 2002. P systems with activated/prohibited membrane channels. In *Lecture Notes in Computer Science,* vol. 2597 (2002). Springer, 261–269.

[37] Rudolf Freund and Andrei Păun. 2002. Membrane systems with symport/antiport rules: Universality results. In *Lecture Notes in Computer Science,* vol. 2597 (2002). Springer, 270–287.

[38] Rudolf Freund and Andrei Păun. 2005. P systems with active membranes and without polarizations. *Soft Computing* 9, 9 (2005), 657–663.

[39] Rudolf Freund, Gheorghe Păun, and Mario J. Pérez-Jiménez. 2005. Tissue P systems with channel states. *Theoretical Computer Science* 330, 1 (2005), 101–116.
[40] Pierluigi Frisco, Gordon Govan, and Alberto Leporati. 2012. Asynchronous P systems with active membranes. *Theoretical Computer Science* 429 (2012), 74–86.
[41] Pierluigi Frisco and Hendrik Jan Hoogeboom. 2002. Simulating counter automata by P systems with symport/antiport. In *Lecture Notes in Computer Science,* vol. 2597 (2002). Springer, 288–301.
[42] Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company.
[43] Rosa Gutiérrez-Escudero, Mario J. Pérez-Jiménez, and Miquel Rius-Font. 2010. Characterizing tractability by tissue-like P systems. In *Lecture Notes in Computer Science,* vol. 5957 (2010). Springer, 289–300.
[44] Miguel Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez. 2005. A fast P system for finding a balanced 2-partition. *Soft Computing* 9 (2005), 673–678.
[45] Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, Agustín Riscos-Núñez, and Francisco J. Romero-Campero. 2006. On the power of dissolution in P systems with active membranes. In *Lecture Notes in Computer Science,* vol. 3850 (2006). Springer, 224–240.
[46] Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, Agustín Riscos Núñez, and Francisco J. Romero-Campero. 2007. Characterizing tractability by cell-like membrane systems (Formal models, languages and applications), M. Mukund, K. G. Subramanian, and K. Rangarajan (Eds.). World Scientific, 137–154. DOI: https://doi.org/10.1142/9789812773036_0009
[47] Yangyang He, Tao Wang, Kang Huang, G. Zhang, and Mario J. Pérez-Jiménez. 2015. 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 (2015), 256–272.
[48] Oscar H. Ibarra and Hsu-Chun Yen. 2006. Deterministic catalytic systems are not universal. *Theoretical Computer Science* 363, 2 (2006), 149–161.
[49] Mihai Ionescu, Gheorghe Păun, and Takashi Yokomori. 2006. Spiking neural P systems. *Fundamenta Informaticae* 71, 2, 3 (2006), 279–308.
[50] Tseren-Onolt Ishdorj and Mihai Ionescu. 2004. Replicative–distribution rules in P systems with active membranes. In *Lecture Notes in Computer Science*, vol. 3407 (2004). Springer, 68–83.
[51] Richelle Ann B. Juayong and Henry N. Adorna. 2015. Relating computations in non-cooperative transition P systems and evolution-communication P systems with energy. *Fundamenta Informaticae* 136, 3 (2015), 209–217.
[52] Richelle Ann B. Juayong and Henry N. Adorna. 2018. On simulating cooperative transition P systems in evolution–communication P systems with energy. *Natural Computing* 17, 2 (2018), 333–343.
[53] Jozef Kelemen, Alica Kelemenová, and Gheorghe Păun. 2004. Preview of P colonies: A biochemically inspired computing model. In *Workshop and Tutorial Proceedings of the 9th International Conference on the Simulation and Synthesis of Living Systems (Alife IX).* Boston, Massachusetts, 82–86.
[54] James Kennedy. 2006. Swarm intelligence. In *Handbook of Nature-Inspired and Innovative Computing*. Springer, 187–219.
[55] Shankara Narayanan Krishna. 2007. On the computational power of flip-flop proteins on membranes. In *Lecture Notes in Computer Science, vol.* 4497 (2007), 695–704.
[56] Shankara Narayanan Krishna, Marian Gheorghe, Florentin Ipate, Erzsébet Csuhaj-Varjú, and Rodica Ceterchi. 2017. Further results on generalised communicating P systems. *Theoretical Computer Science* 701 (2017), 146–160.
[57] Shankara Narayanan Krishna and Andrei Păun. 2004. Results on catalytic and evolution-communication P systems. *New Generation Computing* 22, 4 (2004), 377–394.
[58] Shankara Narayanan Krishna and Rama Raghavan. 1999. A variant of P systems with active membranes: Solving NP-complete problems. *Romanian Journal of Information Science and Technology* 2, 4 (1999), 305–394.
[59] Christopher G. Langton. 1997. *Artificial Life: An Overview.* MIT Press.
[60] Liucheng Liu, Wenmei Yi, Qian Yang, Hong Peng, and Jun Wang. 2019. Numerical P systems with Boolean condition. *Theoretical Computer Science* 785 (2019), 140–149.
[61] Carlos Martín-Vide, Gheorghe Păun, Juan Pazos, and Alfonso Rodríguez-Patón. 2003. Tissue P systems. *Theoretical Computer Science* 296, 2 (2003), 295–326.
[62] Madhu Mutyam and Kamala Krithivasan. 2001. P systems with membrane creation: Universality and efficiency. In *Lecture Notes in Computer Science,* vol. 2055 (2001), Springer, 276–287.
[63] Murphy Niall and Woods Damien. 2014. Uniformity is weaker than semi-uniformity for some membrane systems. *Fundamenta Informaticae* 134, 1–2 (2014), 129–152.
[64] Michael A. Nielsen and Isaac Chuang. 2010. *Quantum Computation and Quantum Information.* Cambridge University Press.

[65] Linqiang Pan, Gheorghe Păun, and Bosheng Song. 2016. Flat maximal parallelism in P systems with promoters. *Theoretical Computer Science* 623 (2016), 83–91.
[66] Linqiang Pan, Gheorghe Păun, Gexiang Zhang, and Ferrante Neri. 2017. Spiking neural P systems with communication on request. *International Journal of Neural Systems* 27, 8 (2017), 1–13.
[67] Linqiang Pan and Mario J. Pérez-Jiménez. 2010. Computational complexity of tissue-like P systems. *Journal of Complexity* 26, 3 (2010), 296–315.
[68] Linqiang Pan, Bosheng Song, Luis Valencia-Cabrera, and Mario J. Pérez-Jiménez. 2018. The computational complexity of tissue P systems with evolutional symport/antiport rules. *Complexity* 2018 (2018).
[69] Linqiang Pan, Jun Wang, and Hendrik Jan Hoogeboom. 2012. Spiking neural P systems with astrocytes. *Neural Computation* 24, 3 (2012), 805–825.
[70] Linqiang Pan, Yanfeng Wang, Suxia Jiang, and Bosheng Song. 2017. Flat maximal parallelism in tissue P systems with promoters. *Romanian Journal of Information Science and Technology* 20, 1 (2017), 42–56.
[71] Linqiang Pan, Zhiqiang Zhang, Tingfang Wu, and Jinbang Xu. 2017. Numerical P systems with production thresholds. *Theoretical Computer Science* 673 (2017), 30–41.
[72] Andrei Păun and Gheorghe Păun. 2002. The power of communication: P systems with symport/antiport. *New Generation Comput.* 20, 3 (2002), 295–306.
[73] Andrei Păun, Gheorghe Păun, and Grzegorz Rozenberg. 2002. Computing by communication in networks of membranes. *International Journal of Foundations of Computer Science* 13, 6 (2002), 779–798.
[74] Andrei Păun, Mihaela Păun, Alfonso Rodríguez-Patón, and Manuela Sidoroff. 2011. P systems with proteins on membranes: A survey. *International Journal of Foundations of Computer Science* 22, 1 (2011), 39–53.
[75] Andrei Păun and Bianca Popa. 2006. P systems with proteins on membranes. *Fundamenta Informaticae* 72, 4 (2006), 467–483.
[76] Andrei Păun and Bianca Popa. 2006. P systems with proteins on membranes and membrane division. In *Lecture Notes in Computer Science,* vol. 4036 (2006). Springer, 292–303.
[77] Andrei Păun and Alfonso Rodríguez-Patón. 2007. On flip-flop membrane systems with proteins. In *Lecture Notes in Computer Science,* vol. 4860 (2007). Springer, 414–427.
[78] Gheorghe Păun. 2000. Computing with membranes. *J. Comput. System Sci.* 61, 1 (2000), 108–143.
[79] Gheorghe Păun. 2001. P systems with active membranes: Attacking NP complete problems. *Journal of Automata, Languages and Combinatorics* 6, 1 (2001), 75–90.
[80] Gheorghe Păun. 2002. *Membrane Computing. An Introduction.* Springer-Verlag.
[81] Gheorghe Păun and Radu Păun. 2006. Membrane computing and economics: Numerical P systems. *Fundamenta Informaticae* 73, 1, 2 (2006), 213–227.
[82] Gheorghe Păun, Mario J. Pérez-Jiménez, and Agustín Riscos-Núñez. 2008. Tissue P systems with cell division. *International Journal of Computers Communications & Control* 3, 3 (2008), 295–303.
[83] Gheorghe Păun, Mario J. Pérez-Jiménez, and Grzegorz Rozenberg. 2006. Spike trains in spiking neural P systems. *International Journal of Foundations of Computer Science* 17, 4 (2006), 975–1002.
[84] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. 2010. *The Oxford Handbook of Membrane Computing.* Oxford University Press.
[85] Ana Pavel, Octavian Arsene, and Catalin Buiu. 2010. Enzymatic numerical P systems-a new class of membrane computing systems. In *2010 IEEE 5th International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010).* IEEE, Liverpool, United Kingdom, 1331–1336.
[86] Hong Peng, Tingting Bao, Xiaohui Luo, Jun Wang, Xiaoxiao Song, Agustín Riscos-Núñez, and Mario J. Pérez-Jiménez. 2020. Dendrite P systems. *Neural Networks* 127 (2020), 110–120. DOI: https://doi.org/10.1016/j.neunet.2020.04.014
[87] Hong Peng and Jun Wang. 2019. Coupled neural P systems. *IEEE Transactions on Neural Networks and Learning Systems* 30, 6 (2019), 1672–1682.
[88] Hong Peng, Jun Wang, Jun Ming, Peng Shi, Mario J. Pérez-Jiménez, Wenping Yu, and Chengyu Tao. 2017. Fault diagnosis of power systems using intuitionistic fuzzy spiking neural P systems. *IEEE Transactions on Smart Grid* 9, 5 (2017), 4777–4784.
[89] Hong Peng, Jun Wang, Mario J. Pérez-Jiménez, Hao Wang, Jie Shao, and Tao Wang. 2013. Fuzzy reasoning spiking neural P system for fault diagnosis. *Information Sciences* 235 (2013), 106–116.
[90] Ignacio Pérez-Hurtado, Miguel Á Martinez-del Amor, Zhang Gexiang, Ferrante Neri, and Mario J. Pérez-Jiménez. 2020. A membrane parallel rapidly-exploring random tree algorithm for robotic motion planning. *Integrated Computer-Aided Engineering* 27 (2020), 121–138. Issue 2.
[91] Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, Gexiang Zhang, and David Orellana-Martín. 2018. Simulation of rapidly-exploring random trees in membrane computing with P-Lingua and automatic programming. *International Journal of Computers, Communications and Control* 13 (2018), 1007–1031. Issue 6.

[92] Mario J. Pérez-Jiménez and Agustín Riscos-Núñez. 2005. Solving the Subset-Sum problem by active membranes. *New Generation Computing* 23, 4 (2005), 367–384.
[93] Mario J. Pérez-Jiménez and Agustín Riscos-Núñez. 2004. A linear-time solution to the knapsack problem using P systems with active membranes. In *Lecture Notes in Computer Science*, vol. 2933 (2004). Springer, 250–268.
[94] Mario J. Pérez-Jiménez, Agustín Riscos-Núñez, M Rius-Font, and Francisco J. Romero-Campero. 2013. A polynomial alternative to unbounded environment for tissue P systems with cell division. *International Journal of Computer Mathematics* 90, 4 (2013), 760–775.
[95] Mario J. Pérez-Jiménez and Francisco J. Romero-Campero. 2004. An efficient family of P systems for packing items into bins. *Journal of Universal Computer Science* 10, 5 (2004), 650–670.
[96] Mario J. Pérez-Jiménez and Francisco J. Romero-Campero. 2005. Attacking the common algorithmic problem by recognizer P systems. In *Lecture Notes in Computer Science,* vol. 3354 (2005). Springer, 304–315.
[97] Mario J. Pérez-Jiménez, Álvaro Romero-Jiménez, and Fernando Sancho-Caparrini. 2003. Complexity classes in cellular computing with membranes. *Natural Computing* 2, 3 (2003), 265–285.
[98] Mario J. Pérez-Jiménez, Álvaro Romero-Jiménez, and Fernando Sancho-Caparrini. 2003. Decision P systems and the **P≠NP** conjecture. In *Lecture Notes in Computer Science,* vol. 2597 (2003). Springer, 388–399.
[99] Mario J. Pérez-Jiménez, Álvaro Romero-Jiménez, and Fernando Sancho-Caparrini. 2006. A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics* 11, 4 (2006), 423–434.
[100] Antonio E. Porreca, Giancarlo Mauri, and Claudio Zandron. 2006. Complexity classes for membrane systems. *Informatique théorique et applications* 40, 2 (2006), 141–162.
[101] Grzegorz Rozenberg and Salomaa Arto. 1997. *Handbook of Formal Languages.* Springer-Verlag.
[102] Bosheng Song and Linqiang Pan. 2015. Computational efficiency and universality of timed P systems with active membranes. *Theoretical Computer Science* 567 (2015), 74–86.
[103] Bosheng Song and Linqiang Pan. 2016. The computational power of tissue-like P systems with promoters. *Theoretical Computer Science* 641 (2016), 43–52.
[104] Bosheng Song, Linqiang Pan, and Mario J. Pérez-Jiménez. 2016. Tissue P systems with protein on cells. *Fundamenta Informaticae* 144, 1 (2016), 77–107.
[105] Bosheng Song, Mario J. Pérez-Jiménez, Gheorghe Păun, and Linqiang Pan. 2016. Tissue P systems with channel states working in the flat maximally parallel way. *IEEE Transactions on Nanobioscience* 15, 7 (2016), 645–656.
[106] Bosheng Song, Xiangxiang Zeng, Min Jiang, and Mario J. Pérez-Jiménez. 2021. Monodirectional tissue P systems with promoters. *IEEE Transactions on Cybernetics* 51, 1 (2021), 438–450. DOI: https://doi.org/10.1109/TCYB.2020.3003060
[107] Bosheng Song, Xiangxiang Zeng, and Alfonso Rodríguez-Patón. 2021. Monodirectional tissue P systems with channel states. *Information Sciences* 546 (2021), 206–219.
[108] Bosheng Song, Cheng Zhang, and Linqiang Pan. 2017. Tissue-like P systems with evolutional symport/antiport rules. *Information Sciences* 378 (2017), 177–193.
[109] Tao Song, Luis F. Macías-Ramos, Linqiang Pan, and Mario J. Pérez-Jiménez. 2014. Time-free solution to SAT problem using P systems with active membranes. *Theoretical Computer Science* 529 (2014), 61–68.
[110] Tao Song, Linqiang Pan, Keqin Jiang, Bosheng Song, and Wei Chen. 2013. Normal forms for some classes of sequential spiking neural P systems. *IEEE Transactions on Nanobioscience* 12, 3 (2013), 255–264.
[111] Petr SosÍk. 2003. The power of catalysts and priorities in membrane systems. *Grammars* 6, 1 (2003), 13–24.
[112] Petr Sosík, Andrei Păun, and Alfonso Rodríguez-Patón. 2013. P systems with proteins on membranes characterize PSPACE. *Theoretical Computer Science* 488 (2013), 78–95.
[113] Petr Sosík and Alfonso Rodríguez-Patón. 2006. P systems with active membranes characterize PSPACE. In *Lecture Notes in Computer Science,* vol. 4287 (2006), 33–46.
[114] Cristian Ioan Vasile, Ana Brânduşa Pavel, Ioan Dumitrache, and Gheorghe Păun. 2012. On the power of enzymatic numerical P systems. *Acta Informatica* 49, 6 (2012). Springer, 395–412.
[115] Sergey Verlan, Francesco Bernardini, Marian Gheorghe, and Maurice Margenstern. 2006. Computational completeness of tissue P systems with conditional uniport. In *Lecture Notes in Computer Science,* vol. 4361 (2006). Springer, 521–535.
[116] Sergey Verlan, Francesco Bernardini, Marian Gheorghe, and Maurice Margenstern. 2008. Generalized communicating P systems. *Theoretical Computer Science* 404, 1–2 (2008), 170–184.
[117] Jun Wang, Hendrik Jan Hoogeboom, Linqiang Pan, Gheorghe Păun, and Mario J. Pérez-Jiménez. 2010. Spiking neural P systems with weights. *Neural Computation* 22, 10 (2010), 2615–2646.
[118] Jun Wang, Hong Peng, Wenping Yu, Jun Ming, Mario J. Pérez-Jiménez, Chengyu Tao, and Xiangnian Huang. 2019. Interval-valued fuzzy spiking neural P systems for fault diagnosis of power transmission networks. *Engineering Applications of Artificial Intelligence* 82 (2019), 102–109.
[119] Tao Wang, Gexiang Zhang, and Mario J. Pérez-Jiménez. 2014. Fault diagnosis models for electric locomotive systems based on fuzzy reasoning spiking neural P systems. In *Lecture Notes in Computer Science,* vol. 8961 (2014). Springer, 385–395.

[120] Tao Wang, Gexiang Zhang, Mario J. Pérez-Jiménez, and Jixiang Cheng. 2015. 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 (2015), 1103–1114.

[121] Tao Wang, Gexiang Zhang, Haina Rong, and Mario J. Pérez-Jiménez. 2014. Application of fuzzy reasoning spiking neural P systems to fault diagnosis. *International Journal of Computers Communications & Control* 9, 6 (2014), 786–799.

[122] Tao Wang, Gexiang Zhang, Junbo Zhao, Zhengyou He, Jun Wang, and Mario J. Pérez-Jiménez. 2014. Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. *IEEE Transactions on Power Systems* 30, 3 (2014), 1182–1194.

[123] Xun Wang, Tao Song, Faming Gong, and Pan Zheng. 2016. On the computational power of spiking neural P systems with self-organization. *Scientific Reports* 6 (2016), 27624.

[124] Xueyuan Wang, Gexiang Zhang, Xiantai Gou, Prithwineel Paul, Ferrante Neri, Haina Rong, Qiang Yang, and Hua Zhang. 2020. Multi-behaviors coordination controller design with enzymatic numerical P systems for robots. *Integrated Computer-Aided Engineering* (2020), 1–22. DOI: https://doi.org/10.3233/ICA-200627

[125] Xueyuan Wang, Gexiang Zhang, Ferrante Neri, Tao Jiang, Junbo Zhao, Marian Gheorghe, Florentin Ipate, and Raluca Lefticaru. 2016. Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots. *Integrated Computer-Aided Engineering* 23 (2016), 15–30. Issue 1.

[126] Xueyuan Wang, Gexiang Zhang, Junbo Zhao, Haina Rong, Florentin Ipate, and Raluca Lefticaru. 2015. A modified membrane-inspired algorithm based on particle swarm optimization for mobile robot path planning. *International Journal of Computers, Communications and Control* 10 (2015), 732–745. Issue 5.

[127] Stephen Wolfram. 2002. *A New Kind of Science*. Vol. 5. Wolfram media, Champaign, IL.

[128] Tingfang Wu, Andrei Păun, Zhiqiang Zhang, and Linqiang Pan. 2018. Spiking neural P systems with polarizations. *IEEE Transactions on Neural Networks and Learning Systems* 29, 8 (2018), 3349–3360.

[129] Tingfang Wu, Zhiqiang Zhang, Gheorghe Păun, and Linqiang Pan. 2016. Cell-like spiking neural P systems. *Theoretical Computer Science* 623 (2016), 180–189.

[130] Jianhua Xiao, Ying Liu, Shuai Zhang, and Ping Chen. 2017. An adaptive membrane evolutionary algorithm for solving constrained engineering optimization problems. *Journal of Universal Computer Science* 23 (2017), 652–672.

[131] Guojiang Xiong, Dongyuan Shi, Lin Zhu, and Xianzhong Duan. 2013. A new approach to fault diagnosis of power systems using fuzzy reasoning spiking neural P systems. *Mathematical Problems in Engineering* 2013 (2013).

[132] Claudio Zandron, Claudio Ferretti, and Giancarlo Mauri. 2000. Solving NP-complete problems using P systems with active membranes. *Unconventional Models of Computation* (2000), 289–301.

[133] Claudio Zandron, Alberto Leporati, Claudio Ferretti, Giancarlo Mauri, and Mario J Pérez-Jiménez. 2008. On the computational efficiency of polarizationless recognizer P systems with strong division and dissolution. *Fundamenta Informaticae* 87, 1 (2008), 79–91.

[134] Gexiang Zhang, Jixiang Cheng, Marian Gheorghe, and Qi Meng. 2013. A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems. *Applied Soft Computing* 13 (2013), 1528–1542.

[135] Gexiang Zhang, Marian Gheorghe, and Yuquan Li. 2012. A membrane algorithm with quantum-inspired subalgorithms and its application to image processing. *Natural Computing* 11 (2012), 701–717.

[136] Gexiang Zhang, Chunxiu Liu, and Haina Rong. 2010. Analyzing radar emitter signals with membrane algorithms. *Mathematical and Computer Modelling* 52 (2010), 1997–2010.

[137] Gexiang Zhang, Mario Pérez-Jiménez, and Marian Gheorghe. 2017. *Real-life Applications with Membrane Computing*. Vol. 25. Springer. DOI: https://doi.org/10.1007/978-3-319-55989-6

[138] Xingyi Zhang, Bin Luo, Xianyong Fang, and Linqiang Pan. 2012. Sequential spiking neural P systems with exhaustive use of rules. *BioSystems* 108, 1–3 (2012), 52–62.

[139] Xingyi Zhang, Linqiang Pan, and Andrei Păun. 2015. On the universality of axon P systems. *IEEE Transactions on Neural Networks and Learning Systems* 26, 11 (2015), 2816–2829.

[140] Zhiqiang Zhang and Linqiang Pan. 2016. Numerical P systems with thresholds. *International Journal of Computers Communications & Control* 11, 2 (2016), 292–304.

[141] Zhiqiang Zhang, Tingfang Wu, Andrei Păun, and Linqiang Pan. 2016. Numerical P systems with migrating variables. *Theoretical Computer Science* 641 (2016), 85–108.