# Automatic Design of Deterministic and Non-Halting Membrane Systems by Tuning Syntactical Ingredients

Gexiang Zhang*, *Member, IEEE*, Haina Rong, Zhu Ou, Mario J. Pérez-Jiménez, and Marian Gheorghe

***Abstract*—To solve the programmability issue of membrane computing models, the automatic design of membrane systems is a newly initiated and promising research direction. In this paper, we propose an automatic design method, *Permutation Penalty Genetic Algorithm* (PPGA), for a deterministic and non-halting membrane system by tuning membrane structures, initial objects and evolution rules. The main ideas of PPGA are the introduction of the permutation encoding technique for a membrane system, a penalty function evaluation approach for a candidate membrane system and a genetic algorithm for evolving a population of membrane systems toward a successful one fulfilling a given computational task. Experimental results show that PPGA can successfully accomplish the automatic design of a cell-like membrane system for computing the square of $n$ ($n \geq 1$ is a natural number) and can find the minimal membrane systems with respect to their membrane structures, alphabet, initial objects, and evolution rules for fulfilling the given task. We also provide the guidelines on how to set the parameters of PPGA.**

***Index Terms*—Automatic design, cell-like membrane systems, genetic algorithm, membrane computing, penalty function evaluation approach, permutation encoding technique.**

## I. Introduction

MEMBRANE computing, initiated by Păun in 1998 [1], aims to investigate the models, called membrane systems or P systems, abstracted from the structure and the functioning of the living cell as well as from the cooperation of cells in tissues, organs, and other populations of cells. Since it was reported in 2003 by Thompson Institute for Scientific Information, ISI, that the seminal paper was a fast breaking one and this area was an emerging research front in computer science, membrane computing has become a branch of natural computing and has developed very fast into a vigorous scientific discipline [2].

This work was supported by the National Natural Science Foundation of China (61170016, 61373047, 61134002), the Program for New Century Excellent Talents in University (NCET-11-0715) and SWJTU supported project (SWJTU12CX008). *Asterisk indicates corresponding author.*

*G. Zhang is with the School of Electrical Engineering, Southwest Jiaotong University, Chengdu, 610031 China (e-mail: zhgxdylan@126.com).

H. Rong and Z. Ou are with the School of Electrical Engineering, Southwest Jiaotong University, Chengdu, 610031 China (e-mail: ronghaina@126.com; 994345836@qq.com).

M. J. Pérez-Jiménez is with the Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda. Reina Mercedes s/n, 41012, Spain (e-mail: marper@us.es).

M. Gheorghe is with the Department of Computer Science, University of Sheffield, Regent Court, Portobello Street, Sheffield S1 4DP, UK (e-mail: m.gheorghe@sheffield.ac.uk).

The P systems in the literature can be structurally grouped into two types: cell-like and tissue-like P systems. The former contains one membrane cell with three main ingredients: a hierarchical membrane structure represented by a tree, multisets of objects and evolution rules. The latter, such as tissue, neural, and numerical P systems [2], [3], has a general graph membrane structure. Various variants of P systems can provide an exponential space for solving NP-hard problems in a linear time [4] and have a wide range of applications, such as the construction of optimization approaches [5], arithmetic operations [6] and controller design for mobile robots [7]. Until now, a P system for fulfilling a specific task, especially for solving an NP-hard, NP-complete or PSPACE problem or for controlling robots, is carefully designed by experts and cannot be automatically obtained by using programs, which extremely limits the application of P systems. How to automatically design a P system by using programs, namely, the programmability of a P system, is a new and ongoing research direction in the area of membrane computing.

The automatic design of a P system is a very complicated and challenging task. Now, a feasible way is to use evolutionary algorithms to evolve a population of P systems toward a successful one. This work started with the selection of an appropriate subset from a redundant set of evolution rules to design a cell-like P system, where a membrane structure and initial objects were pre-defined and fixed in the process of design [8]–[12]. In [8], a genetic algorithm (GA) was used to design the P system for calculating $4^2$. In this design, no encoding technique, such as binary and numeric, was used to represent a P system. The one-point crossover was performed on two evolution rule sets, that is, partial rules of the two evolution rule sets were exchanged. The uniform mutation was applied to change a random object in a randomly chosen evolution rule. In [9], a binary encoding technique was introduced to represent an evolution rule set of a P system and a quantum-inspired evolutionary algorithm (QIEA) was applied to evolve a population of P systems. This method successfully solved the design of P systems for computing $4^2$ and $n^2$ ($n \geq 2$ is a natural number). In [10], an evaluation method considering non-determinism and halting penalty factors and a GA with a binary encoding technique was presented to design P systems for $4^2$, $n^2$ and the generation of the language $a^{2^n}b^{3^n}$ ($n > 1$). In [8]–[10], a specific redundant evolution rule set was designed for a specific computational task. This case was developed in [11], [12] by using one pre-defined redundant evolution rule set to design multiple different P systems, each of which performs a computation task. In [11], an automatic design method of a cell-like P system framework for performing five basic

arithmetic operations (addition, subtraction, multiplication, division, and power) was discussed. In [12], a common redundant set of evolution rules was applied to design successful P systems for fulfilling eight computational tasks: $2(n-1)$, $2n-1$, $n^2$, $(1/2)[(n-1)^2+(n-1)]$, $(n-1)^2+(n-1)$, $(n-1)^2+2^n+2$, $a^{2^n}b^{3^n}$ and $(1/2)(3^n-1)$, ($n>1$ or 2). A significant step in this direction is the study in [13] that a cell-like halting P system for $4^2$ was designed by tuning membrane structures, initial objects, and evolution rules. In this method, a GA with a binary encoding technique was introduced to codify the membrane structure, initial objects, and evolution rules of a P system.

To advance the automatic design of a P system, this paper proposes an automatic design method, *Permutation Penalty Genetic Algorithm* (PPGA), for a deterministic and non-halting P system by tuning membrane structures, initial objects and evolution rules. We discuss the parameter setting of PPGA and conduct extensive experiments to verify the PPGA feasibility and effectiveness. The original work in this study is summarized as follows:

1) An automatic design method for a deterministic and non-halting membrane system by tuning membrane structures, initial objects, and evolution rules is presented for the first time.
2) In PPGA, the permutation encoding technique for representing a cell-like P system is introduced. This technique can overcome the drawback of the binary encoding method in [13] that a certain number of copies of the empty set $\lambda$ are inserted into the binary strings of a membrane structure, initial object set and evolution rule set, which results in the use of the objects in $V$ by unequal probabilities and the production of many infeasible P systems in the process of crossover and mutation.
3) A penalty function evaluation approach of a candidate P system is discussed by considering the feasibility of a P system due to its membrane structure and dissolution rules, the redundancy of objects and evolution rules, non-determinism, and halting feature.
4) The first attempt to consider the rewriting-communication rule in the automatic design methods is made.
5) How to design the minimal P system with respect to its membrane structure, alphabet, initial objects, and evolution rules is discussed.

The rest of this paper is organized as follows. Section II describes the problem to solve. Section III presents the design method PPGA. Experiments and results are provided in Section IV. Finally, some conclusions are drawn in Section V.

## II. Problem Description

This study considers the automatic design of a P system for successfully generating the set $\{n^2|n\geq 1\}$ of natural numbers, where $n$ is a natural number greater than or equal to 1. The system must have the following characteristics:

1) The system is a cell-like P system with a hierarchical membrane structure and can be formally represented as

$$\Pi=(V,O,\mu,W,\Re,i_o)$$

where

a) $V$ is the (finite and non-empty) alphabet of objects. From the perspective of P systems, the alphabet $V$ is usually fixed by the user and the objects in $V$ usually have implicit meanings, for instance, they may be thought as proteins, predators, or clauses, according to the purpose of the P system.
b) $O\subseteq V$ is the output alphabet, namely, the set of output objects.
c) $\mu$ is a hierarchical membrane structure with $m\geq 1$ membranes labeled by the elements of a given set $H$, $H=\{0,1,\ldots,m-1\}$, and the skin membrane is labeled as 0. The hierarchical membrane structure can also be depicted through a rooted tree.
d) $W$ is the vector of initial multisets $w_0,\ldots,w_{m-1}$ over $V$ associated with the regions $0,1,\ldots,m-1$ delimited by the membranes of $\mu$, namely, $W=[w_0,\ldots,w_{m-1}]$.
e) $\Re$ is the set of finite sets $R_0,\ldots,R_{m-1}$ of evolution rules associated with the regions $0,1,\ldots,m-1$ of the membrane structure $\mu$, namely, $\Re=\{R_0,\ldots,R_{m-1}\}$. Three types of evolution rules, rewriting, dissolution and rewriting-communication rules, are considered in this study, that is, $R_i$ ($i=0,1,\ldots,m-1$) has rules of one of the following forms:
   i) *rewriting rule*: $[u\rightarrow v]_i$;
   ii) *dissolution rule*: $[u]_i\rightarrow v$;
   iii) *rewriting-communication rule*: $[u]_i\rightarrow [v]_i x$;

   where $i\in H$; $u\in V$; $v,x\in V^*$; where $V^*$ denotes the set of all strings over $V$. The left hand side of these rules is $u$ and the right hand side of them is $v$ or $v,x$. The length of $u$ is called the radius of each rule. Our design considers the case that the radius of the three rules equals 1. The rewriting rule $[u\rightarrow v]_i$ rewrites $u$ by $v$. The dissolution rule $[u]_i\rightarrow v$ dissolves the compartment $i$ and its content is transferred to the surrounding membrane after all the other rules have been applied and $u$ is replaced by $v$. The rewriting-communication rule $[u]_i\rightarrow [v]_i x$ rewrites $u$ by $v$ inside the compartment $i$ and, in the same time, sends $x$ outside the compartment.
f) $i_o$ is the output membrane of $\Pi$. In this study, $i_o=0$, namely, the output results will be collected inside the skin membrane.

2) The system is non-cooperative, that is, the length of the object in the left hand side of an evolution rule is one.
3) The system is deterministic and non-halting. Thus, non-deterministic membrane systems will not be removed in this design and the target P systems will stop if no termination condition is predefined.

The aim of the design is to obtain a successful P system with the above features through tuning the syntactical ingredients $\mu, W$ and $\Re$.

A concise description on the P system for fulfilling a computational task is as follows. The multisets associated to regions form a configuration of the P system. The computation begins by treating the initial multisets, $w_i$, $0\leq i\leq m-1$, and then

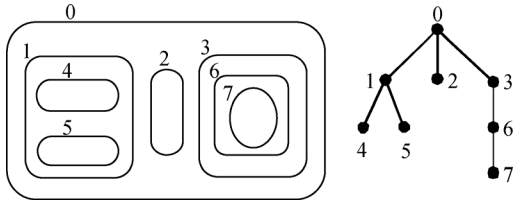



Fig. 1. An example for a cell-like P system membrane structure and its associated tree.

the system will go from one configuration to a new one by applying the evolution rules associated to regions in a deterministic and maximally parallel way, namely, all the objects that may be transformed or communicated must be dealt with. The system will halt when no more rules are available to be applied. A computation is a sequence of configurations obtained as it is described above, starting with the initial configuration and ending with the configuration when the system halts. The result of a computation, a multiset of objects, is obtained in the output region, $i_o$. For more details about P systems description see [2].

## III. Design Method

To design a P system with the prescribed requirements, it is necessary to consider the following three points: representation of a P system, evaluation of a candidate P system, and evolution of a family of P systems toward the expected result. In this section, we propose a P system permutation encoding representation, a penalty function evaluation of a candidate P system, and a genetic algorithm for the P system evolution toward the expected result. In what follows, we first present the three techniques and then we summarize the design method to provide an algorithmic elaboration.

### A. Representation of P Systems

In this study, we use the permutation encoding technique [14] to codify a P system. The representation of a P system consists of the encoding approaches for the alphabet $V$, its membrane structure $\mu$, the initial multiset vector $W$, evolution rules set $\Re$ and an individual chromosome corresponding to a candidate P system. In what follows, we describe these approaches one by one.

*1) Encoding of $V$:* Suppose that there are $N_V$ objects (letters), we use $N_V$ strictly positive integers to represent the objects and 0 to denote the empty set $\lambda$. Thus, $V$ is encoded as an ordered string of numbers, namely, "$01 \ldots N_V$". For instance, if $V = \{a, b, c\}$, its codes are "0123."

*2) Encoding of $\mu$:* The hierarchical membrane structure of a cell-like P system can also be denoted as a rooted tree. Thus, we can use the label of the parent (the neighboring outer membrane, like the parent of a node in a tree) of each membrane to form an ordered string to represent a P system structure. It is worth noting that the skin membrane is not considered in the string because it is the outermost membrane in the structure. Thus, the hierarchical membrane structure of the P system with $N_\mu$ membranes is represented with a string with $(N_\mu - 1)$ numbers. For example, the structure in Fig. 1 can be represented as the codes "0001136."

*3) Encoding of $W$:* Each element $w_i, i = 0, 1, \ldots, m-1$, of the vector $W$ are strings over $V$. The encoding approach of $W$ is designed according to the encoding technique of $V$. Suppose that the largest number of objects in $w_i$ is $N_{w_i}$, so $w_i$ needs $N_{w_i}$ codes, each of which may be $0, 1, \ldots$, or $N_V$. The codes of $W$ can be obtained by concatenating the string of $w_i$, $i = 0, 1, \ldots, m-1$, and a separator symbol $N_V+4$ is used to delimit the codes of $w_i$ and $w_{i+1}$, $i = 0, 1, \ldots, m-2$. Thus, the total number $L_W$ of codes for $W$ is

$$L_W = \sum_{i=0}^{m-1} N_{w_i} + m - 1. \tag{1}$$

For example, $W = [w_0, w_1, w_2]$ is the initial multiset vector of a P system. $N_{w_0} = \lambda$, $N_{w_1} = aa$, $N_{w_2} = bbcc$. Thus, $L_W = 9$ and the string for encoding $W$ is "071172233."

*4) Encoding of $\Re$:* The left hand side $u$ and the right hand side $v$ of the rule (rewriting, dissolution or rewriting-communication rule) are elements of $V$ and $V^*$, respectively. On the basis of the representation of $V$, we can encode the set $\Re$. Suppose that the number of rules in $R_i$ is $N_{R_i}, i = 0, 1, \ldots, m-1$, and the largest numbers of objects in the left hand side $u$ and in the right hand side $v$ of a rule are $N_l$ and $N_r$, respectively. Thus, we use $N_l$ codes, each of which may be $1, 2, \ldots$, or $N_V$, $N_r$ codes, each of which may be $0, 1, \ldots$, or $N_V$, and additional one code to describe its rule type (here we use $N_V+1$, $N_V+2$ and $N_V+3$ denote a rewriting, dissolution, and rewriting-communication rules, respectively) to encode a rule. Thus, the code length $L_{R_i}$ for the rule is $N_l + N_r + 1$, namely, $L_{R_i} = N_l + N_r + 1$. The codes of $\Re$ can be gained by concatenating the string of each rule and by using a separator symbol $N_V + 5$ between $R_i$ and $R_{i+1}$, $i = 0, 1, \ldots, m-2$. So the total code length $L_\Re$ of the set $\Re$ is

$$L_\Re = \sum_{i=0}^{m-1} (N_{R_i} * L_{R_i}) + m - 1. \tag{2}$$

It is worth noting that the dissolution rule is a structural rule, which is applied at most one at each step of a P system evolution, and rewriting and rewriting-communication rules can be normally applied in a maximally parallel mode.

For instance, we encode the set $\Re = \{R_0, R_1, R_2\}$, where $R_0 = \{[a \rightarrow aab], [b \rightarrow cc]\}$, $R_1 = \{[a \rightarrow cc], [b] \rightarrow [b]a\}$ and $R_2 = \{[a] \rightarrow bc\}$, as the code string "11124233481334221681235."

*5) Encoding of a P System:* In this study, we design a P system through tuning membrane structure, initial objects and evolution rules, thus, the codes for the P system can be attained by sequentially concatenating the codes of $\mu$, $W$ and $\Re$, and a separator symbol $N_V + 6$ to enable the separation of the codes of $\mu$, $W$ and $\Re$. We illustrate the encoding of a P system with the following example. Consider the following P system:

$$\Pi_e = (V, O, \mu, W, \Re, i_o)$$

where

1) $V = \{s, a, b\}$;
2) $O = \{s\}$;
3) $\mu = [[[\,]_2[\,]_3]_1]_0$;
4) $W = [w_0, w_1, w_2, w_3]$, $w_0 = \lambda$, $w_1 = b$, $w_2 = a$, $w_3 = b$;

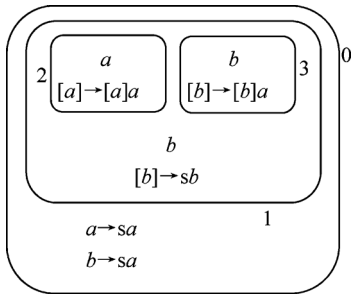



Fig. 2. The initial configuration of $\Pi_e$ (with rules included).

5) $\Re = \{R_0, R_1, R_2, R_3\}$, $R_0 = \{a \to sa, b \to sa\}$, $R_1 = \{b \to sb\}$, $R_2 = \{[a] \to [a]a\}$, $R_3 = \{[b] \to [b]a\}$;
6) $i_o = 0$.

The initial configuration of the P system $\Pi_e$ is illustrated in Fig. 2. If $N_{w_0} = N_{w_1} = N_{w_2} = N_{w_3} = 1$, $N_{R_0} = 2$, $N_{R_1} = N_{R_2} = N_{R_3} = 1$, $N_l = 1$, $N_r = 2$ and $L_{R_i} = 4$, ($i = 0, 1, 2, 3$), the P system $\Pi_e$ is encoded as the string "01190323921243124831358222683326."

### B. Evaluation of P Systems

How to evaluate a candidate P system is a crucial step in the automatic design of membrane systems by using evolutionary algorithms. This step has a direct effect on the characteristics of the P systems obtained and the performance of the design algorithm. In the evaluation, we consider the following seven aspects:

1) The difference between the actual number(s) and the expected number(s) of output objects. The former refers to the simulated result that is returned from the specialized P system simulation software, P-Lingua [15], [16], through inputting a candidate P system into the software. The latter is designated by the designer according to the computational task or the problem to solve.
2) The feasibility of a P system due to its membrane structure $\mu$. In the design, some infeasible membrane structures may be generated by the evolutionary operations such as crossover or mutation in a genetic algorithm. The infeasible membrane structure refers to the one that does not satisfy the syntactical requirement of the P system described in Section II.
3) The redundancy of objects in the initial multiset vector $W$. In this design, some objects exist in the initial multiset vector $W$, but they will not be used through the computation of the P system. We call them *redundant objects*. This redundancy results from the randomness of the generation of the population of initial P systems in an evolutionary algorithm.
4) The non-determinism of a P system resulting from non-deterministic membrane systems due to evolution rules.
5) The infeasibility of a P system due to more than one dissolution rules in one set $R_i$ ($i = 0, 1, \ldots, m-1$).
6) The redundancy of evolution rules in the set $\Re$. The redundant rules refer to the ones in the set $\Re$ that are not used through the computation of the P system.
7) A halting P system due to evolution rules.

It is worth pointing out that the further explanations for 2)–7) will be described in Section III-D.

Based on the above analysis, we define the evaluation function as follows:

$$f = f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 \tag{3}$$

where

$$f_1 = g_1(N_s) = \sum_{i=1}^{N_{obj}} |N_i^{ao} - N_i^{eo}| \tag{4}$$
$$f_2 = g_2(\mu) = \delta \cdot M_f \tag{5}$$
$$f_3 = g_3(W) = \eta \cdot N_{obs} \tag{6}$$
$$f_4 = g_4(\Re) = \alpha \cdot N_{non} \tag{7}$$
$$f_5 = g_5(\Re) = \beta \cdot R_{dis} \tag{8}$$
$$f_6 = g_6(\Re) = \gamma \cdot N_{red} \tag{9}$$
$$f_7 = g_7(\Re) = \xi \cdot H \tag{10}$$

where

— $f_1$ is the object error function; $g_1(N_s)$ is the function of the simulation step $N_s$ of a candidate P system in the P-Lingua software and is designed according to the computational task; $N_i^{ao}$ and $N_i^{eo}$ are the actual number and the expected number of the $i$th ($i = 1, 2, \ldots, N_{obj}$) output objects, respectively; $N_{obj} = |O|$; $N_{obj}$ is the number of distinct letters involved in the output objects;
— $f_2$ is the penalty item of the infeasible membrane structure; $g_2(\mu)$ is the function of the membrane structure $\mu$; $\delta$ is a penalty factor; $M_f \in \{0, 1\}$, where "0" and "1" mean that the membrane structure of a candidate P system is feasible and infeasible, respectively;
— $f_3$ is the penalty item of the redundant objects in the initial multiset vector $W$; $g_3(W)$ is the function of the initial multiset vector $W$; $\eta$ is a penalty factor; $N_{obs}$ is the number of the redundant objects in the initial multiset vector $W$;
— $f_4$ is the penalty item of a non-deterministic P system; $g_4(\Re)$ is a function of the set $\Re$; $\alpha$ is a penalty factor; $N_{non} \in \{0, 1\}$, where "0" and "1" mean that there is not any non-deterministic evolution rule and there is at least one pair of non-deterministic evolution rules in the set $\Re$, respectively;
— $f_5$ is the penalty item of the dissolution rules; $g_5(\Re)$ is a function of the set $\Re$; $\beta$ is a penalty factor; $R_{dis} \in \{0, 1\}$, where "0" and "1" mean that there is less than and at least two dissolution rules in one set $R_i$ ($i = 0, 1, \ldots, m-1$), respectively;
— $f_6$ is the penalty item of the redundant rules; $g_6(\Re)$ is a function of the set $\Re$; $\gamma$ is a penalty factor; $N_{red}$ is the number of the redundant rules in the set $\Re$;
— $f_7$ is the penalty item of the halting P system; $g_7(\Re)$ is a function of the set $\Re$; $\xi$ is a penalty factor; $H \in \{0, 1\}$, where "0" and "1" mean that the candidate P system is a non-halting and halting one, respectively.

In (5)–(10), the introduction of the penalty factors $\delta$, $\alpha$, $\beta$, and $\xi$ is to reject the unexpected candidate P systems and therefore the five factors can be assigned as a larger value as possible, e.g., $\delta = \alpha = \beta = \xi = 999999$; while the use of the two factors $\eta$

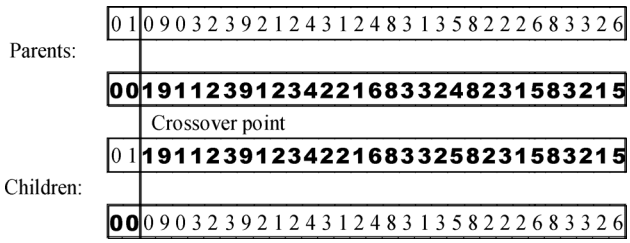



Fig. 3. One point crossover.

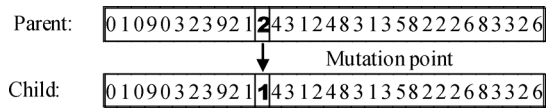



Fig. 4. Uniform mutation.

and $\gamma$ is to remove those candidate P systems having redundant objects or evolution rules as possible as we could and accordingly they can be prescribed as smaller values. The setting of the two factors $\eta$ and $\gamma$ will be discussed in Section IV-A.

### C. Evolution of P Systems

In this study, we apply the genetic algorithm with the permutation encoding technique (GAPE) in JGAP [17] to evolve a family of P systems toward a successful one. GAPE uses the elitist selection strategy, where twenty percent of individuals with the best fitness values are selected to pass to the next generation, being free of the crossover and mutation operators. In GAPE, one-point crossover and uniform mutation are used. We use the representation of P systems in Section III-A5 to illustrate the crossover operation, as shown in Fig. 3, where a single crossover point except for separator symbols on both parents' chromosome strings is selected and all codes beyond that point in either chromosome string is swapped between the two parent chromosomes, and the mutation operation, shown in Fig. 4, where the value of the chosen gene (except for separator symbols) with a uniform random value selected through the string of a P system is replaced by any code in $V$. The resulting chromosomes are the children. The crossover and mutation operations are performed by the probabilities $P_c$ and $P_m$, respectively.

It is worth noting that the evolutionary operators might produce the P systems violating the constraints in (5)–(10) including infeasible membrane structures $\mu$, more than one dissolution rules in one set $R_i$ $(i = 0, 1, \ldots, m-1)$, the redundancy of objects in the initial multiset vector $W$, the redundancy of evolution rules in the set $\Re$, the non-deterministic evolution rule pairs and the halting P system due to evolution rules.

### D. Algorithmic Elaboration

This subsection summarizes the design method PPGA as shown in Fig. 5, where each step is described as follows:

1) This step consists of two processes: the setting of initial parameter values and the generation of initial population. The former process is used to set initial values for $N_V$ $N_{w_i}$, $N_{R_i}$, $L_{R_i}$, $i = 0, 1, \ldots, m-1$, $N_l$, $N_r$, population size $N_P$, $P_c$ and $P_m$, $\delta$, $\eta$, $\alpha$, $\beta$, $\gamma$, $\xi$, the maximal number $MaxGen$ of evolutionary generations as the termination condition of GAPE and the maximal number $MaxStep$ of simulation steps for a P system in the P-Lingua software. The latter process produces a population with $N_P$ individuals, each of which corresponds to a candidate P system, according to the representation described in Section III-A.
2) Each individual is evaluated by using Algorithm 1 and thus, obtains its fitness. In Algorithm 1, the values of the variables, $M_f$, $N_{obs}$, $N_{non}$, $R_{dis}$, $N_{red}$ and $H$, depend on the following constraint recognition techniques:
   a) Infeasible P systems due to infeasible membrane structures: a P system is an infeasible one if it satisfies one of the three conditions: (i) the parent membrane of any one membrane is itself; (ii) the system has not the skin membrane; (iii) two or more membranes form a parent membrane loop, for example, membrane 1 is the parent of membrane 2, membrane 2 is the parent of membrane 3, and membrane 3 is the parent of membrane 1;
   b) Redundant objects: the objects in $W$ do not appear in the left hand side $u$ of all evolution rules in $\Re$.
   c) Non-deterministic P systems have two cases: i) two or more evolution rules in $R_i$ $(i = 0, 1, \ldots, m-1)$ have the identical left hand side $u$; ii) two or more evolution rules in $R_i$ $(i = 0, 1, \ldots, m-1)$ can be applied within one transition, that is, the left hand side objects of two or more evolution rules in $R_i$ $(i = 0, 1, \ldots, m-1)$ can be provided in the current configuration.
   d) Infeasible P systems due to dissolution rules: a P system is an infeasible one if there are two or more dissolution rules in $R_i$ $(i = 0, 1, \ldots, m-1)$ according to the codes describing the rule types.
   e) Redundant evolution rules: an evolution rule is redundant in two cases: i) if the evolution rule in which all the objects in the left hand side do not appear both in the initial multiset and in the right hand side of any one rule in the membrane; ii) if the evolution rule in which the objects in the left hand side are identical with those in the right hand side, and they are neither the expected ones nor appear in the left hand side of any rule in the membrane.
   f) Halting P systems: if there is not any *iterative loop* consisting of one or more evolution rules, the system is a halting one. An *iterative loop* may be one of the following cases: i) One evolution rule forms an *iterative loop*, that is, if one evolution rule $leftObj \rightarrow rightObj$ has the feature $leftObj \subset rightObj$, the rule forms

```
Begin
    t ← 1
1)  Initialization
    While (not termination condition) do
2)      Evaluation
3)      Storage of the best solution
4)      Selection
5)      Crossover
6)      Mutation
        t ← t + 1
    End
End
```

Fig. 5. Pseudocode algorithm of PPGA.

an *iterative loop*; ii) Several evolution rules form an *iterative loop*. If $N_{il}$ evolution rules, $leftObj_1 \rightarrow rightObj_1, leftObj_2 \rightarrow rightObj_2, leftObj_3 \rightarrow rightObj_3, \ldots, leftObj_{N_{il-1}} \rightarrow rightObj_{N_{il-1}}, leftObj_{N_{il}} \rightarrow rightObj_{N_{il}}$, have the features, $leftObj_2 \subseteq rightObj_1, leftObj_3 \subseteq rightObj_2, \ldots, leftObj_{N_{il}} \subseteq rightObj_{N_{il-1}}, leftObj_1 \subseteq rightObj_{N_{il}}$, the rules form an *iterative loop*.

3) The best solution and its corresponding P system are stored.
4) The elitist selection strategy described in Section III-C is considered.
5) The one-point crossover operator is used and depicted in Section III-C.
6) The uniform mutation operator is employed and illustrated in Section III-C.

**Algorithm 1** Evaluation method

**Input:** A candidate P system

1: $f \leftarrow 0$

2: Compute $M_f, N_{obs}, N_{non}, R_{dis}, N_{red}, H$

3: **if** $((M_f > 0) \parallel (N_{non} > 0) \parallel (R_{dis} > 0))$ **then**

4: $\quad f \leftarrow f_2 + f_4 + f_5$

5: **else**

6: $\quad N_s \leftarrow 0$

7: $\quad$ **while** $(H < 1) \wedge (N_s \leq MaxStep)$ **do**

8: $\quad\quad$ Evolve the P system for one step

9: $\quad\quad N_s \leftarrow N_s + 1$

10: $\quad\quad f \leftarrow f + f_1$

11: $\quad$ **end while**

12: $\quad$ **if** $(H > 0)$ **then**

13: $\quad\quad f \leftarrow f_7$

14: $\quad$ **end if**

15: $\quad$ **if** $(f = 0)$ **then**

16: $\quad\quad f \leftarrow f + f_3 + f_6$

17: $\quad$ **end if**

18: **end if**

19: **Output:** Fitness $f$

## IV. Experiments and Results

In this section, the parameter setting of PPGA is first discussed; then an example is used to show the PPGA feasibility; finally PPGA is applied to design the minimal P system with respect to their membrane structures, alphabet, initial objects and evolution rules for fulfilling a given computing task. All the following experiments are implemented on the platform Java and on a HP work station with Intel Xeon 2.93 GHz processor, 12 GB RAM and Windows 7 OS.

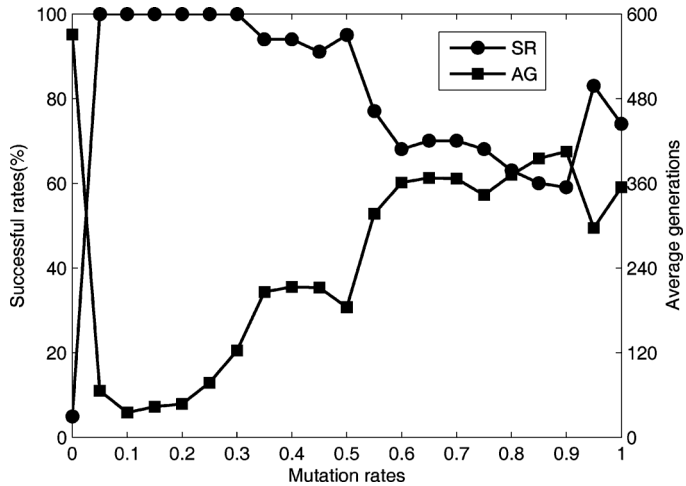



Fig. 6. Experimental results of $P_m$.

### A. Parameter Setting

As described in Section III, six parameters, mutation probability $P_m$, crossover probability $P_c$, population size $N_P$, the maximal number $MaxGen$ of evolutionary generations, two penalty factors $\eta$ and $\gamma$, in PPGA need to be discussed. In what follows, we consider the P system $\Pi_{ex}$ as a design example to discuss the parameter setting of PPGA to obtain some guidelines.

$$\Pi_{ex} = (V, O, \mu, W, \Re, i_o)$$

where $N_V = 3$, namely, $V = \{s, a, b\}$; $O = \{s\}$, $N_\mu = 4$; $N_{w_0} = N_{w_1} = N_{w_2} = N_{w_3} = 1$; $N_{R_0} = 2$, $N_{R_1} = N_{R_2} = N_{R_3} = 1$; $N_l = 1$, $N_r = 2$ and $L_{R_i} = 4$, $(i = 0, 1, 2, 3)$. In the following experiments, the maximal number $MaxStep$ of simulation steps for a candidate P system in P-Lingua assigned is 20; for each parameter value, we perform 100 independent tests, that is, the experimental results are statistical mean values over 100 independent tests.

First of all, we investigate the effect of $P_m$ on the PPGA performance. In the experiment, $P_m$ increases from 0 to 1 with the interval 0.05; $P_c$, $N_P$, $MaxGen$, $\eta$ and $\gamma$ are set to 0.8, 20, 600, 1 and 1, respectively. When we use PPGA to design the system $\Pi_{ex}$, we record the successful rate (SR) and average generation (AG) for each $P_m$ value. SR refers to the ratio of the number of successful computations to 100 independent tests. AG is the average of the evolutionary generations over 100 independent tests when the algorithm stops for each case. Experimental results are shown in Fig. 6 and indicate that PPGA obtains the highest SR and smallest AG when $P_m = 0.1$.

Secondly, we discuss the setting of $P_c$. In the experiment, let $P_c$ vary from 0 to 1 with the interval 0.05. The rest parameters, $P_m$, $N_P$, $MaxGen$, $\eta$ and $\gamma$ are set to 0.1, 20, 600, 1 and 1, respectively. Similarly, we record SR and AG for each $P_c$ value. Fig. 7 shows the experimental results and exhibits that the $P_c$ change does not affect SR, while AG reaches a lower value when $P_c$ increases from 0.6 to 1.

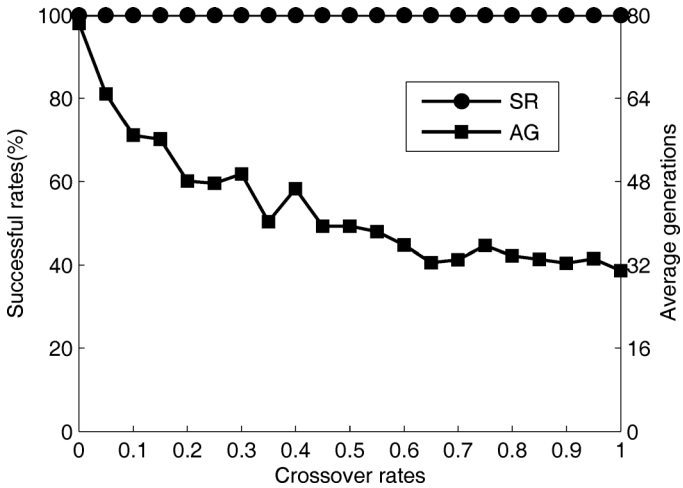



Fig. 7. Experimental results of $P_c$.

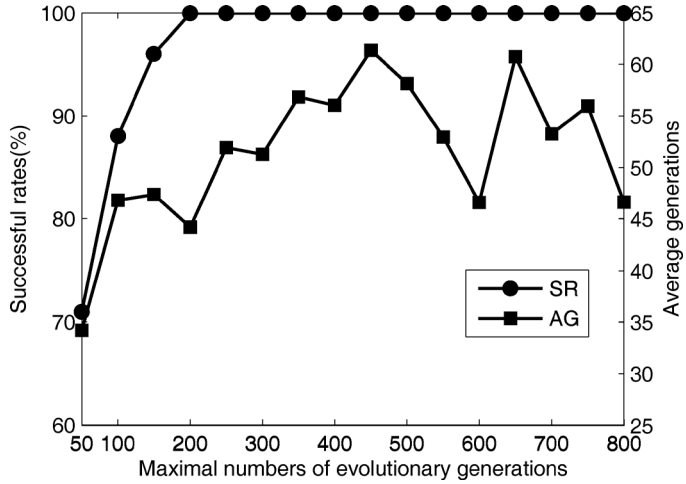



Fig. 9. Experimental results of $MaxGen$.

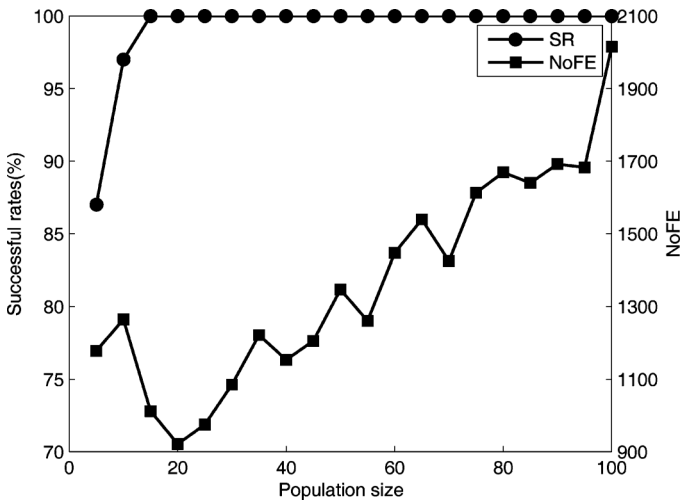



Fig. 8. Experimental results of $N_P$.

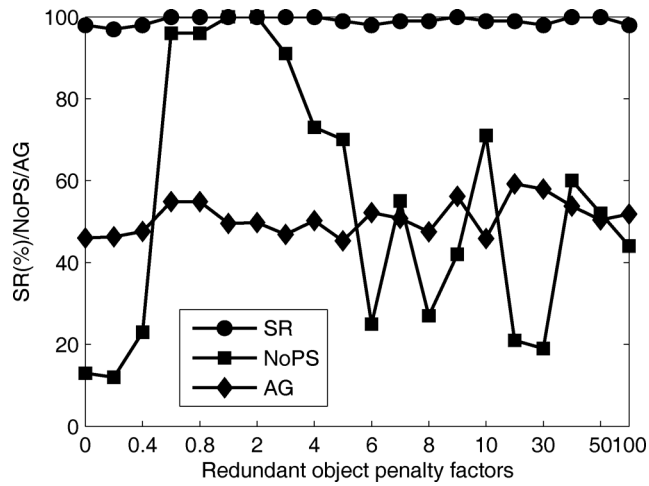



Fig. 10. Experimental results of $\eta$.

Thirdly, the investigation of $N_P$ is involved. In the experiment, the population size $N_P$ goes up from 5 to 100 with an interval 5; other parameters, $P_m$, $P_c$, $MaxGen$, $\eta$ and $\gamma$ are set to 0.1, 0.8, 600, 1 and 1, respectively. SR and the total number of function evaluations (NoFE) are used to evaluate the algorithm performance. NoFE refers to the total number of the fitness function evaluations for candidate P systems in 100 independent runs. Experimental results are given in Fig. 8. It can be seen from this figure that $N_P$ could be assigned as 20.

Next, we describe the discussion of $MaxGen$. In the experiment, the value of $MaxGen$ grows from 50 to 800 at a rate 50; the parameters, $P_m$, $P_c$, $N_P$, $\eta$ and $\gamma$ are set to 0.1, 0.8, 20, 1 and 1, respectively. The changes of SR and AG with $MaxGen$ are illustrated in Fig. 9. The results demonstrate that it is enough for PPGA to set $MaxGen$ to 200 to obtain $\mathrm{SR} = 100$.

Subsequently, the effect of $\eta$ on the PPGA performance is investigated. In the experiment, $\eta$ has twenty choices: 0, 0.2, 0.4, 0.6, 0.8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50 and 100; $P_m$, $P_c$, $N_P$, $MaxGen$ and $\gamma$ are set to 0.1, 0.8, 20, 200 and 1, respectively. The changes of SR, AG and the number of successful P system variants (NoPS) against $\eta$ are provided in Fig. 10. The experimental results reveal that PPGA achieves the best performance with respect to SR, AG and NoPS when $\eta = 1$.

Finally, we discuss the guideline for setting $\gamma$. In the experiment, the $\gamma$ value is sequentially chosen from the range [0, 0.2, 0.4, 0.6, 0.8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 100]; $P_m$, $P_c$, $N_P$, $MaxGen$ and $\eta$ are assigned as 0.1, 0.8, 20, 200 and 1, respectively. Experimental results are shown in Fig. 11 and disclose that the best values of SR, AG and NoPS are gained at $\gamma = 1$.

### *B. Design Examples*

In this subsection, the design of the cell-like P system $\Pi_{ex}$, which is described in Section IV-A, for fulfilling the computation $n^2$ is discussed to test the feasibility and effectiveness of PPGA. The parameters, $P_m$, $P_c$, $N_P$, $MaxGen$, $\eta$ and $\gamma$ are set to 0.1, 0.8, 20, 200, 1 and 1, respectively. We perform 5000 independent runs of the design experiment and obtain the successful rate 100%. The introduced design approach obtains 2930 different variants of cell-like P systems $\Pi_{ex}$ for successfully fulfilling the computation of $n^2$. Due to page limit, Table I lists only five successful P systems. The complete list of the 1936 successful P systems can refer to *http://www.nicsg.net/*

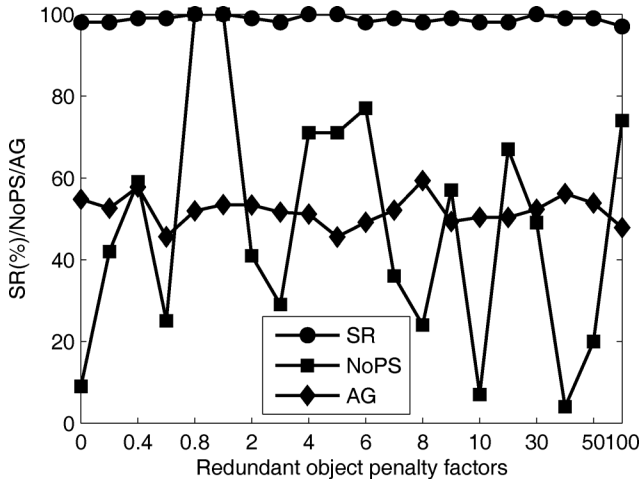



Fig. 11. Experimental results of $\gamma$.

TABLE I
SUCCESSFUL P SYSTEMS

| No | $\mu$ | $W$ | $\Re$ |
|---|---|---|---|
| 1 | $[[\,]_1[\,]_2[\,]_3]_0$ | $w_0 = \lambda$<br>$w_1 = b$<br>$w_2 = a$<br>$w_3 = a$ | $[b \to bs]_0$<br>$[a \to ba]_0$<br>$[b]_1 \to ab$<br>$[a]_2 \to ab$<br>$[a]_3 \to sb$ |
| 2 | $[[[\,]_1]_2[\,]_3]_0$ | $w_0 = \lambda$<br>$w_1 = a$<br>$w_2 = a$<br>$w_3 = a$ | $[b \to bs]_0$<br>$[a \to b^2]_0$<br>$[a]_1 \to [a]_1 a$<br>$[a]_2 \to sb$<br>$[a]_3 \to b^2$ |
| 3 | $[[\,]_1[\,]_2[\,]_3]_0$ | $w_0 = \lambda$<br>$w_1 = a$<br>$w_2 = a$<br>$w_3 = b$ | $[b \to bs]_0$<br>$[a \to ab]_0$<br>$[a]_1 \to ab$<br>$[a]_2 \to sb$<br>$[a]_3 \to ab$ |
| 4 | $[[[\,]_1[\,]_3]_2]_0$ | $w_0 = \lambda$<br>$w_1 = a$<br>$w_2 = b$<br>$w_3 = a$ | $[b \to as]_0$<br>$[a \to sa]_0$<br>$[a]_1 \to [a]_1 a$<br>$[b]_2 \to sb$<br>$[a]_3 \to [a]_3 a$ |
| 5 | $[[\,]_1[\,]_2[\,]_3]_0$ | $w_0 = \lambda$<br>$w_1 = b$<br>$w_2 = b$<br>$w_3 = a$ | $[b \to sa]_0$<br>$[a \to sa]_0$<br>$[b]_1 \to [b]_1 a$<br>$[b]_2 \to [b]_2 a$<br>$[a]_3 \to sb$ |

*portal.php?mod=view&aid=165*, where $\mu$, $W$ and $\Re$ are the membrane structure, the vector of initial multisets and the set of evolution rules in a successful P system, respectively. Due to the randomness of the selection of membrane structure, objects and rules, we can obtain multiple solutions for the same computational task on the identical condition to provide multiple possibilities to construct different complex membrane systems.

### C. Minimal P System Design

In this subsection, we try to use PPGA to find the minimal P system for fulfilling the given computational task $n^2$. Here the minimal P system refers to the one with the simplest membrane structure, the smallest number of objects in $V$, the smallest number of initial objects, the smallest numbers of evolution rules and the smallest number of objects present in their left and right hand sides of its rules.

TABLE II
SUCCESSFUL MINIMAL P SYSTEMS

| No | $\mu$ | $W = [w_0]$ | $\Re = [R_0]$ |
|---|---|---|---|
| 1 | $[\,]_1$ | $[s]_1$ | $[s \to bsa,\ a \to ab^2]_1$ |
| 2 | $[\,]_1$ | $[a]_1$ | $[s \to b^2 s,\ a \to abs]_1$ |

We start with the simplest case: one membrane, two objects in $V$, one initial object, one evolution rule with one left hand object and two right hand objects. In the experiment, we set the values of the parameters, $P_m$, $P_c$, $N_P$, $MaxGen$, $\eta$ and $\gamma$ to 0.1, 0.8, 20, 200, 1 and 1, respectively. Totally 5000 independent tests are performed. Unfortunately no successful P system can be obtained. Next, we adjust the initial setting of the P system as follows: one membrane, two objects in $V$, one initial object, two evolution rules, each of which has one left hand object and two right hand objects. 5000 independent tests are repeated in an attempt to obtain a successful P system, but we get nothing. So we go further to change the initial configuration: one membrane, three objects in $V$, one initial object, two evolution rules, each of which has one left hand object and three right hand objects. Thus, we achieve two successful P systems, which are listed in Table II.

## V. CONCLUSIONS

This study discusses the automatic design of a deterministic and non-halting P system by tuning membrane structures, initial objects and evolution rules. In this design, a permutation encoding technique is introduced to represent a P system including its hierarchical membrane structure, initial objects and evolution rules; a penalty function for evaluating a candidate P system is presented through considering the feasibility of a P system due to its membrane structure and dissolution rules, the redundancy of objects and evolution rules, non-determinism and halting characteristic; a GA is used to guide a family of P systems toward a successful one. In addition, this study considers the rewriting-communication rule in the design, the parameter setting of PPGA and the design of the minimal P system. A large number of experiments verify the feasibility and effectiveness of the proposed method. This work is the foundation of our future design of general polynomial and more complicated P systems.

## ACKNOWLEDGMENT

The authors would like to thank the Editor-in-Chief, Professor M. Hughes and Professor H. Hess, the editors and the anonymous reviewers for their insightful recommendations and comments to improve this paper.

## REFERENCES

[1] G. Păun, "Computing with membranes," *J. Comput. Syst. Sci.*, vol. 61, pp. 108–143, 1998.
[2] G. Păun, G. Rozenberg, and A. Salomaa, *The Oxford Handbook of Membrane Computing*. New York: Oxford Univ. Press, 2010.
[3] T. Song, L. Pan, J. Wang, I. Venkat, K. G. Subramanian, and R. Abdullah, "Normal forms of spiking neural P systems with anti-spikes," *IEEE Trans. NanoBiosci.*, vol. 11, no. 4, pp. 352–359, 2012.
[4] L. Pan and X. Zeng, "Small universal spiking neural P systems working in exhaustive mode," *IEEE Trans. NanoBiosci.*, vol. 10, no. 2, pp. 99–105, Jun. 2011.

[5] G. Zhang, J. Cheng, M. Gheorghe, and Q. Meng, "A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems," *Appl. Soft Comput.*, vol. 13, no. 3, pp. 1528–1542, Mar. 2013.
[6] X. Zeng, T. Song, X. Zhang, and L. Pan, "Performing four basic arithmetic operations with spiking neural P systems," *IEEE Trans. NanoBiosci.*, vol. 11, no. 4, pp. 366–374, 2012.
[7] C. Buiu, C. Vasile, and O. Arsene, "Development of membrane controllers for mobile robots," *Inf. Sci.*, vol. 187, pp. 33–51, Mar. 2012.
[8] G. Escuela and M. A. Gutiérrez-Naranjo, "An application of genetic algorithms to membrane computing," in *Proc. 8th Brainstorming Week Membrane Comput.*, 2010, pp. 101–108.
[9] X. Huang, G. Zhang, H. Rong, and F. Ipate, M. Gheorghe, G. Păun, G. Rozenberg, A. Salomaa, and S. Verlan, Eds., "Evolutionary design of a simple membrane system," in *Proc. Membrane Comput. (CMC2011)*, vol. 7184, Lecture Notes in Computer Science, pp. 203–214.
[10] C. Tudose, R. Lefticaru, and F. Ipate, "Using genetic algorithms and model checking for P systems automatic design," in *Nature Inspired Cooperative Strategies for Optimization*, D. A. Pelta, N. Krasnogor, D. Dumitrescu, C. Chira, and R. Lung, Eds. New York: Springer, 2011, vol. 387, Studies in Computational Intelligence, pp. 285–302.
[11] Y. Chen, G. Zhang, T. Wang, and X. Huang, "Automatic design of a P system for basic arithmetic operations," *Chinese J. Electron.*, vol. 23, no. 2, pp. 302–304, 2014.
[12] G. Zhang, M. Gheorghe, L. Pan, and M. J. Pérez-Jiménez, "Evolutionary membrane computing: A comprehensive survey and new results," *Inf. Sci.*, vol. 279, pp. 528–551, 2014.
[13] Z. Ou, G. Zhang, T. Wang, and X. Huang, "Automatic design of cell-like P systems through tuning membrane structures, initial objects and evolution rules," *Int. J. Unconventional Comput.*, vol. 9, no. 5-6, pp. 425–443, 2013.
[14] S. Ronald, "Robust encodings in genetic algorithms: A survey of encoding issues," in *Proc. IEEE Int. Conf. Evol. Comput.*, Indianapolis, IN, USA, Apr. 1997, pp. 43–48.
[15] M. García-Quismondo, R. Gutiérrez-Escudero, M. A. Martinez-del-Amor, E. Orejuela-Pinedo, and I. Pérez-Hurtado, "P-Lingua 2.0: A software framework for cell-like P systems," *Int. J. Comput. Commun. Control*, vol. 4, pp. 234–243, Sep. 2009.
[16] M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and A. Riscos-Núñez, G. Păun, M. J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, and A. Salomaa, Eds., "An overview of P-Lingua 2.0," in *Proc. Workshop Membrane Comput.*, 2010, vol. 5957, Lecture Notes in Computer Science, pp. 264–288.
[17] K. Meffert, J. Meseguer, E. Mart, J. Vos, A. Meskauskas, and N. Rotstan, JGAP—Java Genetic Algorithms Package [Online]. Available: http://jgap.sf.net

**Gexiang Zhang** (M'03) received his Ph.D. degree in 2005 from Southwest Jiaotong University, Chengdu, China. Since 2005, he has been a Professor at the School of Electrical Engineering in Southwest Jiaotong University, where he leads the research group of Nature-Inspired Computation and Smart Grid (NICSG).

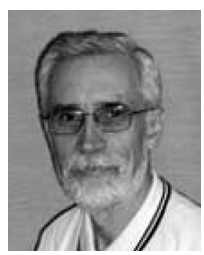
His research interests include natural computing (membrane computing, evolutionary computation, and DNA computing), smart grid and robotics.

**Haina Rong** received her Ph.D. in 2010 from Southwest Jiaotong University, China, where she is now a lecturer at the School of Electrical Engineering. His research interests cover natural computing, such as membrane computing and evolutionary computation, and smart grid.

**Zhu Ou** received his master's degree in 2013 from Southwest Jiaotong University. His research interests include membrane computing and evolutionary computation.

**Mario J. Pérez-Jiménez** received his Ph.D. degree in mathematics from the University of Sevilla, Sevilla, Spain in 1992. Currently, he is a numerary member of the Academia Europaea (The Academy of Europe), and a full Professor in Computer Science and Artificial Intelligence, University of Sevilla, where he is the head of the Research Group on Natural Computing.

His main research interests include theory of computation, computational complexity theory, natural computing (DNA computing and membrane computing), bioinformatics and computational modeling for systems biology and population dynamics.



**Marian Gheorghe** received his Ph.D. degree from the University of Bucharest, Romania, in 1991.

He is currently Reader with the Department of Computer Science, Sheffield University, U.K., and Head of the Verification and Testing Group. He has extensively published on topics regarding computational models, such as automata and languages, or unconventional computing, especially membrane computing and membrane algorithms. He has also contributions in verification and testing, software engineering, agent-based systems, systems and synthetic biology.