3 Applications of Software Implementations of P Systems

3.1 Introduction

Since the introduction of membrane computing in 1998 [49], there has been a rapid theoretical development in this area with respect to computing models and their computing power and computational efficiency [50, 51]. Moreover, numerous real-world applications of membrane computing models [72, 76, 77] have been reported. P systems can perform specific tasks such as solving an NP-hard [48], NP-complete [37, 62], or PSPACE-complete problems [1], control language generation [63, 75], controlling robots [79], heuristic optimization problems [27, 71, 73], and arithmetic operation [78]. These tasks were accomplished manually, instead of by means of automatic design. The manual implementation of membrane computing models has some drawbacks. For example, it could be time-consuming, tedious, and impossible to implement large-scale systems. It limits the application scope of P system models. Therefore, the question on how to automatically design a P system by using programs, namely, the *programmability of a P system*, has become an urgent and attractive research direction in the area of membrane computing [74].

The automatic design of a P system is a very complicated and challenging task [70, 80]. There has been many works focused on the use of evolutionary algorithms to make a population of P systems evolve toward a successful one [34, 72]. These works start 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 predefined and fixed in the process of design [7, 23, 28, 66, 72]. In [23], a genetic algorithm was used to design a P system to calculate 4^2. In [28], a binary encoding technique was presented to denote an evolution rule set of a P system, and a quantum-inspired evolutionary algorithm (QIEA) was used to make a population of P systems evolve toward successful ones. This method successfully solved the design of P systems to compute 4^2 and n^2 (for natural numbers $n \geq 2$). In [66], an evaluation approach considering nondeterminism and halting penalty factors and a genetic algorithm with the binary encoding technique in [28] were introduced to

G. Zhang et al., *Membrane Computing Models: Implementations*,
https://doi.org/10.1007/978-981-16-1566-5_3

design P systems for computing 4^2, n^2, and the generation of language $\{a^{2^n} b^{3^n} | n > 1\}$. In these studies mentioned above, a specific redundant evolution rule set was designed for a specific computational task. This was developed in [7,72] by applying one predefined redundant evolution rule set to design multiple different P systems, each of which executes a computation task. In [7], an automatic design method of a cell-like P system framework for performing five basic arithmetic operations (addition, subtraction, multiplication, division, and power) was presented. In [72], 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, $\frac{1}{2}[n(n-1)]$, $n(n-1)$, $(n-1)^2+2^n+2$, $a^{2^n} b^{3^n}$ and $\frac{1}{2}(3^n-1)$, ($n > 1$ or 2). In [21,31], the automatic design of SN P systems is discussed.

This chapter is organized in the following manner: Sect. 3.2 discusses automatic design of cell-like P systems with P-Lingua, Sect. 3.3 discusses automatic design of spiking neural P systems with P-Lingua, Sect. 3.4 discusses modelling real ecosystems with MeCoSim, and Sect. 3.5 discusses robot motion planning.

3.2 Automatic Design of Cell-Like P Systems with P-Lingua

In this section, some preliminaries are first provided. An automatic design approach with a *genetic algorithm* (GA) for a cell-like P system through tuning membrane structures, initial objects, and evolution rules is discussed. Next, an automatic design method with a permutation penalty genetic algorithm (PPGA) for a deterministic and non-halting membrane system by tuning membrane structures, initial objects, and evolution rules is discussed.

3.2.1 Preliminaries

3.2.1.1 Alphabet and Multisets

An *alphabet* Γ is a non-empty set, and their elements are called *symbols*. A *string* u over Γ is an ordered finite sequence of symbols, that is, a mapping from a natural number $n \in \mathbb{N}$ onto Γ. The number n is called the *length* of the string u, and it is denoted by $|u|$. The empty string (with length 0) is denoted by λ. A *multiset* over an alphabet Γ is a mapping f from Γ onto the set of natural numbers $\mathbb{N}$. For each symbol $a \in \Gamma$, the natural number $f(a)$ is called the *multiplicity* of symbol a in multiset f. We denote by $M(\Gamma)$ the set of all multisets over Γ.

3.2.1.2 Rooted Tree

An *undirected graph* G is an ordered pair (V, E) where V is a set whose elements are called *nodes* and $E = \{\{x, y\} \mid x, y \in V, x \neq y\}$ whose elements are called *edges*. A *path* of length $k \geq 1$ from $x \in V$ to $y \in V$ is a sequence $(x_0, \ldots, x_k)$ such that $x_0 = x$ and $x_k = y$. If $x_0 = x_k$, then we say that the path is a *cycle*. An undirected graph is *connected* if every pair of nodes is connected by a path. An

undirected graph with no cycle is said to be *acyclic*. A *rooted tree* is a connected, acyclic, undirected graph in which one of the vertices (called *the root of the tree*) is distinguished from the others.

3.2.1.3 Cell-Like P System/Transition P System

1. A cell-like P system with a hierarchical membrane structure can be formally represented as $\Pi = (V, O, \mu, W, \Re, i_o)$ [49], where
   (a) V is the (finite and non-empty) alphabet of objects.
   (b) $O \subseteq V$ is the output alphabet, namely, the set of output objects.
   (c) μ 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 μ, 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 μ, 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*: $[u \to v]_i$;
      (ii) *dissolution rule*: $[u]_i \to v$;
      (iii) *rewriting-communication rule*: $[u]_i \to [v]_i x$;
      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. The rewriting rule $[u \to v]_i$ rewrites u by v. The dissolution rule $[u]_i \to v$ 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 $[u]_i \to [v]_i x$ rewrites u by v inside the compartment i and, at the same time, sends x outside the compartment.
   (f) i_o is the output membrane of Π.
2. The system is called noncooperative if the length of the object in the left-hand side of an evolution rule is one.

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 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, that is, 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.

Sometimes cell-like P systems are of the form $\Pi = (V, O, \mu, \mathcal{M}_0, \ldots, \mathcal{M}_{m-1}, (\mathcal{R}_0, \rho_0), \ldots, (\mathcal{R}_{m-1}, \rho_{m-1}), i_o)$, where $\mathcal{M}_0, \ldots, \mathcal{M}_{m-1}$ are multisets over O; $\mathcal{R}_0, \ldots, \mathcal{R}_{m-1}$ are rules associated with membrane $0, 1, \ldots, m-1$, respectively; and $\rho_0, \ldots, \rho_{m-1}$ are the partial order relations associated with the rules in $\mathcal{R}_0, \ldots, \mathcal{R}_{m-1}$. The ρ_i provides priorities between rules in $\mathcal{R}_i$, in such a manner that if $(r_1, r_2) \in \rho_i$ we say that rule r_1 has a higher priority than r_2, and we denote it by $r_1 > r_2$;

3.2.2 Automatic Design of P Systems with an Elitist Genetic Algorithm

In this subsection, *automatic design approach for a cell-like P system* through tuning membrane structures, initial objects, and evolution rules is discussed. In this method, a binary encoding technique is used to codify the P system with variable membrane structures, initial objects, and evolution rules; an elitist genetic algorithm is applied to evolve a population of P systems toward a successful P system for fulfilling a specific task, the calculation of the square of 4 [47]; an effective fitness function is employed to evaluate each candidate P system by a using *P-Lingua simulator* [58].

3.2.2.1 Problem Statement

Automatic design of cell-like P systems through tuning membrane structures, initial objects, and evolution rules [47] is performed by advancing the design of a cell-like P system step-by-step. While performing the task, a family of membrane systems $\prod$ of P systems, that is, $\prod = \{\Pi_i\}_{i \subseteq N}$, where N is the set of natural numbers and each P system Π_i has the structure $\Pi_i = (V, \mu, W, R, i_o)$, where V is a predefined alphabet of objects; μ is a hierarchical membrane structure with m membranes labeled by the elements of a given set H, $H = \{0, 1, \ldots, m-1\}$, and the skin membrane is labeled as 0; W is the set of initial multisets $w_0, \ldots, w_{m-1}$ over V associated with the regions $0, 1, \ldots, m-1$ of μ, that is, $W = \{w_0, \ldots, w_{m-1}\}$; R is the set of evolution rule sets $R_0, \ldots, R_{m-1}$ associated with the regions $0, 1, \ldots, m-1$ of μ, that is, $R = \{R_0, \ldots, R_{m-1}\}$. In order to perform the task of automatic design, μ, W, and R need to be designed. Moreover, only the rewriting and dissolution rules are considered in this design, and $i_0 = 0$ implies that the output result is inside the skin membrane.

Considering a family $\prod$ of P systems, $\prod = \{\Pi_i\}_{i \subseteq N}$, where each P system Π_i has a variable μ, W, and R, where μ, W, and R are attained by using an optimization approach. W and R coming from the alphabet V are generated in the process of design.

3.2.2.2 Design Method

The general steps of the design method can be summarized as follows:

Step 1 Design of membrane structure μ: A hierarchical membrane structure with m membranes is considered in the cell-like P systems.

Step 2 Definition of an alphabet V: As usual, a certain number of letters from English alphabet is chosen so that it satisfies the requirement of initial objects w_i and evolution rules R_i.

Step 3 Design of evolution rule set R: The evolution rule set R is obtained by using a genetic algorithm, where the maximal number of evolution rules in R_i, that is, the length of R_i, and the types of evolution rules are considered.

Step 4 Design of initial object set W: The initial objects inside each membrane w_i ($i = 0, 1, \ldots, m-1$) are obtained by using a genetic algorithm, where the maximal number of initial objects inside each membrane w_i ($i = 0, 1, \ldots, m-1$) need to be prescribed.

Step 5 Design of a genetic algorithm: This step has to consider four points: (1) an encoding technique for membrane structure μ, evolution rule set R or $\Re$, and initial object set W; (2) a fitness function for evaluating a candidate P system; (3) the choices of selection, crossover, and mutation operators; and (4) parameter setting.

The details are described as follows:

(1) The encoding techniques for initial object set W, evolution rule set $\Re$, and membrane structure μ are as follows.
   (a) *Encoding W*: In a cell-like P system, encoding of an object in V is introduced and then turn to the representation of each initial object set w_i ($i = 0, 1, \ldots, m-1$) and the initial object set W. Suppose the alphabet $V = \{a_0, a_1, \ldots, a_{n_o-1}\}$, where n_o is the number of objects in V. In the genetic algorithm, a binary string with n ($n = ceil(\log_2 n_o)$ bits (0 or 1) is used to represent the object a_j ($j = 0, 1, \ldots, n_o - 1$), where the *ceil* function returns the smallest integer value that is greater than or equal to the number $\log_2 n_o$. In this representation of each object, if the number of objects in V is less than 2^n, then $(2^n - n_o)$ copies of the empty set λ are inserted into V. For example, if $V = \{a, b, c, z_1, z_2, z_3, z_4\}$, then the binary string with 3 bits is used to represent each object in V. Thus, the codes corresponding to each object can be listed as follows:

$$000 \to a, 001 \to b, 010 \to c, 011 \to z_1$$
$$100 \to z_2, 101 \to z_3, 110 \to z_4, 111 \to \lambda.$$

   Thus, the initial object set W can be encoded, $W = \{w_1, \ldots, w_m\}$, where w_i ($i = 0, 1, \ldots, m-1$) is the initial object set in the ith membrane and is composed of a certain number of copies of each object in V. So the maximal number is limited, denoted by n_{w_i} ($i = 0, 1, \ldots, m-1$) of copies of objects for each w_i in the design of a P system. Therefore, the initial object set w_i in the ith membrane can be represented by using a binary string with nn_{w_i} bits, and consequently, W is denoted by using a binary string with L_W bits, where

$$L_W = \sum_{i=0}^{m-1} nn_{w_i}. \tag{3.1}$$

That is, W is obtained by concatenating w_i ($i = 0, 1, \ldots, m-1$) one by one. For example, if $V = \{a, b, c, z_1, z_2, z_3, z_4\}$, W=$\{w_0, w_1\}$, $n_{w_0} = 2$, and $n_{w_1} = 2$, W is represented by applying 12 bits. W=000000011111 means that $w_0 = a^2$ and $w_1 = z_1$.

(b) *Encoding* $\Re$: Considers two types of evolution rules, that is,

$$[leftObSet \rightarrow rightObSet]_{label}$$
$$[leftObSet]_{label} \rightarrow rightObSet$$

where $leftObSet$ and $rightObSet$ are the multisets of objects selected from the alphabet V; $label$ represents the label of a membrane, that is, the location of the evolution rule. The value of $label$ needs to be preset. The first rule is a transition rule, and the second one is a dissolution rule. So a multi-tuple ($leftObSet$, $rightObSet$, $dissolution$) is used to represent an evolution rule, where $dissolution$ is a binary bit, that is, '0' or '1', where '0' and '1' representing the rule will be dissolved or not. The encoding methods of $leftObSet$ and $rightObSet$ are the same as in w_i. So the binary string concatenating the three strings $leftObSet$, $rightObSet$, and $dissolution$ represents an evolution rule. For example, if $V = \{a, b, c, z_1, z_2, z_3, z_4\}$, $label = 1$, the length of $leftObSet$ equals 1, and the length of $rigthObSet$ equals 2, an evolution rule can be denoted as a binary string with 13 bits. The string 0000010101111 means $r_1 \equiv [a]_1 \rightarrow b$.

Thus, the evolution rule set R, $R = \{r_1, r_2, \ldots, r_{n_R}\}$ is encoded, where n_R is the number of evolution rules in R. If an evolution rule is represented by using a binary string with L_r bits, the evolution rule set R can be represented as a binary string with n_R L_r. For example, if $R = \{r_1, r_2\}$, where $r_1 \equiv [a]_1 \rightarrow b$ and $r_2 \equiv [a \rightarrow bc]_1$, the evolution rule set R can be denoted as 0000010101111000001010010, that is, $R = \{[a]_1 \rightarrow b, [a \rightarrow bc]_1\}$.

In this case, the membrane structure μ is fixed, and a candidate P system can be represented as the binary string concatenating W and R, that is, the string with $L_W + n_R L_r$ binary bits.

(c) *Encoding* μ: The hierarchical membrane structure of a cell-like P system can be represented as a tree structure, where each of the membranes except for the skin membrane has a parent membrane. For example, if $\mu = \left[[[]_2]_1\right]_0$, the parent membranes of the membranes with the label 2 and 1 are the membranes labeled as 1 and 0, respectively. The skin membrane in a cell-like P system is the outermost membrane. So only the codes of the rest ($m-1$) membranes are considered. In this section, a parent membrane encoding method to represent a membrane structure is introduced. Suppose that a cell-like P system has m membranes labeled as $0, 1, \ldots, m-1$. The symbol n_m ($n_m = ceil(\log_2 m)$ binary bits (0 or 1) is used to encode each of the $m-1$ membranes, where the *ceil* function returns the smallest integer value that is greater than or equal to the number $\log_2 m$. The code of each membrane refers to the label of its parent membrane. Thus, the membrane structure μ can be represented by using a binary string with $(m-1)n_m$ bits. In this representation of a membrane structure, if the

number of membranes in μ is less than 2^{n_m}, the membrane structure with m membranes is extended to the new structure with 2^{n_m} membranes, where the last $(2^{n_m} - m)$membranes are represented by using the empty set λ. For example, if a cell-like P system has four membranes, μ is represented by applying six binary bits. Thus, the string 000010 means $\mu_1 = [[\,]_1[[\,]_3]_2]_0$, and the string 000000 corresponds to $\mu_2 = [[\,]_1[\,]_2[\,]_3]_0$. It is worth noting that this representation of μ may result in unfeasible membrane structures. So the traversal of a tree is applied to check whether a candidate membrane structure is feasible or not. Each of the three traversal approaches, preorder, inorder, and postorder traversal, can effectively solve this problem.

In this case, a candidate P system can be represented as the binary string concatenating the codes of μ, W, and R, that is, the string with $L_W+n_R L_r+(m-1)n_m$ binary bits.

(2) The fitness function with penalty items is used to evaluate a candidate P system. The model of the fitness function is described as follows:

$$fitness = fitness + f(\mu) + f(W) + f(R) + f(Halt) \tag{3.2}$$

where $f(W)$ is the penalty term of the undesired properties in the initial objects set, that is, when W contains redundant objects, an extra punishment is needed to the fitness function; $f(R)$ is the penalty term of the undesired properties in the evolution rules set. Four undesired properties are considered, and they are as follows: (1) The dissolution rule is a structural rule, and it is applied at most once per step. If R contains more than one dissolution rules in one membrane, the simulation will report errors. In order to avoid this kind of errors, a fitness function value with penalty term is directly returned to stop the simulation. (2) If R contains nondeterministic rules, the fitness function value with penalty term is directly returned. (3) A candidate P system contains useless rules: If a candidate P system contains a useless rule, then it will be added to a penalty term with η. (4) A candidate P system contains evolutionary rules which cannot forward the calculation process. $f(Halt)$ is the penalty term of the non-termination property, that is, if a candidate P system is not in a halting configuration, an extra punishment is added to the fitness function. If a candidate P system cannot satisfy the desired properties, a penalty term is added to the fitness function to reduce the probability of the selection in the candidate population. So the genetic algorithm can gradually remove the undesired candidate P systems. $f(\mu)$ is the penalty term of the undesired properties of the membrane structure. If a candidate P system has an invalid membrane structure, it cannot satisfy the basic syntax of the membrane systems. So in order to make error in the process of simulation, a penalty term is returned to stop its simulation in P-Lingua. The evaluation method is shown in Fig. 3.1.

In the evaluation method, the selection of these constants is based on the designers' experience, the experimental results, and some investigations such as in [66].

Evaluation method

$fitness \leftarrow 0$

Load the P system corresponding to the current chromosome

$NondePairs \leftarrow$ number of rule pairs with the same left hand side

$DisRuleNum \leftarrow$ number of dissolution rules in a membrane

$UselessNum \leftarrow$ number of useless initial objects in the process of the simulation

$UselessRule \leftarrow$ number of useless rules in the process of the simulation

$NotEvoRule \leftarrow$ number of not evolutionary rules in the simulation

$UselessMemSructure \leftarrow$ the value represents that the membrane structure is error

if (($NondePairs > 0$)||($UselessMemSructure > 0$)||($DisRuleNum > 1$)) **then**

$fitness \leftarrow \delta * NondePairs + \delta * DisRuleNum + \delta * UselessMemSructure$

return $fitness$

else

{the P system is deterministic so we need to simulate only computation}

$step \leftarrow 0$

while ((P system is not a halting state) $\wedge$ ($step < MaxSteps$))**do**

evolve one step(move to the next configuration of the P system)

$step \leftarrow step+1$

end while

if P system is in a halting configuration **then**

$fitness \leftarrow fitness + |simulation\_result - desired\_result|$

else

$fitness \leftarrow fitness + |simulation\_result - desired\_result| + \eta_1$

end if

if $fitness = 0$ **then**

$fitness \leftarrow fitness + UselessNum * \eta_2 + NotUseRule * \eta_2 + NotEvoRule * \eta_2$

end if

return $fitness$

end if

Fig. 3.1 Evaluation methods. From [47]

(3) In this section, JGAP-Java Genetic Algorithms and Genetic Programming Package [43, 44] are used. The genetic algorithm applies an elitist selection operator, a single-point crossover operator, and a uniform mutation operator.
(4) There are four parameters in the genetic algorithm, and they are represented as a parameter set Pa_set where $Pa\_set = \{N_p, P_c, P_m, IterNum\}$, N_p, P_c, P_m, and $IterNum$ are the population size, the crossover rate, the mutation rate, and the maximal number of evolutionary generations in the genetic algorithm, respectively.

P system $\prod = \{V, \mu, W, R, i_o\}$ is considered, where the membrane structure consists of four membranes, and the skin membrane is labeled 0; the alphabet $V = \{a, b\}$; i_o=0; W=$\{w_0, w_1, w_2, w_3\}$, $R = \{R_0, R_1, R_2, R_3\}$, and μ are obtained by using JGAP. The parameters are assigned as follows: $n_{w_0} = n_{w_1} = n_{w_1} = n_{w_1} = 1$; $n_R = 4$, that is, the evolution rule set R consisting of four rules. In the experiment, each of R_0, R_1, R_2, and R_3 consists of only one rule. The maximal number of objects in the $leftObjSet$ and $leftObjSet$ of each rule are 1 and 4, respectively. According to the referring existing literature [66] and the design rules, the rest of the parameters in the experiments are set as follows: $m = 4$, $n = 2$, $L_W = 8$, $L_R = 44$, $\delta = 25$, $\eta_1 = 1$, $\eta_2 = 1$, and $MaxSteps = 25$.

In the following description, at first, the choices of the mutation rate P_m, the crossover rate P_c, the population size N_p, and the maximal number $IterNum$ of evolutionary generations in the genetic algorithm are discussed, and then the result of this design is provided.

In the experiments, the parameter sets for P_m, P_c, N_p, and $IterNum$ are set as follows:

$Pa\_set_{P_m} = \{30; 0.1; \{0.01, 0.05, 0.1, 0.125, 0.2, 0.25, 0.35, 0.5, 1.0\}; 300\}$ for P_m; $Pa\_set_{P_c} = \{30; \{0.01, 0.05, 0.09, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}; 0.1; 300\}$ for P_c; $Pa\_set_{N_p} = \{\{10, 20, 30, 40, 50, 60, 70\}; 0.1; 0.1; 300\}$ for N_p; $Pa\_set_{IterNum} = \{30; 0.1; 0.1; \{50, 100, 150, 200, 250, 300, 350, 400\}\}$ for $IterNum$.

Experimental results for the discussion of the four parameters are shown in Fig. 3.2, where the successful rate refers to the ratio of the number of successful computations to 100 independent runs; the average generation is the average of the evolutionary generations over 100 independent runs when the algorithm stops for each test, and the total number of function evaluations refers to the total number of the fitness function evaluations for candidate P systems in 100 independent runs.

Figure 3.2a shows that the highest successful rate and the lowest average generation are obtained when the value of P_m equals 0.100. Figure 3.2b shows that the best results of the successful rate and the average generation are obtained when $P_c = 1.0$. In Fig. 3.2c, when the value of N_p is greater than 30, the success rate arrives at 100%, while the best result of the total number of function evaluations is achieved when N_p=30. According to the results in Fig. 3.2d, as $IterNum$ increases from 50 to 400, the elapsed time per run gradually goes up; on the other hand, the success rate arrives at 100% if $IterNum$ is equal to or greater than 300. Therefore, it is better to assign P_m, P_c, N_p and $IterNum$ as 0.100, 1.0, 30, and 300, respectively.

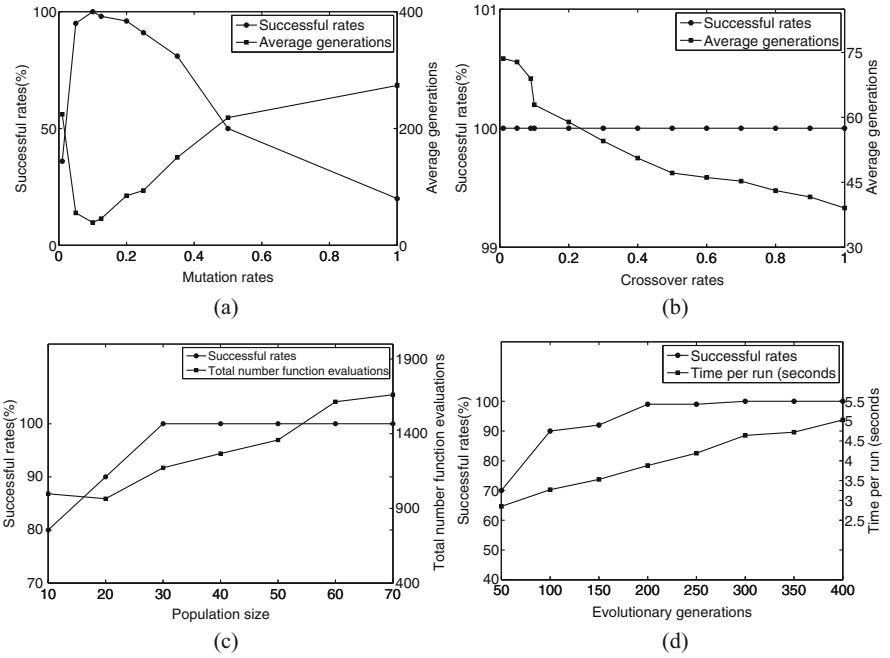

Fig. 3.2 Experimental results for the four parameters. From [47]. (**a**) Mutation rates. (**b**) Crossover rates. (**c**) Population sizes. (**d**) Evolutionary generations

In the design of a cell-like P system for calculating the square of 4, $Pa\_set = \{30; 0.1; 1.0; 300\}$. Next 1000 independent runs are performed, and it obtains 100% success rate. The introduced design approach obtains 65 different variants of cell-like P systems for successfully fulfilling the computation of 4^2. Their details can be referred to [47]. The design method obtains different solutions having different initial objects and evolution rules sets, due to the randomness of the selection of objects, the rules, and the membrane structure.

3.2.3 Automatic Design of P Systems with a Permutation Penalty Genetic Algorithm

In this section, an automatic design method, that is, *permutation penalty genetic algorithm* (PPGA), for a deterministic and non-halting membrane system generating the set $\{n^2 | n \geq 1\}$ of natural numbers, by tuning the syntactical ingredients consisting of membrane structures μ, initial objects W, and evolution rules $\Re$ [74] is discussed. The design approach is described in detail. And then a cell-like membrane system for computing the square of n (n is a natural number) is presented.

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, 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 are discussed. At first, three techniques are presented, and then the design method to provide an algorithmic elaboration is summarized.

1. Representation of P Systems

The permutation encoding technique [60] is used to codify a P system. The representation of a P system consists of the encoding approaches for the alphabet V, its membrane structure μ, the initial multiset vector W, evolution rules set $\Re$, and an individual chromosome corresponding to a candidate P system. Next, these approaches are discussed one by one.

(a) Encoding of V

Suppose that there are N_V objects (letters), and the N_V strictly positive integers are used to represent the objects and 0 to denote the empty set λ. 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".

(b) Encoding of μ

The hierarchical membrane structure of a cell-like P system can also be denoted as a rooted tree. Thus, the label of the parent (the neighboring outer membrane, like the parent of a node in a tree) of each membrane can be used 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_μ membranes is represented with a string with $(N_\mu - 1)$ numbers. For example, the structure in Fig. 3.3 can be represented as the codes "0001136".

(c) 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, ... or N_V. The codes of W can be obtained by concatenating

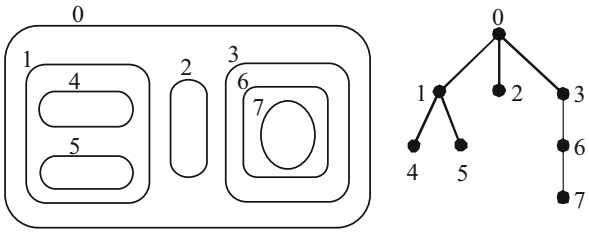

Fig. 3.3 An example for a cell-like P system membrane structure and its associated tree. From [60, 74]

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

$$L_W = \sum_{i=0}^{m-1} N_{w_i} + m - 1 \tag{3.3}$$

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

(d) 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, the set $\Re$ is encoded. 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, N_l codes are used, 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 $N_V + 3$ to denote a rewriting, dissolution, and rewriting-communication rules, respectively) to encode a rule. Thus, the code length L_{R_i} for the rule is $N_l + N_r + 1$, that is, 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

$$L_\Re = \sum_{i=0}^{m-1} (N_{R_i} * L_{R_i}) + m - 1 \tag{3.4}$$

It is worth noting that the dissolution rule is a structural rule, which is applied at most once 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, the set $\Re$=$\{R_0, R_1, R_2\}$ is encoded as the string 11124233481334221681235" where $R_0 = \{[a \to aab], [b \to cc]\}$, $R_1 = \{[a \to cc], [b] \to [b]a\}$, and $R_2 = \{[a] \to bc\}$.

(e) Encoding of a P System

Next, a P system through tuning membrane structure, initial objects, and evolution rules is designed where the codes for the P system can be attained by sequentially concatenating the codes of μ, W, and $\Re$, and a separator symbol $N_V + 6$ to enable the separation of the codes of μ, W, and $\Re$. The encoding of a P system

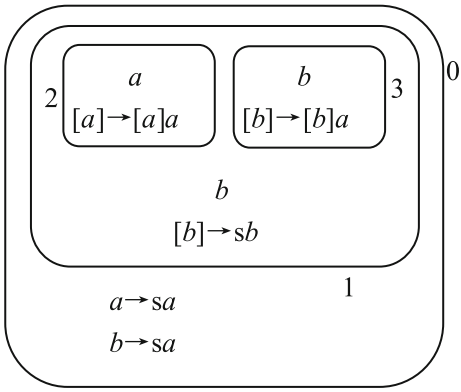

Fig. 3.4 The initial configuration of Π_e (with rules included). From [74]

has been illustrated using 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=λ, $w_1 = b$, $w_2 = a$, $w_3 = b$;
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 Π_e is illustrated in Fig. 3.4. 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 Π_e is encoded as the string "011903239212431248313582226833 26".

2. 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, the following seven aspects are considered:

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 [25, 26], 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 μ. 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.
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. They are called *redundant objects*. This redundancy results from the randomness of the generation of the population of initial P systems in an evolutionary algorithm.
4. The nondeterminism of a P system resulting from nondeterministic 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

Based on the above analysis, the following evaluation functions are defined:

$$f = f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 \tag{3.5}$$

where

$$f_1 = g_1(N_s) = \sum_{i=1}^{N_{obj}} |N_i^{ao} - N_i^{eo}| \tag{3.6}$$

$$f_2 = g_2(\mu) = \delta \cdot M_f \tag{3.7}$$

$$f_3 = g_3(W) = \eta \cdot N_{obs} \tag{3.8}$$

$$f_4 = g_4(\Re) = \alpha \cdot N_{non} \tag{3.9}$$

$$f_5 = g_5(\Re) = \beta \cdot R_{dis} \tag{3.10}$$

$$f_6 = g_6(\Re) = \gamma \cdot N_{red} \tag{3.11}$$

$$f_7 = g_7(\Re) = \xi \cdot H \tag{3.12}$$

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 ith $(i = 1, 2, \ldots, N_{obj})$ output objects, respectively; $N_{obj} = |O|$; and 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 μ, δ is a penalty factor, and $M_f \in \{0, 1\}$, 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, η is a penalty factor, and N_{obs} is the number of the redundant objects in the initial multiset vector W.
- f_4 is the penalty item of a nondeterministic P system, $g_4(\Re)$ is a function of the set $\Re$, α is a penalty factor, and $N_{non} \in \{0, 1\}$, where "0" and "1" mean that there is not any nondeterministic evolution rule, and there is at least one pair of nondeterministic 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 $\Re$, β is a penalty factor, and $R_{dis} \in \{0, 1\}$, 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 $\Re$, γ is a penalty factor, and 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 $\Re$, ξ is a penalty factor, and $H \in \{0, 1\}$, where "0" and "1" mean that the candidate P system is a non-halting and halting one, respectively.

In Eqs. (3.7)–(3.12), the purpose of introduction of the penalty factors $\delta, \alpha, \beta, \gamma$, and ξ is to reject the unexpected candidate P systems, and therefore, the five factors can be assigned as a larger value as possible, for example, δ=α=β=ξ=999999, while the purpose of the use of the two factors η and γ 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. They are empirically set to 1 and 1, respectively.

3. Evolution of P Systems

The genetic algorithm with the permutation encoding technique (GAPE) in JGAP [43] is used to evolve a family of P systems toward a successful one. GAPE uses the elitist selection strategy, where 20% 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.

It is worth noting that the evolutionary operators might produce the P systems violating the constraints in (3.7)–(3.12) including infeasible membrane structures μ, 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 nondeterministic evolution rule pairs, and the halting P system due to evolution rules.

```
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. 3.5 Pseudocode algorithm of PPGA. From [74]

4. Algorithmic Elaboration

This subsection summarizes the design method PPGA as shown in Fig. 3.5, 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, δ, η, α, β, γ, ξ, 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.
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, and (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) Nondeterministic 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 and (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 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 is considered.
5. The one-point crossover operator is used and depicted.
6. The uniform mutation operator is employed and illustrated.

Algorithm 1: Evaluation method

Require: A candidate P system
$f \leftarrow 0$
Compute M_f, N_{obs}, N_{non}, R_{dis}, N_{red}, H
if $((M_f > 0) \| (N_{non} > 0) \| (R_{dis} > 0))$ **then**
  $f \leftarrow f_2 + f_4 + f_5$
else
  $N_s \leftarrow 0$
  while $(H < 1) \wedge (N_s \leq MaxStep)$ **do**
    Evolve the P system for one step
    $N_s \leftarrow N_s + 1$
    $f \leftarrow f + f_1$
  end while
  if $(H > 0)$ **then**
    $f \leftarrow f_7$
  end if
  if $(f = 0)$ **then**
    $f \leftarrow f + f_3 + f_6$
  end if
end if
Ensure: Fitness f

Table 3.1 Successful P systems

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

The design of the cell-like P system Π_{ex} for fulfilling the computation n^2 is discussed to show the results. The parameters, P_m, P_c, N_P, $MaxGen$, η, and γ, are set to 0.1, 0.8, 20, 200, 1, and 1, respectively. Next, 5000 independent runs of the design experiment are performed, and it obtains the success rate 100%. The introduced design approach obtains 2930 different variants of cell-like P systems Π_{ex} for successfully fulfilling the computation of n^2. Table 3.1 lists only five successful P systems. The complete list of the 1936 successful P systems can refer to [74]. Due to the randomness of the selection of membrane structure, objects, and rules, multiple solutions for the same computational task can be obtained on the identical condition to provide multiple possibilities to construct different complex membrane systems.

3.3 Automatic Design of Spiking Neural P Systems with P-Lingua

In this section, an automatic design method based on genetic algorithms for evolving SN P systems for generating natural numbers within P-Lingua [20, 21, 54, 67] is discussed.

An SN P system consists of five main elements: the amount of neurons in the system, the synapse connections between neurons, the amount of rules within each neuron, the regular expressions which define each rule, and the initial number of spikes within each neuron.

A, SN P system [30] of degree $m \geq 1$ is a tuple $\Pi = (O, \sigma_1, \cdots, \sigma_m, syn, i_o)$, where:

(1) $O = \{a\}$ is the singleton alphabet (a is called spike);
(2) $\sigma_1, \cdots, \sigma_m$ are neurons, identified by pairs

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m \tag{3.13}$$

where:
(a) $n_i \geq 0$ is the initial number of spikes contained in σ_i.
(b) R_i is a finite set of rules of the following two forms:
(i) $E/a^c \rightarrow a; d$ where E is a regular expression over O, and $c \geq 1, d \geq 0$;
(ii) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (i) from R_i, we have $a^s \notin L(E)$;
(3) $syn \subseteq \{1, \ldots m\} \times \{1, \ldots m\}$ with $(i, i) \notin syn$ for $i \in \{1, \ldots m\}$ (synapses between neurons);
(4) $i \in \{1, \ldots m\}$ indicates the output neuron (i.e., σ_{io} is the output neuron).

The firing and forgetting rules of an SN P system are described and discussed in detail in [30, 73]. The distinguishing feature of SN P system is that the sequence of configurations can produce an associated spike train. If the output neuron spikes, then we have 1, and otherwise, we have 0. Hence, the spike train can be represented by the sequence of ones and zeros.

In order to automatically generate an SN P system, we should consider each aspect in an SN P system. The number of neurons in system, the synapse connections between neurons, the number of rules within each neurons, and the number of spikes within each neuron, according to specific task, are previously determined, but the regular expressions which define each rule and the delays on each rule are randomly generated in an SN P system. Then we can generate a population of SN P systems by same method. The aim is to use genetic algorithms to get an optimal SN P system by appropriately evolving an SN P system. The steps are listed as follows:

Step 1: First of all, we define a population of SN P systems $\Pi = \{\Pi_i\}_{i \in H}$, where H is a subset of natural numbers, and each SN P system Π_i of degree $m \geq 1$ is

described as follows:

$$\Pi_i = (O, \sigma_1, \cdots, \sigma_m, syn, i_o) \tag{3.14}$$

where

(1) $O = \{a\}$ is a predefined singleton alphabet;
(2) $\sigma_1, \cdots, \sigma_m$ is the neurons from 1 to m.

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m \tag{3.15}$$

where:
(a) $n_i \geq 0$ is the initial number of spikes contained in σ_i.
(b) R_i is a finite set of rules of the following two forms:
(i) **Spike transfer rules:** $E/a^c \rightarrow a; d$. When fulfilling spike transfer rules and $d = 0$, a spike in the neuron should leave along the synapses and travel to the neurons connected to the neuron where the rule is applied.
(ii) **Spike forgetting rules:** $a^s \rightarrow \lambda$. When performing spike forgetting rules, s spikes are consumed.

Step 2: Determine fitness of each individual in the population.
Step 3: Reserve the individual with higher fitness from the population.
Step 4: Select parents from the population and produce offsprings.
Step 5: Randomly perform mutation.
Step 6: Check whether any individual meets the requirements. If so, terminate the algorithm; otherwise, continue the algorithm.

The pseudocode algorithm of automatic design method is shown in Fig. 3.6.
More explanations for each step are provided as follows:

Step 1: Input required parameters, which include m, n_i, syn, i_o, H, $MaxSteps$, $StepRepetition$, $MutationRate$, $MinFitness$, $MaxGeneration$, $BestFitness$, and $ExpectedSet$,
where:

(a) m, n_i, syn, and i_o represent the number of neurons in each P system, the number of spikes in each neuron, the synapse connections between each neuron, and the output neurons, respectively.
(b) H is population size.
(c) $MaxSteps$ represents the maximum steps that each network will take.
(d) $StepRepetition$ is the amount of repetitions each network will undergo to generate an output list.
(e) $MutationRate$ is the percentage chance for mutation.
(f) $MinFitness$ represents minimal fitness.
(g) $MaxGeneration$ is the max amount of generations.

Require: Initial membrane construction and objects and genetic algorithm
i=1
while ($i \leq H$) **do**
  Generating random $SNPS_i$
  Caculating fitness value F($SNPS_i$)
  if ($F(SNPS_i) \leq MinFitness || F(SNPS_i) == null$) **then**
    Generating new $SNPS_i$ and replacing old $SNPS_i$
  end if
  $i = i + 1$
end while
while ($generation \leq MaxGeneration$) **do**
  Caculating fitness value each SNPS
  Sorting population accordding to set F($SNPS$)
  i=1
  while ($i \leq H$) **do**
    if ($i \leq Elitism$ && $i \leq H$) **then**
      $Newpopulation[i] = Population[i]$
      if (F($SNPS_i > BestFitness$)) **then**
        $BestFitness = SNPS_i$
      end if
    else
      Parent1=ChooseParent( )
      Parent2=ChooseParent( )
      Child=Crossover(Parent1,Parent2)
      Child=Mutate(Child)
      $Newpopulation[i] = Child$
    end if
    if ($F(SNPS_i) == 0 || F(SNPS_i) == null$) **then**
      $F(SNPS_i) = 0$
    else
      $F(SNPS_i) = F(SNPS_i)$
    end if
  end while
  $genration = gneration + 1$
end while
Ensure: Spiking neural P system

Fig. 3.6 Automatic design algorithm of SN P systems

(h) $BestFitness$ represents the best fitness through generations.
(i) $ExpectedSet$ is the expected set.

Step 2: A population of SN P systems and their fitness values are calculated. $F(SNPS_i)$ and $F(SNPS)$ represent the fitness value of the ith SN P system and the fitness set of all SN P systems in the population, respectively. Check whether SN P systems are correct according to the fitness function value of each SN P system in the population.

Step 3: The genetic algorithm is used to automatically design each SN P system in the population. $Elitism$ represents the number of reserving a certain number of better SN P systems in the population. $Parent1$ and $Parent2$ are two randomly selected SN P system with larger fitness values. $Crossover()$ and $Mutate()$ represent the crossover and mutate functions, respectively.

Step 4: Output a new SN P system with high sensitivity and precision after completion of automatic design.

The most important three steps in Fig. 3.6 including building a population of SN P systems, designing a fitness function, and setting elitism, crossover, and mutation are detailed in the following description.

1. Building a Population of SN P Systems
An SN P system includes the number of neurons, the synapse connections between neurons, the number of rules within each neuron, the regular expressions which define each rule, and the number of spikes in each neuron. An SN P system represents an individual (DNA, $SNPS_i$) in the population. Here, an individual is also thought of as a set, which contains above five aspects. As a result, the building of a population of SN P systems can be divided into the following steps.

Step 1: Generate a random individual, where rules are randomly generated and other elements are predefined.
Step 2: Repeat the first step until all the individuals($SNPS_i$) in the population are produced.
Step 3: Check whether each individual is correct.
Step 4: Delete and replace individuals with incorrect and low fitness values.
Step 5: Save the initial population.

With the initial population, it is necessary to have an appropriate evaluation function to guide the population to evolve to the optimal solution. Therefore, it is worth noting that the fitness function plays an important role throughout the automatic design process. We describe the details of the fitness function as follows.

2. Design of Fitness Function
Here, we discuss how to design the fitness function, which is used to calculate the sensitivity and the precision of SN P systems. There are two data sets after the establishment of the SN P systems. One is a real output set $OutputSet$. Another is given expected set $ExpectedSet$. $OutputSet$ represents generating number set of repeating execution of SN P systems for a specifical task. $ExpectedSet$ is expected number set for a special task. So a fitness function is established by comparing elements in the real output set and the expected set. The pseudocode of the fitness function is shown in Fig. 3.7.

The category of an element in the above two sets is as follows:

(1) The output set is compared with the expected set, and for every number that is in both of the sets, the true positive count tp increases.
(2) The output set is compared with the expected set, and for every number that is in the output set but not in the target set, the false positive count fp increases.
(3) The output set is compared with the expected set, and for every number that is not in the output set but is in the target set, the false negative count fn increases.
(4) The true negative values, those that are not in the output set and not in the target set, are not counted as they are not needed for this design.

3. Elitism, Crossover, and Mutations
An individual consists of genes, which in the case of this section are represented by an SN P system. The crossover function allows the exchange of genes between two

Require: $OutputSet, ExpectedSet, tp = 0, fp = 0, fn = 0$
1: Initialization settings
2: Merging elements from $OutputSet$ and $ExpectedSet$ into $OutExSet$. The length of $OutExSet$ is n
3: $i = 1$
4: **while** $(i \leq H)$ **do**
5: $i = i + 1$
6: **if** $OutExSet(i) \in OutputSet$ **then**
7: **if** $OutExSet(i) \in ExpectedSet$ **then**
8: $tp = tp + 1$
9: Turn to **Step 21**
10: **else**
11: $fp = fp + 1$
12: Turn to **Step 21**
13: **end if**
14: **else**
15: **if** $OutExSet(i) \in ExpectedSet$ **then**
16: $fn = fn + 1$
17: Turn to **Step 21**
18: **else**
19: Turn to **Step 21**
20: **end if**
21: **end if**
22: **if** $i \geqslant n$ **then**
23: Turn to **Step 26**
24: **else**
25: Turn to **Step 4**
26: **end if**
27: $Fitness = (\frac{2 \times tp}{2 \times tp + fp + fn}) \times sf$
28: **end while**
Ensure: Return $Fitness$

Fig. 3.7 The design of the fitness function

parents, creating a new child individual with the characteristics of the parents that were used. After the crossover, there is also a chance for the new child individual to mutate, changing one of the rules in the generated network at random. To ensure diversity in the population, a certain number of individuals are added to the population pool at each generation.

Except for crossover and mutation, this algorithm also allows the use of elitism selection. This feature allows a selected number of best SN P systems to be introduced with a new generation.

The detailed procedure of elitism, crossover, and mutation are described as follows:

Elitism: Elitism, the best optimal individuals in the current population, is set to 1 in the method of the automatic design, $thatis$, an SN P system with the high sensitivity and precision can be saved to new population of each generation.

Crossover: The crossover is mainly composed of two steps, one is to choose the parent individuals (parents with a higher fitness will have a higher chance of reproducing), and the other is to exchange the corresponding rules in the two parent individuals.

Require: $GlobleBestFitness = 0$, $CurrentBestFitness$, $RateChange = 0$, $MutationRate = 0$
1: $i = 1$
2: **while** $(i \leq H)$ **do**
3: $i = i + 1$
4: **if** $GlobleBestFitness \leq CurrentBestFitness$ **then**
5: $GlobleBestFitness = CurrentBestFitness$
6: $RateChange++$
7: **else**
8: $RateChange = 0$
9: **end if**
10: **if** $RateChange \geq 10$ **then**
11: $MutationRate = random(0, 10)$
12: **else**
13: $MutationRate = random(10, 20)$
14: **end if**
15: **end while**
Ensure: Return $MutationRate$

Fig. 3.8 Dynamic adjustment procedure of mutation probability

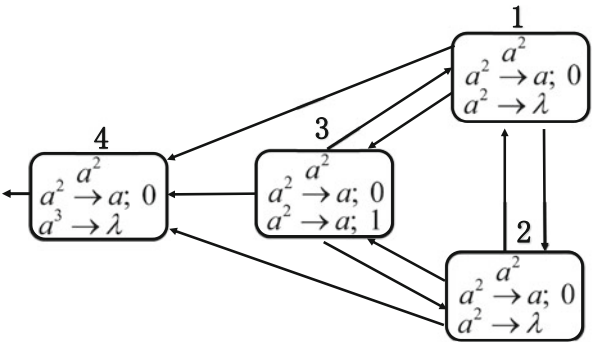

Fig. 3.9 A SN P system generating all natural numbers

Mutations: After getting new sub-individuals from the crossover of two parent individuals, new sub-individuals are mutated and added to new population, where $MutationRate$ is dynamically adjusted according to the detailed problem. The pseudocode algorithm of dynamic adjustment is described in Fig. 3.8.

The automatic design method is further expounded by considering an SN P system generating all even natural numbers.

An SN P system generating all natural numbers mainly contains four elements: four neurons, ten synapse connections between neurons, eight rules, and two starting spikes each neuron. Out of four neurons, three neurons are general neurons, and remaining one is an output neuron. The specific sketch of an SN P system generating all natural numbers is shown in Fig. 3.9.

To illustrate the performance of the design method when simulating an SN P system generating all natural numbers, we make a dynamic behavior analysis from the fitness function value of the experimental testing process.

The average fitness value across ten runs is denoted by F_{av}. A larger value of F_{av} represents a smaller difference between the expected set and the output set.

$$F_{av} = \sum_{j=1}^{10} \sum_{i=1}^{n} F(SNPS_i) \tag{3.16}$$

where $F(SNPS_i)$ represents the fitness value of the ith SN P systems, n is the number of SN P systems in the population, and j represents the jth run.

In the process of simulated evolution, the design parameters are set as follows: expected output set for the natural numbers system, 1, 2, 3, 4, 5, 6, 7, 8, 9; the population size, 4; maximum number of steps per system, 50; maximum number of repeats per system, 50; and maximum number of generations, 200.

We obtain the change curves of the average fitness value of static and dynamic mutation probabilities in Fig. 3.10, respectively. As can be seen from Fig. 3.11, the results of the correct natural data output are produced by a real natural SNP system and is the same as the expected set.

3.4 Modelling Real Ecosystems with MeCoSim

Membrane computing was not conceived in 1998 as a computational modelling framework for complex systems. It was far from the initial studies proposing a novel bioinspired computing model, with roots in formal languages theory and computation theory. The computational power and efficiency of these devices were studied in order to provide alternative paths to traditional computers based on Turing machines, proving the universality of different types of P systems. Besides, these new machines presented desirable properties in terms of the inherent parallelism and the promising effects derived from mechanisms as the cellular division, doubling the computation resources at any given step, trading space for time in order for these systems to present a great ability to solve NP-complete problems in reasonable time. Therefore, nothing in this new paradigm made its founder imagine that the research lines opened could diverge so significantly as it started to happen a few years after the first technical report published in 1998.

3.4.1 Problem Description

As a general idea, the primary intent we pursue is problem-solving through membrane systems. Thus, given a certain abstract problem (as `3-COL` problem, deciding if a coloring with three colors is possible for a given graph), a membrane system is designed to solve the problem, according to the rules satisfying the constraint of the specific computing model chosen among all the possible types and variants of P systems. Then, the design is translated into a P-Lingua format specification and saved in a file with *.pli* extension. For instance, if the solution is

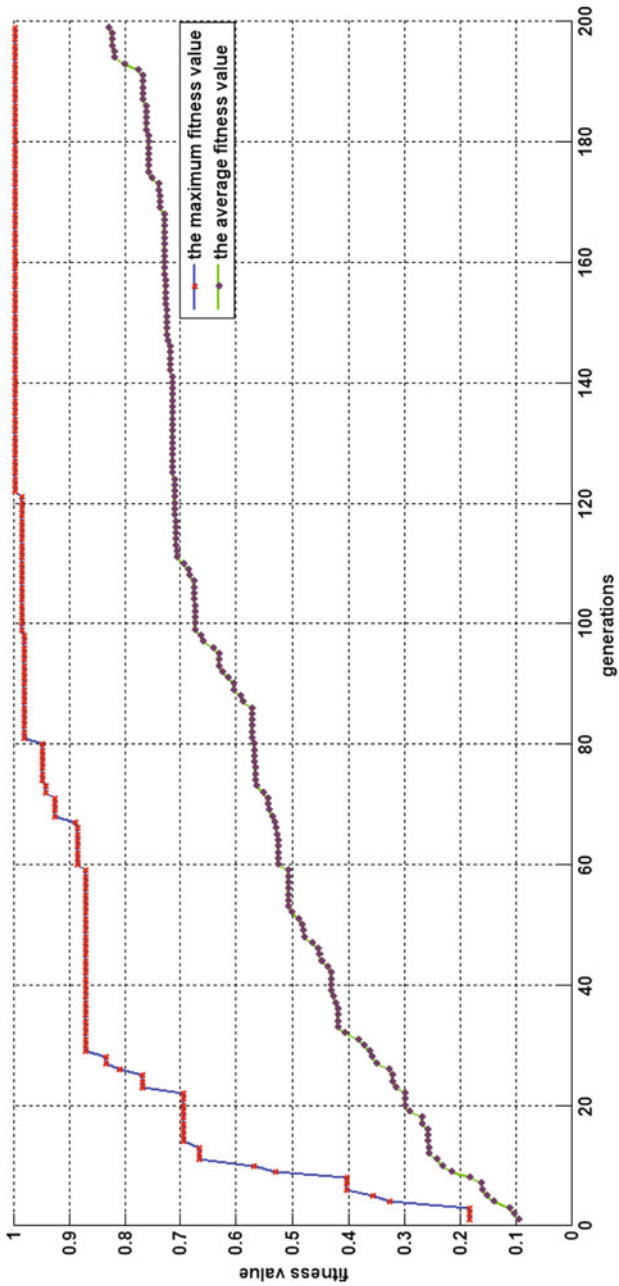

Fig. 3.10 The change curves of the average fitness values and the maximum fitness values

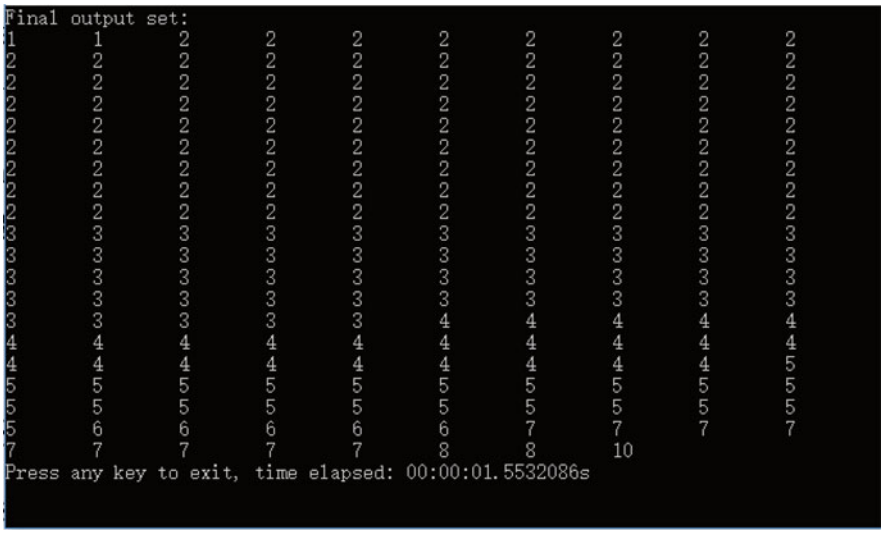

Fig. 3.11 The output set of SN P systems generating all natural numbers

using a cell-like P system with active membranes, the following elements should be given: the membrane structure, μ, the initial multisets for each region, and their corresponding sets of rules.

A file as the one described above could specify a concrete P system, but in our approach, we generally provide solutions for an abstract problem, through a family of P systems (let us denote it by $\Pi(< pars >)$), subject to certain parameters. Thus, for instance, if a solution $\Pi(< n >)$ for 3-COL problem is provided (with the parameter n representing the number of nodes in the graph, getting possibly involved in the sets of rules, alphabet, initial multisets, and membrane structure), the P-Lingua file will generically define the structure applicable to every P system member of the family. Thus, for each particular value of the parameters involved (n in the example), a different P system will be instantiated. Then, for a given member of the family (e.g., the P system $\Pi(5)$, solving 3-COL with $n = 5$, i.e., graphs with five nodes), many different possible inputs could be provided (in the example, one for each possible graph with five nodes, determined by its specific nodes and edges). In our approach, P-Lingua files should specify the solution for the abstract problem, while the introduction of the specific parameter values to instantiate the member of the family and the input to accompany the problem with the particular input is performed through MeCoSim layer. This imposes a clear separation between the abstract problem solved (P-Lingua specification) and the virtual experimentation with each specific instance and input, constituting what is called a scenario (MeCoSim [42, 55]).

For sure, P-Lingua files accept solutions providing non-parameterized models, or models whose parameters are also hard-coded in the same files, but this would

break the proper separation of the responsibilities of each tool and consequently the adequate separation of the roles involved (P system designer, solving the general problem, and end users, running their virtual experiments through the provision of the particular data of each scenario of interest). The approach proposed is even more interesting when we are dealing with models representing real-life problems, for instance, ecology, where the profiles of people in charge of P system design and those managing the actual ecosystem are more clearly identified, in such a way that their background, knowledge, and main focus are generally different, being the former ones interested in designing solutions based on their computing models, taking advantage of their novel theories, and the latter ones interested in managing their ecosystems, getting abstracted from internal details of the models once they have experimentally validated with the experts in the models that they seem to behave properly according to their well-known scenarios, and therefore, the tools provided are useful for their virtual experiments aiding them in their decision-making process to manage their populations.

All in all, MeCoSim raised with this twofold intention now clearer with the roles described above: (1) providing a high-level visual environment to design, debug, simulate, analyze, and visualize models based on P systems and (2) putting at disposal a simulation environment for end users to introduce different instances of the problems and run their experiments according to their final needs.

The needs leading the such development were first detected in the context of ecological modelling and simulation of certain real ecosystems, as the ones described in Sect. 3.4. The managers of the ecosystems under study needed some tools to predict the evolution of the population of certain species in the corresponding systems (for different purposes, such as endangered species conservation or invasive species control). Initially, certain software applications were developed (`Ecosim 1.0` family (see Refs. [20, 54]) to allow the introduction of different initial scenarios and visualize specific outputs, showing certain elements of the ecosystems. The effort was significant to develop each of these specific-purpose applications handling models of different ecosystems (in tasks from the analysis and design to the pure development).

Then, after those experiences, a number of common needs were identified. Investing that amount of effort for each possible future model designed was unfeasible, so a new approach emerged requiring the development of a software environment providing the generic mechanisms not only to handle P system-based models but also to allow the guided delivery of custom applications for each model designed, hence adapted for each end user problem addressed. With this view, `MeCoSim` environment would act as a meta-simulation app, allowing the definition of a customized simulation app for each problem, with the specific inputs and outputs required by that problem. Just to summarize, the definition of such custom user interfaces would imply the following mechanisms:

1. Definition of *input tables* where the user could introduce the external data of each particular scenario (possibly including both data involved in parameter values generation and input data for the specific instance for the experiment to conduct)

2. Calculation of *parameters* and inputs of the P system, from the data of the instance just introduced
3. Definition of the *output tables and charts*, to show the end users requested information for their target application (depending on their user view, but designed by the experts in the underlying P systems, configuring which specific elements of the computation should be extracted and how)
4. Specification of the arrangement of all these inputs and outputs in the custom app.

Further details of this approach are provided in Refs. [42, 55, 67]

Among all the possible practical applications emerged along the years in membrane computing, probably one of the most successful ones is the provision of a very useful methodology [13] for ecologists interested in certain problems related with real ecosystems, involving a number of species, in competition or cooperation, with a number of processes taking place simultaneously in the ecosystem and many biotic and abiotic factors and parameters playing different roles in the interactions.

In what follows, we list in Tables 3.2, 3.3, and 3.4 the main models created following the approach proposed of P systems as modelling framework and P-Lingua and MeCoSim as the tools for virtual experimentation.

3.5 Robot Motion Planning

The problem of motion planning is a crucially important problem in mobile robotics. The problem consists of finding a sequence of motion commands to move a robot in a complex environment from a starting point to a goal area while avoiding static and dynamic obstacles. The problem is even more complex if kinematic and nonholonomic constraints are considered. This problem has been studied from several years ago [35, 59], proving it is PSPACE-hard when the positions of obstacles are known. Several approximate algorithms have been proposed in the literature [19, 64]. A special mention should be given to a category of algorithms to build rapidly exploring random trees (RRTs) [36]. They are based on the randomized exploration of the configuration space by building a tree where nodes represent reachable points in the configuration space, and edges represent the corresponding transitions. In particular, the RRT* algorithm [33] is able to build an RRT whose paths asymptotically converge in time of computation to optimal solutions with respect to a predefined cost function. One of the main challenges by applying motion planning algorithms in robotics is the parallelization in software or hardware of such algorithms in order to accelerate them. For example, in [2], a GPU-based version of the RRT algorithm is presented. One alternative is to model the algorithms over an inherently parallel model of computation and then apply software/hardware simulators. With this idea, membrane computing has been used to design bioinspired parallel RRT models that can be efficiently simulated by means of parallel software/hardware architectures such as OpenMP [46] and CUDA [45]. The first approximation was introduced in [56] by using an extension

Table 3.2 Ecosystems models based on P systems, simulated with EcoSim/MeCoSim (I)

Reference	Case study	Comments
M. Cardona et al. 2008 [4]	**Bearded Vulture** The cliff-nesting and territorial mountains in Catalan Pyrenees (Northeastern Spain)	Five wild and domestic ungulates are included as carrion (prey) species.
M. Cardona et al. 2008 [3]	**Bearded Vulture** Catalan Pyrenees(NE)	Similar structure to [4]
M. Cardona et al. 2010 [5]	**Scavenger Birds** Catalan Pyrenees(NE)	Nomadic and non-nomadic species, and density regulation. Thirteen species, including two scavenger birds in competition with bearded vulture
M.A. Colomer et al. 2010 [10]	**Pyrenean Chamois** Catalan Pyrenees(NE)	Four influencing factors: introduced disease such as pestivirus infection, climate change, hunting, and migrations among areas
M.A. Colomer et al. 2010 [8]	**Bearded Vulture** The cliff-nesting and territorial mountains in Catalan Pyrenees (NE Spain)	Same model presented in [10]
M. Cardona et al. 2011 [6]	**Scavengers/Zebra mussel** Catalan Pyrenees (NE Spain) and a fluvial reservoir (Riba-roja-Ebro river, NE Spain)	For the scavengers, a simplified version of [4]. For mussels, focus on temperature and its effect on reproduction, fixation of the mussel to the substrate, movement of larvae, and density regulations.
M.A. Colomer et al. 2011 [11]	**Scavenger Birds** Catalan Pyrenees /Pyrenean and Pre-pyrenean mountains.	Species move among areas if lack of feeding resources in origin region. The model studied: (a) 13 species, including three avian scavengers (predators), six wild ungulates, and four domestic ungulates (preys); (b) interactions among species; (c) communication among areas; (d) load capacity regulation
M.A. Colomer et al. 2011 [9]	**Plant Communities** (sub)Alpine(NE Spain)	Model with climatic variability and orographic factors. Impact of the plant community module on population dynamics
A. Margalida et al. 2011 [39]	**Scavenger Birds** Catalan Pyrenees(NE)	Wild ungulates considered due to limitation of domestic carcasses. It causes an impact on the biomass. When only considering wild ungulates, the ecosystem cannot offer enough food for predators.
M.A. Colomer et al. 2012 [12]	**A carnivore that predates on ungulates and five ungulates** Catalan Pyrenees(NE)	Impacts of environment factors such as weather, orography, and soil conditions on carnivore size
A. Margalida et al. 2012 [38]	**European vultures** as **the Bearded vulture, Egyptian vulture, and Cinereous vulture** 10 municipalities in Catalonia, Northern Spain.	Food source: four scenarios of food availability. Taking 10 areas and 4 avian scavengers as research object. Impact of climate variations, such as seasons (summer and winter), food shortage, density regulation, and changes in species habitats (insufficient resources)

Table 3.3 Ecosystems models based on P systems, simulated with EcoSim/MeCoSim (II)

Reference	Case study	Comments
M.A. Colomer et al. 2013 [13]	**Birds, cats, and rats** General model	Prey-predator. Natural mortality, intraguild predation, and mesopredator release effect on long-lived prey
M.A. Colomer et al. 2014 [14]	**Zebra mussel** Reservoir of Ribarroja	Twenty membranes used for 20 weeks first reproductive cycle, 16 for the weeks of second reproductive cycle, and 2 membranes to handle regulation and mortality
M.A. Colomer et al. 2014 [15]	**Calotriton asper (newt)** Pi Valley (Noth Spain) water streams	Reproduction, mortality, and displacements in the terrestrial environment, possibly colonizing new streams
A. Margalida et al. 2015 [40]	**Avian scavengers** Pi Valley (Noth Spain) water streams	Impact of removal scenarios on population viability
A. Cortés-Avizanda et al. 2015 [18]	**Wild rabbits and avian scavengers** Mediterranean landscapes	Reproduction, mortality, foraging behavior of Egyptian vultures, wild rabbit carcass biomass availability, maximum carrying capacity, and carcass-sharing with competitors
A. Kane et al. 2015 [32]	**Gyps africanus** Hlane-Mlawula-Mbuluzi reserve network in Swaziland	Carrion feeding, feeding needs analysis, natural mortality, and many interacting species
C. Fondevilla et al. 2016 [24]	**Land use and land cover (plant communities)** Stubai Valley (Central Alps)	Grazing, foraging, natural mortality of animal species, movement, and land use
Z. Huang, G. Zhang, et al. 2017 [29]	**Domestic Giant Panda** Chengdu Research Base of Giant Panda Breeding (GPBB), Wolong China Conservation and Research Center for Giant Panda (CCRCGP)	The evolution process of the species: RMF+Rescue module, where RMF is also modified as RFM, FMR, or other forms, showing the robustness of the system independently on the order of the modules
H. Tian, G. Zhang, et al. 2018 [65]	**Domestic Giant Panda** Two regions: GPBB/CCRCGP	The membrane structure is the same as in [29], and the only difference is that release module is added to the previous module, that is, RMF+Rescue module+Release module.
A. Margalida, et al. 2018 [41]	**European avian scavengers** North Spain	Two periods (summer, breeding), reproduction, mortality, feeding, and carrying capacity. Forage in peripheral areas depending on availability in origin
M.A. Colomer, et al. 2019 [17]	**Porcine Reproductive and Respiratory Syndrome** Spain	Births, lactation, transmission, and fattening
M.A. Colomer, et al. 2020 [16]	**Porcine production** Vaccination against Aujeszky's disease Spain	Based on [17]
Y. Duan, et al. 2020 [22]	**Giant Panda in captivity** GPBB and related centers	Reproduction, mortality, feeding, and rescue models

Table 3.4 Ecosystems models based on P systems, simulated with EcoSim/MeCoSim (III)

Reference	Case study	Comments
L. Valencia-Cabrera, et al. 2013[68]	**Gene regulatory networks** General model	The first membrane computing model applied to reconstruct the behavior of logic networks of species with PDP systems
L. Valencia-Cabrera, et al. 2013[69]	**Gene regulatory networks** Arabidopsis thaliana	Based on [68], P systems are used to reproduce a logic gene network of (real) Arabidopsis thaliana in order to regulate the flowering processes.
M.A. Colomer et al. 2014 [14]	**Pandemics** General model	Different areas, neighborhoods, families, and infections at home, in school, workplace, among communities, etc.
E. Sánchez-Karhunen, et al. 2019[61]	**Market interactions**	Economic ecosystem modelled with PDP systems

of the enzymatic numerical P systems (ENPS) [53] framework to simulate basic RRT algorithms. In [57], the framework of ENPS was used for modelling the RRT and RRT* algorithms. It is worth pointing out that in [57], no additional ingredients to the ENPS framework were included. In consequence, the resulting models are compatible with existent ENPS robot controllers [52,53,77]. In [57], two simulators were also presented: The first one is based on OpenMP, and the second one is based on CUDA. The current challenges in this research line are related to simulate on hardware (FPGA) the models, to adapt the software/hardware simulators to actual robots, and, finally, to study the inclusion of dynamic obstacles such as people surrounding the robot.

3.5.1 Problem Definition

Let $\mathcal{X} \subseteq \mathbb{R}^d$ be the *configuration space* of the robot, where $d \in \mathbb{N}, d \geq 2$. Let $\mathcal{X}_{obs}$ be the *obstacle space* and $\mathcal{X}_{free}$ be the *obstacle-free space* such that $\mathcal{X} = \mathcal{X}_{obs} \cup \mathcal{X}_{free}$ and $\mathcal{X}_{obs} \cap \mathcal{X}_{free} = \emptyset$. Let the *initial configuration* $x_{init} \in \mathcal{X}_{free}$ and the *goal region* $\mathcal{X}_{goal} \subsetneq \mathcal{X}_{free}$. A motion planning problem is defined by $(\mathcal{X}_{free}, x_{init}, \mathcal{X}_{goal})$.

A function $\sigma : [0, 1] \rightarrow \mathbb{R}^d$ is called:

- Path, if it is continuous;
- Collision-free path, if it is a path and $\sigma(\tau) \in \mathcal{X}_{free}$, for all $\tau \in [0, 1]$;
- Feasible path, if it is a collision-free path, $\sigma(0) = x_{init}$ and $\sigma(1) \in \mathcal{X}_{goal}$.

The motion planning problem can be solved in two ways:

1. Given a motion planning problem $(\mathcal{X}_{free}, x_{init}, \mathcal{X}_{goal})$, find a feasible path σ. If no such path exists, return failure. This is called *the feasible motion planning*.

2. Given a motion planning problem $(\mathcal{X}_{free}, x_{init}, \mathcal{X}_{goal})$ and a cost function $c : \sigma \rightarrow \mathbb{R}$, find a feasible path σ^* such that $c(\sigma^*) = min\{c(\sigma) : \sigma isfeasible\}$. If no such path exists, return failure. This is called *the optimal motion planning*.

3.5.2 Path Planning for Mobile Robots

The path planning problem for mobile robots is a type of motion planning problem in which a wheeled or legged robot is considered, and it should navigate from an initial position to a goal region while avoiding obstacles. We can consider two types of mobile robots: On the one hand, *holonomic robots* are those that can move in any direction from its current state. On the other hand, *nonholonomic robots* have constrained motions with respect to its current state. For example, a two-wheeled robot is a nonholonomic robot that cannot follow a direction along its axes without applying previously a rotation in-place motion.

For the sake of simplicity, we will consider holonomic robots in the rest of this section. For this type of robots, the configuration space can be defined as $\mathcal{X} = \{(x, y)\} \subseteq \mathbb{R}^2$ where (x, y) are the Cartesian coordinates of the center of the robot. The radius of the robot is given by a constant R, and the sets $\mathcal{X}_{obs}$ and $\mathcal{X}_{free}$ are given by an occupancy matrix.

3.5.3 Rapidly-Exploring Random Tree (RRT) Algorithm

The RRT algorithm [36] is a classical solution to the feasibility motion planning problem. On the other hand, the RRT* algorithm [33] provides an approximate solution to the optimal motion planning problem. The original algorithms are sequential, but there are parallel versions as [2].

In [56] and [57], membrane computing has been used as computational framework to model parallel versions of such algorithms, providing also simulators in parallel architectures such as OpenMP and CUDA.

In general terms, the RRT algorithm gives a solution to the feasible motion planning problem by making a random tree exploring the free-obstacle configuration space. The nodes in the tree represent states in the obstacle-free space, and the edges represent transitions or movements between such states. The root is located in the initial robot position. The algorithm explores the space until a node in the goal region is reached or until a number of iterations. For a holonomic robot, nodes contain Cartesian coordinates in a 2D space, and edges represent straight-line movements.

In Fig. 3.12, an example of the RRT is represented. It can be seen as the free space is explored by the edges of the tree.

Fig. 3.12 A rapidly exploring random tree example

3.6 Conclusion

Following software implementation of P systems in Chap. 2, this chapter discussed the use of P-Lingua and MeCoSim to fulfill some applications such as automatic design of cell-like P systems and spiking neural P systems for performing specific tasks, and modelling ecosystems and robot path planning. P-Lingua is a widely used simulator for many variants of P systems such as cell- and tissue-like P systems, spiking neural P systems, fuzzy reasoning spiking neural P systems, and kernel P systems. MeCoSim is a visualization simulator based on P-Lingua. Both of them are very useful to support the exploration of more and more applications with automation, such as power system fault diagnosis, modelling giant panda ecosystem, and mobile robot controller design.

References

1. A. Alhazov, C. Martín-Vide, L. Pan, Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. Fundam. Inform. **58**(2), 66–77 (2003)
2. J. Bialkowski, S. Karaman, E. Frazzoli, Massively parallelizing the RRT and the RRT*Massively parallelizing the RRT and the RRT*, in *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Francisco, CA, 2011* (2011), pp. 3513–3518. https://doi.org/10.1109/IROS.2011.6095053
3. M. Cardona, M.A. Colomer, M.J. Pérez-Jiménez, D. Sanuy, A. Margalida, A P System modeling an ecosystem related to the bearded vulture, in *Proceedings of the Sixth Brainstorming Week on Membrane Computing*, Fénix Editora, ed. by D. Díaz-Pernil, C. Graciani, M.A. Gutiérrez-Naranjo, Gh. Păun, I. Pérez-Hurtado, A. Riscos-Núñez (2008), pp. 51–66
4. M. Cardona, M.A. Colomer, M.J. Pérez-Jiménez, Modeling ecosystems using P systems: the bearded vulture, a case study, in *Membrane Computing (WMC 2008)*, ed. by D.W. Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 5391

(2009), pp. 137–156. https://doi.org/10.1007/978-3-540-95885-7_11
5. M. Cardona, M.A. Colomer, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy, A P system based model of an ecosystem of some scavenger birds, in *Membrane Computing (WMC 2009)*, ed. by Gh. Păun, M.J. Pérez-Jiménez, A. Riscos, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 5957 (2010), pp. 182–195. https://doi.org/978-3-642-11467-0_14
6. M. Cardona, M.A. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy, A computational modeling for real ecosystems based on P systems. Nat. Comput. **10**(1), 39–53 (2011). https://doi.org/10.1007/s11047-010-9191-3
7. Y. Chen, G. Zhang, T. Wang, X. Huang, Automatic design of a P system for basic arithmetic operations. Chin. J. Electron. **23**(2), 302–304 (2014)
8. M.A. Colomer, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez Jiménez, A uniform framework for modeling based on P systems, in *Proceedings of the 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010), Changsha, China*, vol. 1 (IEEE Press, New York, 2010), pp. 616–621. https://doi.org/10.1109/BICTA.2010.5645196
9. M.A. Colomer, C. Fondevilla, L. Valencia-Cabrera, A new P system to model the subalpine and alpine plant communities, in *Proceedings of the Ninth Brainstorming Week on Membrane Computing*, Fénix Editora, ed. by M.A. Martínez-del-Amor, Gh. Păun, I. Pérez-Hurtado, F.J. Romero-Campero, L. Valencia-Cabrera (2011), pp. 91–112
10. M.A. Colomer, S. Lavín, I. Marco, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy, E. Serrano, L. Valencia-Cabrera, Modeling population growth of Pyrenean chamois (Rupicapra p. pyrenaica) by using P-systems, in *Membrane Computing (CMC 2010)*, ed. by M. Gheorghe, T. Hinze, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 6501 (2011), pp. 144–159. https://doi.org/10.1007/978-3-642-18123-8_13
11. M.A. Colomer, A. Margalida, D. Sanuy, M.J. Pérez-Jiménez, A bio-inspired computing model as a new tool for modeling ecosystems: the avian scavengers as a case study. Ecol. Modell. **222**(1), 33–47 (2011). https://doi.org/10.1016/j.ecolmodel.2010.09.012
12. M.A. Colomer, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, Comparing simulation algorithms for multienvironment probabilistic P systems over a standard virtual ecosystem. Nat. Comput. **11**(3), 369–379 (2012). https://doi.org/10.1007/s11047-011-9289-2
13. M.A. Colomer, A. Margalida, M.J. Pérez-Jiménez, Population Dynamics P System (PDP) models: a standardized protocol for describing and applying novel bio-inspired computing tools. PloS One **8**(4) (2013). https://doi.org/10.1371/journal.pone.0060698
14. M.A. Colomer, A. Margalida, L. Valencia-Cabrera, A. Palau, Application of a computational model for complex fluvial ecosystems: the population dynamics of zebra mussel dreissena polymorpha as a case study. Ecol. Complexity **20**, 116–126 (2014). https://doi.org/10.1016/j.ecocom.2014.09.006
15. M.A. Colomer, A. Montori, E. García, C. Fondevilla, Using a bioinspired model to determine the extinction risk of Calotriton asper populations as a result of an increase in extreme rainfall in a scenario of climatic change. Ecol. Modell. **281**, 1–14 (2014). https://doi.org/10.1016/j.ecolmodel.2014.02.018
16. M.A. Colomer, A. Margalida, L. Fraile, Improving the management procedures in farms infected with the porcine reproductive and respiratory syndrome virus using PDP models. Sci. Rep. **9**, 9959 (2019). https://doi.org/10.1038/s41598-019-46339-w
17. M.A. Colomer, A. Margalida, L. Fraile, Vaccination is a suitable tool in the control of Aujeszky's disease outbreaks in pigs using a Population Dynamics P Systems model. Animals **10**, 909 (2020). https://doi.org/10.3390/ani10050909
18. A. Cortés-Avizanda, M.A. Colomer, A. Margalida, O. Ceballos, J.A. Donázar, Modeling the consequences of the demise and potential recovery of a keystone-species: wild rabbits and avian scavengers in Mediterranean landscapes. Sci. Rep. **5**, 17033 (2015). https://doi.org/10.1038/srep17033
19. K. Daniel, A. Nash, S. Koenig, A. Felner, Theta*: Any-Angle Path Planning on Grids. J. Artif. Intell. Res. **39**, 533–579 (2010). https://doi.org/10.1613/jair.2994

20. D. Díaz-Pernil, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, A P-Lingua programming environment for Membrane Computing, in *Membrane Computing (WMC 2008)*, ed. by D.W. Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 5391 (2009), pp. 187–203. https://doi.org/10.1007/978-3-540-95885-7_14
21. J. Dong, M. Stachowicz, G. Zhang, M. Cavaliere, H. Rong, P. Paul, Automatic design of spiking neural P systems based on genetic algorithms. Int. J. Unconv. Comput. **16**(2–3), 201–216 (2021)
22. Y. Duang, H. Rong, D. Qi, L. Valencia-Cabrera, G. Zhang, M.J. Pérez-Jiménez, A review of membrane computing models for complex ecosystems and a case study on a complex Giant Panda system. Complexity 2020, Article ID 1312824, 26, (2020). https://doi.org/10.1155/2020/1312824
23. G. Escuela, M.A. Gutiérrez-Naranjo, An application of genetic algorithms to Membrane Computing, in *Proceedings of the Eighth Brainstorming Week on Membrane Computing*, Fénix Editora, ed. by M.A. Martínez-del-Amor, Gh. Păun, I. Pérez-Hurtado, A. Riscos-Núñez (2010), pp. 101–118
24. C. Fondevilla, M.A. Colomer, F. Fillat, U. Tappeiner, Using a new PDP modelling approach for land-use and land-cover change predictions: a case study in the Stubai Valley (Central Alps). Ecol. Modell. **322**, 101–114 (2016). https://doi.org/10.1016/j.ecolmodel.2015.11.016
25. M. García-Quismondo, R. Gutiérrez-Escudero, M.A. Martínez-del-Amor, E. Orejuela-Pinedo, I. Pérez-Hurtado, P-Lingua 2.0: a software framework for cell-like P systems. Int. J. Comput. Commun. Control **4**(3), 234–243 (2009). https://doi.org/10.15837/ijccc.2009.3.2431
26. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. An Overview of P-Lingua 2.0, in *Membrane Computing (WMC 2009)*, ed. by Gh. Păun, M.J. Pérez-Jiménez, A. Riscos, G. Rozenberg, A. Salomaa. Lecture Notes in Computer Science, vol. 5957 (2010), pp. 264–288. https://doi.org/10.1007/978-3-642-11467-0_20
27. J. He, J. Xiao, X. Liu, T. Wu, T. Song, A novel membrane-inspired algorithm for optimizing solid waste transportation. Optik—Int. J. Light Electron Opt. **126**(23), 3883–3888 (2015). https://doi.org/10.1016/j.ijleo.2015.07.152
28. X. Huang, G. Zhang, H. Rong, F. Ipate, Evolutionary Design of a Simple Membrane System, in *Membrane Computing (CMC 2011)*, ed. by M. Gheorghe, Gh. Păun, G. Rozenberg, A. Salomaa, S. Verlan. Lecture Notes in Computer Science, vol. 7184 (2012), pp. 203–214. https://doi.org/10.1007/978-3-642-28024-5_14
29. Z. Huang, G. Zhang, D. Qi, H. Rong, M.J. Pérez-Jiménez, L. Valencia-Cabrera, Application of probabilistic membrane systems to model giant panda population data. Comput. Syst. Appl. **26**(8), 252–256 (2017). https://doi.org/10.15888/j.cnki.csa.005878 (in Chinese)
30. M. Ionescu, Gh. Păun, T. Yokomori, Spiking Neural P Systems. Fundam. Inform. **71**(2–3), 279–308 (2006)
31. J. Juico, J. Silapan, F.G.C. Cabarle, I. Macababayao, R.T.A. De la Cruz. Evolving spiking neural P systems with polarization. Philipp. Comput. J. (Special Issue on P systems) **14**(2), 11–20 (2020)
32. A. Kane, A.L. Jackson, A. Monadjem, M.A. Colomer, A. Margalida, Carrion ecology modelling for vulture conservation: are vulture restaurants needed to sustain the densest breeding population of the African white-backed vulture? Anim. Conserv. **18**(3), 279–286 (2015). https://doi.org/10.1111/acv.12169
33. S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning. Int. J. Rob. Res. **30**(7), 846–894 (2011). https://doi.org/10.1177/0278364911406761
34. S. Kazarlis, A. Bakirtzis, V. Petridis, A genetic algorithm solution to the unit commitment problem. IEEE Trans. Power Syst. **11**(1), 83–92 (1996). https://doi.org/10.1109/59.485989
35. J. Latombe, Motion planning: a journey of robots, molecules, digital actors, and other artifacts. Int. J. Rob. Res. **18**(11), 1119–1128 (1999). https://doi.org/10.1177/02783649922067753
36. S. LaValle, *Rapidly-exploring Random Trees: A New Tool for Path Planning, TR 98-11* (Computer Science Department, Iowa State University, Iowa, 1998). http://lavalle.pl/papers/Lav98c.pdf

37. X. Liu, J. Suo, S. Leung, J. Liu, X. Zeng, The power of time-free tissue P systems: Attacking NP-complete problems. Neurocomputing **159**, 151–156 (2015). https://doi.org/10.1016/j.neucom.2015.01.072
38. A. Margalida, M.A. Colomer, Modelling the effects of sanitary policies on European vulture conservation. Sci. Rep. **2**, 753 (2012). https://doi.org/10.1038/srep00753
39. A. Margalida, M.A. Colomer, D. Sanuy, Can wild ungulate carcasses provide enough biomass to maintain avian scavenger populations? An empirical assessment using a bio-inspired computational model. PloS One **6**(5), e20248 (2011). https://doi.org/10.1371/journal.pone.0020248
40. A. Margalida, M.A. Colomer, D. Oro, R. Arlettaz, J.A. Donázar, Assessing the impact of removal scenarios on population viability of a threatened, long-lived avian scavenger. Sci. Rep. **5**, 16962 (2015). https://doi.org/10.1038/srep16962
41. A. Margalida, P. Oliva-Vidal, A. Llamas, M.A. Colomer, Bioinspired models for assessing the importance of transhumance and transboundary management in the conservation of European avian scavengers. Biol. Conserv. **228**, 321–330 (2018). https://doi.org/10.1016/j.biocon.2018.11.004
42. MeCoSim website. http://www.p-lingua.org/mecosim
43. K. Meffert, J. Meseguer, E.D. Mart, A. Meskauskas, J. Vos, N. Rotstan (last visited-July 2011), *JGAP—Java Genetic Algorithms and Genetic Programming Package* (2011). http://jgap.sf.net
44. M. Mitchell, *An Introduction to Genetic Algorithms* (MIT Press, Cambridge, 1998)
45. NVIDIA CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit. NVIDIA Corporation. Online (accesed August 2019)
46. OpenMP specification, version 4.5. https://www.openmp.org/specifications. The OpenMP ARB (Architecture Review Boards). Online (accessed August 2019)
47. Z. Ou, G. Zhang, T. Wang, X. Huang, Automatic design of cell-like P systems through tuning membrane structures, initial objects and evolution rules. Int. J. Unconv. Comput. **9**(5–6), 425–443 (2013)
48. L. Pan, Gh. Păun, M.J. Pérez-Jiménez, Spiking neural P systems with neuron division and budding. Sci. China Inf. Sci. **54**(8), 1596–1607 (2011). https://doi.org/10.1007/s11432-011-4303-y
49. Gh. Păun, Computing with membranes. J. Comput. Syst. Sci. **61**(1), 108–143 (2000). https://doi.org/10.1006/jcss.1999.1693 (first circulated at TUCS Research Report No. 208, November 1998. http://www.tucs.fi)
50. Gh. Păun, G. Rozenberg, A guide to membrane computing. Theor. Comput. Sci. **287**, 73–100 (2002). https://doi.org/10.1016/S0304-3975(02)00136-6
51. Gh. Păun, G. Rozenberg, A. Salomaa, *The Oxford Handbook of Membrane Computing* (Oxford University, Oxford, 2010)
52. A.B. Pavel, C. Buiu, Using enzymatic numerical P systems for modeling mobile robot controllers. Nat. Comput. **11**(3), 387–393 (2012). https://doi.org/10.1007/s11047-011-9286-5
53. A.B. Pavel, O. Arsene, C. Buiu, Enzymatic numerical P systems: a new class of Membrane Computing systems, in *Proceedings of the 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), Changsha, 2010* (2010), pp. 1331–1336. https://doi.org/10.1109/BICTA.2010.5645071
54. I. Pérez-Hurtado, *Desarrollo y Aplicaciones de un Entorno de Programación para Computación Celular: P-Lingua*. Ph.D. Thesis (Universidad de Sevilla, Sevilla, 2010, in Spanish). http://hdl.handle.net/11441/66241
55. I. Pérez-Hurtado, L. Valencia-Cabrera, M.J. Pérez-Jiménez, M.A. Colomer, A. Riscos-Núñez, MeCoSim: a general purpose software tool for simulating biological phenomena by means of P systems, in *Proceedings of the IEEE Fifth International Conference on Bio-inspired Computing: Theories and Applications (BIC-TA 2010)*, vol. I, ed. by K. Li, Z. Tang, R. Li, A.K. Nagar, R. Thamburaj (2010), pp. 637–643. https://doi.org/10.1109/BICTA.2010.5645199

56. I. Pérez-Hurtado, M.J. Pérez-Jiménez, G. Zhang, D. Orellana-Martín. Simulation of rapidly-exploring random trees in Membrane Computing with P-lingua and automatic programming. Int. J. Comput. Commun. Control **13**(6), 1007–1031 (2019). https://doi.org/10.15837/ijccc.2018.6.3370
57. I. Pérez-Hurtado, M.A. Martínez-del-Amor, G. Zhang, F. Neri, M.J. Pérez-Jiménez, A membrane parallel rapidly-exploring random tree algorithm for robotic motion planning. Integr. Comput.-Aided Eng. **27**, 1–18 (2020). https://doi.org/10.3233/ICA-190616
58. P-Lingua website (last visited, July 2011). http://www.p-lingua.org
59. J.H. Reif, Complexity of the mover's problem and generalizations, in *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (SFCS 1979), San Juan, Puerto Rico, USA* (1979), pp. 421–427. https://doi.org/10.1109/SFCS.1979.10
60. S. Ronald, Robust encodings in genetic algorithms: a survey of encoding issues, in *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97), Indianapolis, IN, USA, 1997* (1997), pp. 43–48. https://doi.org/10.1109/ICEC.1997.592265
61. E. Sánchez-Karhunen, L. Valencia-Cabrera, Modelling complex market interactions using PDP systems. J. Membr. Comput. **1**(1), 40–51 (2019). https://doi.org/10.1007/s41965-019-00008-z
62. T. Song, L.F. Macías-Ramos, L. Pan, M.J. Pérez-Jiménez, Time-free solution to SAT problem using P systems with active membranes. Theor. Comput. Sci. **529**, 61–68 (2014). https://doi.org/10.1016/j.tcs.2013.11.014
63. T. Song, Q. Zou, X. Liu, X. Zeng, Asynchronous spiking neural P systems with rules on synapses. Neurocomputing **151**, 1439–1445 (2015). https://doi.org/10.1016/j.neucom.2014.10.044
64. A. Stentz, The focussed D* algorithm for real-time replanning, in *IJCAI95: Proceedings of the 14th International Joint Conference on Artificial Intelligence*, vol. 2 (1995), pp. 1652–1659
65. H. Tian, G. Zhang, H. Rong, et al. Population model of giant panda ecosystem based on population dynamics P system. J. Comput. Appl. **38**(5), 1488–1493 (2018). https://doi.org/10.11772/j.issn.1001-9081.2017102551 (in Chinese)
66. C. Tudose, R. Lefticaru, F. Ipate, Using genetic algorithms and model checking for P systems automatic design, in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2011). Studies in Computational Intelligence*, vol 387, ed. by D.A. Pelta, N. Krasnogor, D. Dumitrescu, C. Chira, R. Lung (2012), pp. 285–302. https://doi.org/10.1007/978-3-642-24094-2_20
67. L. Valencia-Cabrera, *An Environment for Virtual Experimentation with Computational Models Based on P Systems*. Ph.D. Thesis (Universidad de Sevilla, Sevilla, 2015). http://hdl.handle.net/11441/45362
68. L. Valencia-Cabrera, M. García-Quismondo, M.J. Pérez-Jiménez, Y. Su, H. Yu, L. Pan, Modeling logic gene networks by means of probabilistic dynamic P systems. Int. J. Unconv. Comput. **9**(5–6), 445–464 (2013)
69. L. Valencia-Cabrera, M. García Quismondo, M.J. Pérez-Jiménez, Analysing gene networks with PDP systems. Arabidopsis thailiana, a case study, in *Proceedings of the Eleventh Brainstorming Week on Membrane Computing*, Fénix Editora, ed. by L. Valencia-Cabrera, M. García-Quismondo, L.F. Macías-Ramos, M.A. Martínez-del-Amor, Gh. Păun, A. Riscos-Núñez (2013), pp. 257–272
70. M. Yuan, G. Zhang, M.J. Pérez-Jiménez, T. Wang, Z. Huang, P systems based computing polynomials: design and formal verification. Nat. Comput. **15**, 591–596 (2016). https://doi.org/10.1007/s11047-016-9577-y
71. G. Zhang, J. Cheng, M. Gheorghe, Q. Meng, A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems. Appl. Soft Comput. **13**(3), 1528–1542 (2013). https://doi.org/10.1016/j.asoc.2012.05.032
72. G. Zhang, M. Gheorghe, L. Pan, M.J. Pérez-Jiménez, Evolutionary membrane computing: a comprehensive survey and new results. Inf. Sci. **279**, 528–551 (2014). https://doi.org/10.1016/j.ins.2014.04.007

73. G. Zhang, H. Rong, F. Neri, M.J. Pérez-Jiménez, An optimization spiking neural P system for approximately solving combinatorial optimization problems. Int. J. Neural Syst. **24**(5), 01–16 (2014). https://doi.org/10.1142/S0129065714400061
74. G. Zhang, H. Rong, Z. Ou, M.J. Pérez-Jiménez, M. Gheorghe, Automatic design of deterministic and non-halting membrane systems by tuning syntactical ingredients. IEEE Trans. Nanobiosci. **13**(3), 363–371 (2014). https://doi.org/10.1109/TNB.2014.2341618
75. X. Zhang, Y. Liu, B. Luo, L. Pan, Computational power of tissue P systems for generating control languages. Inf. Sci. **278**, 285–297 (2014). https://doi.org/10.1016/j.ins.2014.03.053
76. G. Zhang, J. Cheng, T. Wang, X. Wang, J. Zhu, *Membrane Computing: Theory and Applications* (Science China Press, Beijing, 2015) (in Chinese)
77. G. Zhang, M.J. Pérez-Jiménez, M. Gheorghe, *Real-life Applications with Membrane Computing*. Series Emergence, Complexity and Computation (Springer, Berlin, 2017)
78. G. Zhang, H. Rong, P. Paul, Y. He, F. Neri, M.J. Pérez-Jiménez, A complete arithmetic calculator constructed from spiking neural P systems and its application to information fusion. Int. J. Neural Syst., 2050055 (2020). Available online, published 16 September 2020. https://doi.org/10.1142/S0129065720500550
79. J. Zhao, X. Wang, G. Zhang, F. Neri, T. Jiang, M. Gheorghe, F. Ipate, R. Lefticaru, Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots. Integr. Comput.-Aided Eng. **23**, 15–30 (2016). https://doi.org/10.3233/ICA-150503
80. M. Zhu, G. Zhang, Q. Yang, H. Rong, W. Yuan, M.J. Pérez-Jiménez. P systems based computing polynomials with integer coefficients: design and formal verification. IEEE Trans. NanoBiosci. **17**(3), 272–280 (2018). https://doi.org/10.1109/TNB.2018.2836147