Knowledge-Based Systems 170 (2019) 20–25

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Original software publication

UPSimulator: A general P system simulator

Ping Guo [a,b,*], Changsheng Quan [a], Lian Ye [a]

[a] College of Computer Science, Chongqing University, Shazhengjie. 174, Chongqing 400044, China
[b] Chongqing Key Laboratory of Software Theory & Technology, Chongqing 400044, China

ARTICLE INFO

Article history:
Received 12 September 2018
Received in revised form 4 January 2019
Accepted 5 January 2019
Available online 10 February 2019

Keywords:
Membrane computing
Cell-like P system
Tissue-like P system
Neural-like P system
Hybrid P system
Simulator

ABSTRACT

In membrane computing, researchers have designed various P systems to solve problems, for example, NP-hard problems. The research of verification methods and simulation tools for these P systems is one of the main topics in membrane computing. In this paper, we present a general-purpose P system simulator named UPSimulator for cell-like, tissue-like and neural-like P systems. UPSimulator can serve as both a simulator and a development library for P system models, which processes objects in the regions defined by membranes. UPSimulator provides a feasible solution for the simulation of hybrid P systems by supporting the complex evolutionary rules and structures in different types of P systems.

1. Introduction

Membrane computing is a branch of natural computing which abstracts distributed parallel computing models from the structure and functioning of living cells [1]. The P system is a computing system designed based on a membrane computing model. In P system, information transfer (communication) is an abstraction of material transfer between cells, and information processing (data processing) is an abstraction of intracellular biochemical reactions. Many variants of P systems were introduced in the past few years. Most of these variants can be divided into three types, such as cell-like P systems, tissue-like P systems [2,3] and neural-like P systems [4]. The cell-like P systems consist of membrane structure, objects and evolutionary rules. The evolutionary rules are used to control objects evolution within a membrane and information transaction between membranes. In a tissue-like P system, membranes are freely placed in the environment, each membrane consisting of objects and rules. Environment and membrane use rules to communicate with each other. The neural-like P system has a neural cell structure, the information is represented by spikes, and the spikes are sent and processed by activated evolutionary rules. These P systems have been applied in many fields, for examples, fuzzy inference [5], numerical computation [6–8] and machine learning [9–11]. Moreover, based on these three models, many P systems with polynomial time complexity have been designed to solve NP-hard problems, such as All-Set [12], TSP [13], Hamilton [14], N-Queen [15] and so on.

In recent years, not only many new features have been abstracted to form new P systems, but also the features of existing P systems have been combined to form a new P system. However, existing P system simulators are developed for specific P systems. For example, P-Lingua provides support for cell-like P systems, tissue-like P systems, and neural-like P systems, but it does not support a cSN P system formed by a combination of a cell-like P system and a neural-like P system. SimCM is a simulator for Transition P Systems. Like P-Lingua, SimCM cannot support new P systems formed by different P system combinations. Therefore, when a new P system is proposed, even if the rule types in the new P system are already used in the existing simulator, it is necessary to maintain these existing simulation tools or develop a new simulator. This paper deals with this problem and designs a generic simulation tool called UPSimulator that not only supports a large number of new features in cell-like, tissue-like, and neural-like P systems, but also supports their combinations.

2. Problems and background

P system simulators are important tools for designing and verifying P systems. So far, researchers have developed several P-system simulators [16–18], which can simulate some of the proposed P-system models. However, with the increasing research and application of membrane computing, new P system models are constantly being proposed. For example, in recent years, cell-like Spiking Neural P Systems [19,20], tissue P systems with evolutional symport/antiport rules [21,22], Spiking Neural P Systems with multiple channels [23,24], dynamic threshold neural P systems [25] and coupled neural P systems [26] have been introduced.

* Corresponding author at: College of Computer Science, Chongqing University, Shazhengjie. 174, Chongqing 400044, China.
E-mail address: guoping@cqu.edu.cn (P. Guo).

https://doi.org/10.1016/j.knosys.2019.01.013

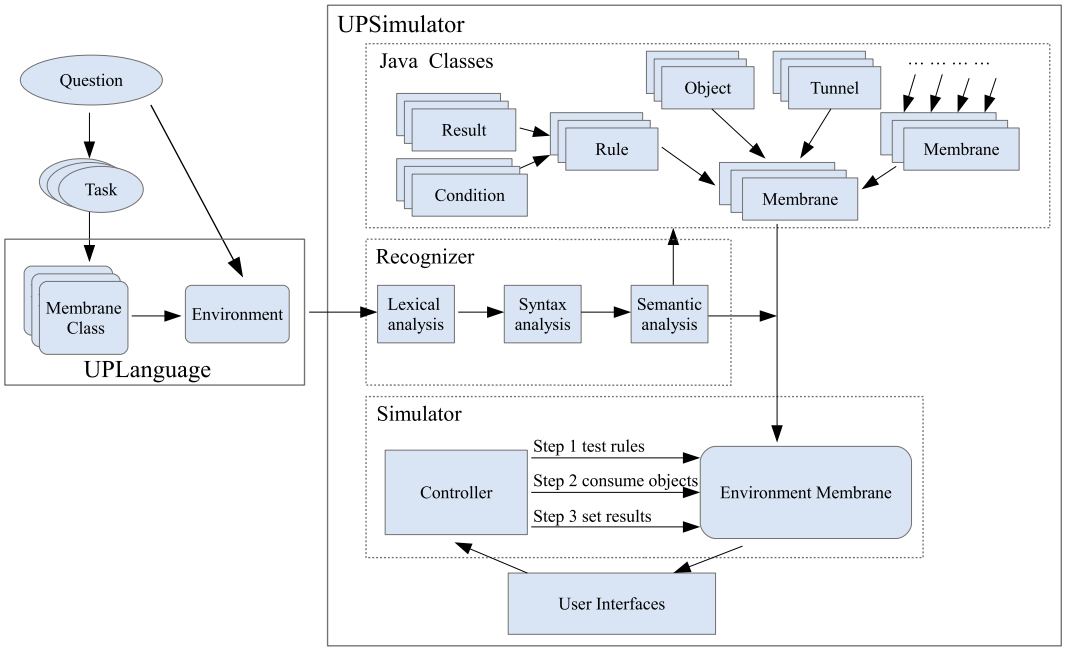


Fig. 1. The framework of UPSimulator.

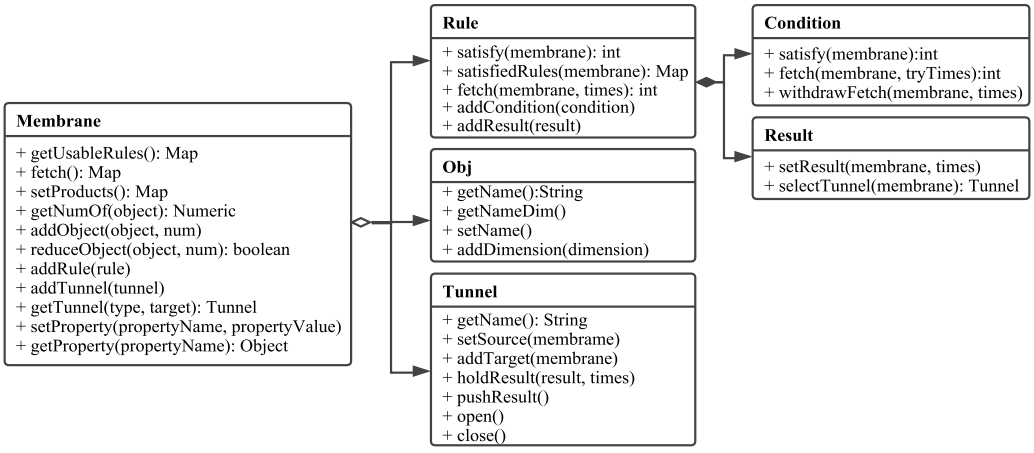


Fig. 2. UML interface diagram of UPSimulator.

These new models are not well supported by the existing simulators.

In general, the proposed new P system models mainly include two aspects. First, new membrane structures or new evolutionary rules are presented in the new P system, e.g., multi-channel P systems. Second, the new P system combines the characteristics of different P systems. For example, cell-like SN P Systems combines the hierarchical structure of cell-like P system with the SN P system. Therefore, a general-purpose P system simulator is required to handle the above two problems.

The simulator presented in this paper addresses these issues. At first, a new description language UPLanguage is proposed to describe objects, rules and membrane structures of the P system. In addition, different concept interfaces are designed to implement different rule types and membrane structures in UPSimulator. Finally, by implementing the existing interfaces, UPSimulator can also be extended to implement new rule types and membrane structures in the future.

3. Software framework and functionalities

UPSimulator consists of the description language and the simulator. The language called UPLanguage can be used to describe the P system. The simulator is used to simulate and verify the P system described by UPLanguage.

3.1. UPLanguage

UPLanguage is an object-oriented description language for a P system model. It is used to describe the membrane structure, the multisets and the rule sets. In UPLanguage, a membrane with a certain function can be defined as a membrane class. Membrane structure is determined by the relationship between membrane instances. The UPLanguage rule is divided into two parts, conditions and results. The combinations of different conditions and results can be described by UPLanguage, which extends the description ability of UPLanguage. UPLanguage uses the symbol '^' to describe the number of objects in a multiset. For example, 3 copies of object a can be described as $a\hat{}3$. In this way, we can describe multisets more flexible and convenient.

Table 1
Supported concepts in UPSimulator.

Concepts	Cell-like P system	Tissue-like P system	Neural-like P system
Promoter	✓	✓	✓
Inhibitor	✓	✓	✓
Probability	✓	✓	✓
Rule priority	✓	✓	✓
Regular expression	✓	✓	✓
Thickness	✓	✓	✓
Polarity	✓	✓	✓
Dissolution	✓	✓	✓
Division	✓	✓	✓
Creation	✓	✓	✓
Symport/ Antiport	✓	✓	✓
Multiple channels	✓	✓	✓
Anti-object/ Anti-spike	✓	✓	✓
Delay	✓	✓	✓

The main features of UPLanguage are:

(1) In UPLanguage, membrane class is used to describe a class of membranes with a certain function. For example, the Dijkstra algorithm can be implemented to a membrane class. If one wants to use the algorithm, just create an instance of the membrane class. Thus, a standard algorithm library can be built by UPLanguage.

(2) UPLanguage can form new evolutionary rules by describing combinations of different concepts. In terms of this feature, the simulator can well adapt to the extension of existing P system and newly proposed P system.

(3) A dimension label system is designed in UPLanguage. It simplifies the description of membranes, objects and rules. It also makes UPLanguage more powerful to describe the relationship between reactants and products. For example, rule $h_{i,j} \rightarrow (k_{i+1,j+1}$, in $m_i)$ can be described as h[i][j] → (k[i+1][j+1], in m[i]).

(4) A special membrane named "Environment" is used to place the instances of membranes that need to be verified. All the contents of "Environment" will be simulated by UPSimulator.

(5) For flexibility, UPLanguage is recognized by the parser generator ANTLR [27]. New changes of syntax can be quickly supported by using ANTLR.

3.2. UPSimulator framework

Fig. 1 shows the framework of UPSimulator, including Java classes, recognizer, simulator, and user interfaces. The Java classes show the owner–member relationship between those classes. The recognizer consists of lexical analysis, syntax analysis and semantic analysis. The simulator is composed of an environment membrane and a controller. The controller ensures that all the rules evolve correctly. The user interface part provides ways to control the simulation process and shows the results.

The execution process of UPSimulator is also shown in Fig. 1. Firstly, the description text of P system written in UPLanguage will be recognized by recognizer. In addition, according to the meaning of the description text, conditions and results will make up rules. Then, rules, objects, tunnels and sub-membranes will constitute an outer membrane. This process is repeated until the environment membrane is recognized. Finally, the simulator will simulate the recognized model and the result will be presented on the user interfaces.

3.3. UPSimulator functionalities

UPSimulator has two major functionalities. Firstly, UPSimulator will recognize and simulate the membrane classes and the environment defined in one or multiple structured text files. In a text file, users can define complex membrane structures and combined rules using the concepts summarized in Table 1. In addition, the mathematical operators and functions in Table 2 can be used to describe the relationships between the objects in one rule.

Table 2
Mathematical operators and functions in UPSimulator.

Classification	Operator or function
Binary arithmetic operators	$+ - * /$
Boolean logic operators	$< <= (\leq) ==$(equal) $! =(\neq) >= (\geq) >$
Generic functions	abs, sqrt, pow, log, log10
Trigonometric functions	sin, cos, tan

Furthermore, due to the modular architecture of the simulator, UPSimulator can be used as a runtime library in more complex membrane computing applications. By importing this library, the input variables of the application can be transformed to objects. Then, during the simulation, objects will be processed by using the rules within the system. After that, the resulting objects will be produced. Finally, the resulting objects are converted to obtain the output variables, and other commands are executed to complete the application or to continue adding new input variables.

4. Implementation and extensibility

4.1. Implementation of UPSimulator

UPSimulator is implemented in Java (1.8+). Since Java is a widely supported language, UPSimulator can be used in most operating systems, including Windows, OS X, Linux, and UNIX. Except a standalone Java platform, all the other dependencies are self-included in UPSimulator.

As shown in Section 3.2, UPSimulator consists of Membrane, Rule, Condition, Result, Object and Tunnel (all of them are defined as interfaces in the simulator). The internal relationships of these interfaces are illustrated in Fig. 2 using a UML diagram. Through these interfaces, the implementation of UPSimulator does not directly support specific P system model definitions (e.g. Cell-like P system, Tissue-like P system, Neural-like P system), but supports its basic concepts.

Now, UPSimulator have supported the main concepts in cell-like, tissue-like and neural-like P system, as shown in Table 1. This implementation method makes UPSimulator more flexible and easier to be maintained and extended.

4.2. Extensibility of UPSimulator

First, UPSimulator extends the model implemented by P-Lingua. P-Lingua is currently the most popular simulator in membrane computing, and it supports the models shown in Table 3 [31]. Except budding, Stochastic P systems and Simple kernel P systems, all the other concepts are supported by UPSimulator.

Table 3
Supported models in pLingua.

Classification	Model
Cell-like P system [16]	Active membranes with division rules Active membranes with creation rules Transition P systems Symport/antiport P systems Stochastic P systems (discontinued) Probabilistic P systems
Tissue-like P system [28,29]	Tissue P systems with symport/antiport rules and division rules Tissue P systems with cell separation rules
Neural-like P system [30]	Spiking neural P systems with division, budding and delays
Other P systems	Simple kernel P systems

Table 4
Software metadata (optional).

Nr.	(executable) Software metadata description	Please fill in this column
S1	Current software version	v2.5.0
S2	Permanent link to executables of this version	https://github.com/quancs/UPSimulator/releases/tag/v2.5.0
S3	Legal software license	GPL-3.0
S4	Computing platform/Operating System	BSD, Linux, OS X, Microsoft Windows, Unix-like
S5	Installation requirements & dependencies	Java 1.8+
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	https://github.com/quancs/UPSimulator
S7	Support email for questions	quancs@cqu.edu.cn

Table 5
Code metadata (mandatory).

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v2.5.0
C2	Permanent link to code/repository used of this code version	https://github.com/quancs/UPSimulator/releases/tag/v2.5.0
C3	Legal code license	GPL-3.0
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Java
C6	Compilation requirements, operating environments & dependencies	Java 1.8+
C7	If available Link to developer documentation/manual	https://quancs.github.io/UPSimulator/
C8	Support email for questions	quancs@cqu.edu.cn

Meanwhile, not all models supported by UPSimulator are supported by P-Lingua, for example, cell-like SN P system.

Second, the extensibility of UPSimulator comes from its implementation method. If all the concepts in a new P system model are supported in UPSimulator, then this model is supported by UPSimulator. That is, UPSimulator can support new models if they are generated by a combination of different concepts in UPSimulator. For example, by combining different concepts, UPSimulator support cSN P system and its variant cSN P system with request rule [19,20], tissue P systems with evolutional symport/antiport rules [21], the variants of SN P system with multiple channels and anti-spikes [23,24], and so on.

Third, UPSimulator can easily support other new concepts not included in UPSimulator. For example, if one wants to support the newly proposed concept polarization [32], a new polarization condition and a new polarization result are enough for UPSimulator to simulate the function of polarization in biological systems. The new condition and result can be easily created by implementing the five functions of the Condition and Result interfaces (see Fig. 2). Only few works are needed in this process. Once polarization is supported, it can be combined with other concepts to form more complex rule types.

5. Illustrative examples

This section gives an example of how to use UPSimulator. In Fig. 4(a), we applied the Dijkstra algorithm to find the shortest path from vertex 1 to vertex 5.

First, the definition of the membrane class SPM (Shortest-Path-Membrane) is given in Fig. 3 according to the Dijkstra algorithm. SPM is a general membrane class that can be used to find the shortest path in any directed graph. The membrane class SPM is saved as the file "spm.txt". In this file, we suppose:

- s[i] represents the start point i;
- end[i] represents the end point i;
- e[i][j][v] indicates that the weight of the known shortest path from point i to point j is equal to v;
- fp[i][j] indicates that the first point in the known shortest path from start point to point i is point j;
- The objects from *s0* to *s6* are used to control the process of parallel computing;

Fig. 4 shows a constructed environment membrane of a test graph that contains the start point 1, the end point 5 and all the edges of the graph.

Second, based on Fig. 4(a), the computing environment definition file "example.txt" is implemented as shown in Fig. 4(b).

Third, we use the following command or double-click the jar file (if the jar file is linked to JAVA) to start UPSimulator.

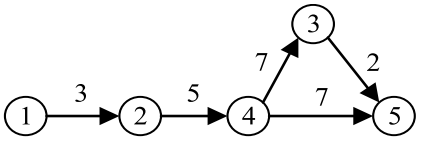
java -jar ./UPSimulator-2.4.8.jar

Now we can simulate the model in UPSimulator following the steps shown in Fig. 5. The results and processes of the simulation will be displayed in the result panel and process panel, respectively. The result of this example is shown in Fig. 6. The shortest path from point 1 to point 5 is $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$.

In Fig. 5, step 1 (Import) is to read the membrane class files and the computing environment definition files (e.g., 'spm.txt' for the membrane class file and 'example.txt' for the environment file). Step 2 (Environment Selection) is to select the computing environment definition file (e.g., 'example.txt') for this simulation.

```
Membrane SPM{
        Rule r1[i]=s[i] -> s0[i] e[i][i][0] fp[i][i] u[i];
        Rule r2[i][j][v]=e[i][j][v] -> e[i][j][v] fp[j][i] | @s[i];
        Rule r3[i]=s0[i] -> s1[i];
        Rule r4[i][j][v]=e[i][j][v] -> e[i][j][v] t1[i][j][v] | @s0[i] & !u[j];
        Rule r5[i]=s1[i]->s2[i] c2;
        Rule r6=c2-> ;
        Rule r7[i][j1][v1][j2][v2]= t1[i][j1][v1] t1[i][j2][v2] -> t1[i][j1][v1] c2 | @s2[i] & @c2 & v1<v2;
        Rule r8[i]=s2[i]->s3[i] c| !c2;
        Rule r9[i][j][v]=t1[i][j][v] -> u[j] | @s3[i];
        Rule r10[i][j][k][v1][v2]=e[j][k][v2] -> e[j][k][v2] t2[i][k][j][v1+v2] c | @t1[i][j][v1] & @s3[i];
        Rule r11[i]=s3[i]->s4[i];
        Rule r12[i][j][k][m][v1][v2]=t2[i][k][j][v1] e[i][k][v2] fp[k][m] -> e[i][k][v1] fp[k][j] c | v1<v2;
        Rule r13[i][j][k][m][v1][v2]=t2[i][k][j][v1] e[i][k][v2] fp[k][m] -> e[i][k][v2] fp[k][m] c | v1>=v2;
        Rule r14[i]=s4[i]->s5[i];
        Rule r15[i][j][k][v]=t2[i][k][j][v] -> e[i][k][v] fp[k][j] c | @s5[i];
        Rule r16[i]=s5[i]->s0[i]@c;
        Rule r17[i]=s5[i]->s6[i]!c;
        Rule r18[i]=s6[i]->s0[i]|@c;
        Rule r19[i]=s6[i]->start[i] d |!c;
        Rule r20[i][j]=end[i] fp[i][j] -> end[j] edge[j][i] d | i!=j;
        Rule r21[i]=start[i] end[i] -> dissolve(all);
        Rule r22= c->;
        Rule r23[i]=start[i] -> dissolve(all) ( notexist ,out)| !d;
        Rule r24= d->;
        Rule r25[i][j][k]= edge[i][j] -> ( edge[i][j], out)| @start[k] & @end[k];
}
```

Fig. 3. An implementation of Dijkstra algorithm in P system.



(a) Test graph

```
Environment{
    Membrane SPM spm{
      Object s[1], end[5];
      Object e[1][2][3], e[2][4][5], e[4][3][7], e[4][5][7], e[3][5][2];
    }
}
```

(b) The environment definition

Fig. 4. An example of the environment definition.

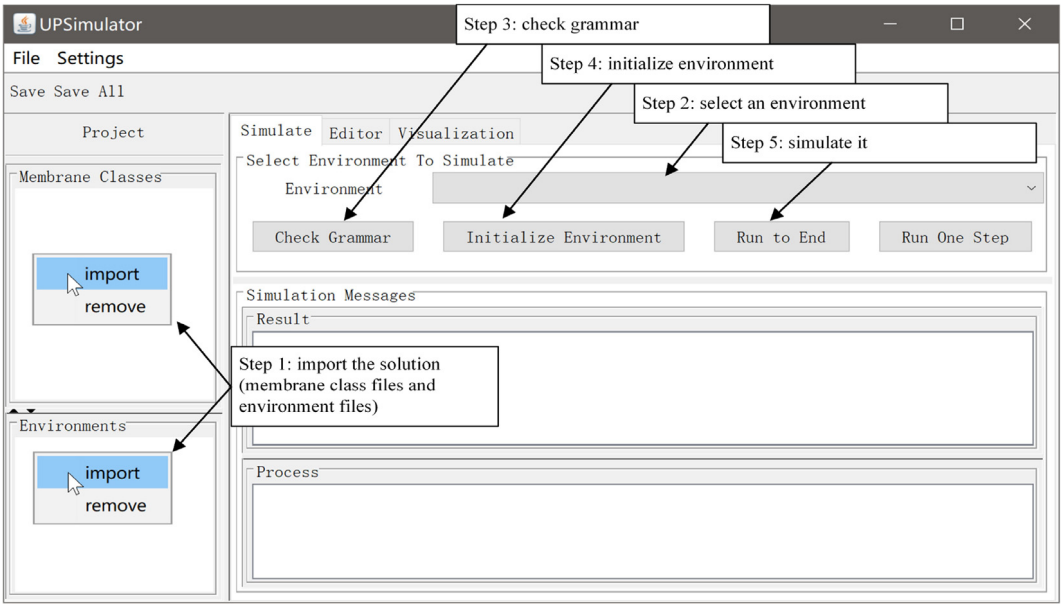


Fig. 5. The steps to use UPSimulator to simulate a model.

Step 3 is to perform lexical and grammar analysis on these files. Step 4 is to build the simulation environment. Step 5 is to perform the simulation and obtain the simulation results.

6. Conclusions

In this paper, a new general-purpose simulation tool UPSimulator and an object-oriented description language UPLanguage are proposed in the field of membrane computing. On the one hand, UPSimulator not only supports simple membrane structures, but also supports nested membrane structures and mesh membrane structures. On the other hand, UPSimulator not only supports simple evolution rule types, but also supports complex rule types that are combined with multiple concepts. In fact, UPSimulator and UPLanguage provide a new choice for the simulation of P system.

In the future, based on the flexible and extendable framework, UPSimulator will support more new concepts and new membrane structures through UPLanguage.

```
Environment {
   Object edge[1][2], edge[2][4], edge[4][5];
}
```

Fig. 6. The simulation result of the example in Fig. 4.

Appendix. Required metadata

Current executable software version

See Table 4.

Current code version

See Table 5.

References

[1] G. Păun, Computing with membranes, J. Comput. System Sci. 61 (1) (2000) 108–143.
[2] C. Martín-Vide, G. Păun, J. Pazos, A. Rodríguez-Patón, Tissue p systems, Theoret. Comput. Sci. 296 (2) (2003) 295–326.
[3] B. Song, Y. Hu, H.N. Adorna, F. Xu, A quick survey of tissue-like P systems, Romanian J. Inf. Sci. Technol. 21 (3) (2018) 310–321.
[4] M. Ionescu, G. Păun, T. Yokomori, Spiking neural P systems, Fundam. Inform. 71 (2–3) (2006) 279–308.
[5] J. Wang, H. Peng, Adaptive fuzzy spiking neural P systems for fuzzy inference and learning, Int. J. Comput. Math. 90 (4) (2013) 857–868.
[6] P. Guo, H.Z. Chen, H. Zheng, Arithmetic expression evaluations with membranes, Chin. J. Electron. 23 (1) (2014) 55–60.
[7] L. Ye, P. Guo, Design a membrane system for matrix multiplication, Int. J. Light Electron Opt. 127 (20) (2016) 8231–8239.
[8] A. Adrian, Arithmetic with membranes, Romanian J. Inf. Sci. Technol. 4 (1–2) (2001) 5–20.
[9] H. Peng, J. Wang, rez Jim, M.J. Nez, N. Riscos, A. Ez, An unsupervised learning algorithm for membrane computing, Inform. Sci. 304 (2015) 80–91.
[10] M. Cardona, M.A. Colomer, A. Zaragoza, M.J. Pérez-Jiménez, Hierarchical clustering with membrane computing, Comput. Inform. 27 (3) (2012) 497–513.
[11] L. Ye, P. Guo, An immune algorithm based on p system for classification, Commun. Comput. Inform. Sci., Springer 681 (1) (2016) 133–141.
[12] G. Ping, Z. Jian, C. Haizhu, Y. Ruilong, A linear-time solution for all-SAT problem based on P system, Chin. J. Electron. 27 (2) (2018) 367–373.
[13] G. Zhang, M. Gheorghe, J. Cheng, An approximate algorithm combining P systems and ant colony optimization for traveling salesman problems, in: Proceedings of the Eighth Brainstorming Week on Membrane Computing, Sevilla, E.T.S. de Ingeniería Informática, 2010, pp. 321–340.
[14] P. Guo, Y. Dai, H. Chen, A p system for Hamiltonian cycle problem, Int. J. Light Electron Opt. 127 (20) (2016) 8461–8468.
[15] M. Ali, C. Ravie, Enhancing the simulation of membrane system on the GPU for the N-queens problem, Chin. J. Electron. 24 (4) (2015) 740–743.
[16] M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, An overview of P-lingua 2.0, in: G. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing, Vol. 5957, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 264–288.
[17] 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: IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications, Liverpool, UK, 2010, pp. 637–643.
[18] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, D. Ramírez-Martínez, A software tool for verification of spiking neural P systems, Nat. Comput. 7 (4) (2008) 485–497.
[19] T. Wu, Z. Zhang, G. Păun, L. Pan, Cell-like spiking neural p systems, Theoret. Comput. Sci. 623 (2016) 180–189.
[20] L. Pan, T. Wu, Y. Su, A.V. Vasilakos, Cell-like spiking neural p systems with request rules, IEEE Trans. NanoBiosci. 16 (6) (2017) 513–522.
[21] B. Song, C. Zhang, L. Pan, Tissue-like P systems with evolutional symport/antiport rules, Inform. Sci. 378 (2017) 177–193.
[22] L. Pan, B. Song, L. Valencia-Cabrera, M.J. Pérez-Jiménez, The computational complexity of tissue p systems with evolutional symport/antiport rules, Complexity 2018 (2018).
[23] H. Peng, et al., Spiking neural P systems with multiple channels, Neural Netw. 95 (2017) 66–71.
[24] X. Song, et al., Spiking neural P systems with multiple channels and anti-spikes, Biosystems 169–170 (2018) 13–19.
[25] H. Peng, J. Wang, M.J. Pérez-Jiménez, A. Riscos-Núñez, Dynamic threshold neural P systems, Knowl.-Based Syst. 163 (2019) 875–884.
[26] H. Peng, J. Wanñ, Coupled neural p systems, IEEE Trans. Neural Netw. Learn. Syst. (2018) 1–11.
[27] T.J. Parr, R.W. Quong, ANTLR: A predicated-lL(k) parser generator, Softw. Pract. Exp. 25 (7) (1995) 789–810.
[28] M.A. Martínez-del Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, A P-lingua based simulator for tissue P systems, J. Log. Algebr. Program. 79 (6) (2010) 374–382.
[29] I. Perez-Hurtado, L. Valencia-Cabrera, J.M. Chacon, A. Riscos-Nunez, M.J. Perez-Jimenez, A P–lingua based simulator for tissue P systems with cell separation, Romanian J. Inf. Sci. Technol. 17 (1) (2014) 89–102.
[30] L.F. Macías-Ramos, I. Pérez-Hurtado, M. García-Quismondo, L. Valencia-Cabrera, M.J. Pérez-Jiménez, A. Riscos-Núñez, A P–lingua based simulator for spiking neural P systems, in: M. Gheorghe, G. Păun, G. Rozenberg, A. Salomaa, S. Verlan (Eds.), Membrane Computing, Vol. 7184, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 257–281.
[31] Supported models - The P-Lingua Website. [Online]. Available: http://www.p-lingua.org/wiki/index.php/Supported_models.
[32] T. Wu, A. Paun, Z. Zhang, L. Pan, Spiking neural P systems with polarizations, IEEE Trans. Neural Netw. Learn. Syst. 29 (8) (2018) 3349–3360.