Tenth Brainstorming Week on Membrane Computing

Sevilla, January 30 – February 3, 2012

Volume II

Manuel García-Quismondo
Luis F. Macías-Ramos
Gheorghe Păun
Luis Valencia-Cabrera
Editors

Tenth Brainstorming Week on Membrane Computing

Sevilla, January 30 – February 3, 2012

Volume II

Manuel García-Quismondo Fernández
Luis Felipe Macías Ramos
Gheorghe Păun
Luis Valencia Cabrera
Editors

RGNC REPORT 2/2012

Research Group on Natural Computing
Sevilla University

Fénix Editora, Sevilla, 2012

©Autores
ISBN: ??????
Depósito Legal: SE-????–06
Edita: Fénix Editora
Avda. de Cádiz, 7 – 1C
41004 Sevilla
fenixeditora@telefonica.net
Telf. 954 41 29 91

Preface

These proceedings, consisting in two volumes, contain the papers emerged from the Tenth Brainstorming Week on Membrane Computing (BWMC), held in Sevilla, from January 30 to February 3, 2012, in the organization of the Research Group on Natural Computing from the Department of Computer Science and Artificial Intelligence of Sevilla University. The first edition of BWMC was organized at the beginning of February 2003 in Rovira i Virgili University, Tarragona, and all the next editions took place in Sevilla at the beginning of February, each year.

The 2012 edition of BWMC was organized in conjunction with the First International Conference on Developments in Membrane Computing (ICDMC2012).

In the style of previous meetings in this series, the tenth BWMC was conceived as a period of active interaction among the participants, with the emphasis on exchanging ideas and cooperation. Several "provocative" talks were delivered, mainly devoted to open problems, research topics, conjectures waiting for proofs, followed by an intense cooperation among the 40 participants – see the list in the end of this preface. The efficiency of this type of meetings was again proved to be very high and the present volumes illustrate this assertion.

Slightly different from the previous meetings was the combination with the ICDMC2012, in the sense that several talks also had the style of a conference: more time dedicated to presenting achievements obtained by the research groups from where the participants came from, but also converging towards the style of the brainstorming, i.e., presenting frontier results, research topics, ongoing applications.

The papers included in these volumes, arranged in the alphabetic order of the authors, were collected in the form available at a short time after the brainstorming; several of them are still under elaboration. The idea is that the proceedings are a working instrument, part of the interaction started during the stay of authors in Sevilla, meant to make possible a further cooperation, this time having a written support.

Selections of the papers from these volumes will be considered for publication in special issues of *Theoretical Computer Science* and of *International Journal of Computer Mathematics*.

After each BWMC, one or two special issues of various international journals were published. Here is their list:

- BWMC 2003: *Natural Computing* – volume 2, number 3, 2003, and *New Generation Computing* – volume 22, number 4, 2004;
- BWMC 2004: *Journal of Universal Computer Science* – volume 10, number 5, 2004, and *Soft Computing* – volume 9, number 5, 2005;
- BWMC 2005: *International Journal of Foundations of Computer Science* – volume 17, number 1, 2006);
- BWMC 2006: *Theoretical Computer Science* – volume 372, numbers 2-3, 2007;
- BWMC 2007: *International Journal of Unconventional Computing* – volume 5, number 5, 2009;
- BWMC 2008: *Fundamenta Informaticae* – volume 87, number 1, 2008;
- BWMC 2009: *International Journal of Computers, Control and Communication* – volume 4, number 3, 2009;
- BWMC 2010: *Romanian Journal of Information Science and Technology* – volume 13, number 2, 2010;
- BWMC 2011: *International Journal of Natural Computing Research* – volume 2, numbers 2-3, 2011.

Other papers elaborated during the tenth BWMC will be submitted to other journals or to suitable conferences. The reader interested in the final version of these papers is advised to check the current bibliography of membrane computing available in the domain website `http://ppage.psystems.eu`.

***

The list of participants as well as their email addresses are given below, with the aim of facilitating the further communication and interaction:

1. Artiom Alhazov, University of Milano - Bicocca, Italy, *aartiom@yahoo.com*
2. Ioan Ardelean, Institute of Biology of the Romanian Academy, Bucharest, Romania, *ioan.ardelean57@yahoo.com*
3. Mari Angels Colomer Cugat, University of Lleida, Spain, *colomer@matematica.udl.cat*
4. Erzsébet Csuhaj-Varjú, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary, *csuhaj@inf.elte.hu*
5. Rudolf Freund, Technological University of Vienna, Austria, *rudifreund@gmx.at*
6. Manuel García-Quismondo Fernández, University of Seville, Spain, *mgarciaquismondo@us.es*
7. Marian Gheorghe, University of Sheffield, United Kingdom, *m.gheorghe@sheffield.ac.uk*
8. Carmen Graciani Díaz, University of Seville, Spain, *cgdiaz@us.es*

9. Miguel A. Gutiérrez Naranjo, University of Seville, Spain, *magutier@us.es*
10. Florentin Ipate, University of Piteşti, Romania, *florentin.ipate@ifsoft.ro*
11. Jozef Kelemen, Silesian University, Opava, Czech Republic, *jozef.kelemen@fpf.slu.cz*
12. Abhay Krishna, CABIMER, Seville, Spain, *Abhay.Krishan@cabimer.es*
13. Raluca Lefticaru, University of Piteşti, Romania, *raluca.lefticaru@gmail.com*
14. Alberto Leporati, University of Milano - Bicocca, Italy, *leporati@disco.unimib.it*
15. Luis Felipe Macías Ramos, University of Seville, Spain, *lfmaciasr@us.es*
16. Vincenzo Manca, University of Verona, Italy, *vincenzo.manca@univr.it*
17. Luca Marchetti, University of Verona, Italy, *luca.marchetti@univr.it*
18. Miguel A. Martínez del Amor, University of Seville, Spain, *mdelamor@us.es*
19. Giancarlo Mauri, University of Milano - Bicocca, Italy, *mauri@disco.unimib.it*
20. Adam Obtulowicz, Polish Academy of sciences, Warsaw, Poland, *A.Obtulowicz@impan.pl*
21. Ana Brânduşa Pavel, Politehnica University of Bucharest, Romania, *anabrandusa@gmail.com*
22. Gheorghe Păun, Romanian Academy, Bucharest, Romania, and University of Seville, Spain, *gpaun@us.es*
23. Hong Peng, School of Mathematics and Computer Engineering, Xihua University, China, *ph.xhu@hotmail.com*
24. Ignacio Pérez Hurtado de Mendoza, University of Seville, Spain, *perezh@us.es*
25. Mario de J. Pérez Jiménez, University of Seville, Spain, *marper@us.es*
26. Antonio Enrico Porreca, University of Milano - Bicocca, Italy, *porreca@disco.unimib.it*
27. Raúl Reina Molina, University of Seville, Spain, *m75@gmail.com*
28. Agustín Riscos Núñez, University of Seville, Spain, *ariscosn@us.es*
29. Iurie Rogojin, Institute of Mathematics and Computer Science of the Academy of Sciences of Moldova, Chişinău, Moldova, *yrogozhin@gmail.com*
30. Álvaro Romero Jiménez, University of Seville, Spain, *romero.alvaro@us.es*
31. Francisco José Romero Campero, University of Seville, Spain, *fran@us.es*
32. Jose María Sempere Luna, Polytechnical University of Valencia, Spain, *jsempere@dsic.upv.es*
33. Petr Sosík, Silesian University, Opava, Czech Republic, and Universidad Politécnica de Madrid, Spain, *petr.sosik@fpf.slu.cz*
34. Cristian Ştefan, University of Piteşti, Romania, *liviu.stefan@yahoo.com*
35. Luis Valencia Cabrera, University of Seville, Spain, *lvalencia@us.es*
36. György Vaszil, Faculty of Informatics, University of Debrecen, Hungary, *vaszil.gyorgy@inf.unideb.hu*
37. Serghei Verlan, University of Paris Est, France, *verlan@univ-paris12.fr*
38. Jun Wang, Electrical and Information Engineering, Xihua University, China, *wj.xhu@hotmail.com*
39. Claudio Zandron, University of Milano - Bicocca, Italy, *zandron@disco.unimib.it*

40. Gexiang Zhang, School of Electrical Engineering, Southwest Jiaotong University, China, *gexiangzhang@gmail.com*

As mentioned above, the meeting was organized by the Research Group on Natural Computing from Sevilla University (`http://www.gcn.us.es`)– and all the members of this group were enthusiastically involved in this (not always easy) work.

The meeting was supported from various sources: (i) Proyecto de Excelencia con investigador de reconocida valía, de la Junta de Andalucía, grant P08 – TIC 04200, (ii) Proyecto del Ministerio de Educación y Ciencia, grant TIN2009 – 13192, (iii) Instituto de Matemáticas de la Universidad de Sevilla (IMUS), (iv) Consejería de Innovacion, Ciencia y Empresas de la Junta de Andalucía, as well as by the Department of Computer Science and Artificial Intelligence from Sevilla University.

Gheorghe Păun
Mario de Jesús Pérez-Jiménez
(Sevilla, May 3, 2012)

Contents

Contents of Volume I

Inverse Dynamical Problems: An Algebraic Formulation Via MP Grammars

Vincenzo Manca and Luca Marchetti

University of Verona, Department of Computer Science
Strada Le Grazie 15, 37134 Verona, Italy
vincenzo.manca@univr.it luca.marchetti@univr.it

Summary. Metabolic P grammars are a particular class of multiset rewriting grammars introduced in the MP systems' theory for modelling metabolic processes. In this paper, a new algebraic formulation of inverse dynamical problems, based on MP grammars and Kronecker product, is given, for further motivating the correctness of the LGSS (Log-gain Stoichiometric Stepwise) algorithm, introduced in 2010s for solving dynamical inverse problems in the MP framework. At the end of the paper, a section is included that introduces the problem of multicollinearity, which could arise during the execution of LGSS, and that defines an algorithm, based on a hierarchical clustering technique, that solves it in a suitable way.

Key words: Metabolic P systems, dynamical systems, dynamical inverse problems, Kronecker product, stepwise regression.

1 Introduction

Metabolic P (MP) systems are a particular class of cell-like P systems [33, 34, 36, 35] introduced by Vincenzo Manca in 2004, for modelling metabolic processes [29]. An MP system is essentially a particular type of deterministic discrete dynamical system which inherits from the P systems' framework a native similitude with the functioning of a living cell.

MP systems share with P systems the multiset rewriting mechanism as their fundament. However, while P systems are essentially unconventional computational models, MP systems are intended to generate dynamics instead of computations. Namely, their aim in modelling biological phenomena is that of finding the multiset rewriting mechanism underlying an observed biological behaviour.

Metabolic P systems can be considered as the result of a research activity initiated in 1990s with some initial works [15, 28, 30]. They are different, with respect to other "P variants" applied in the context of systems biology [3, 4, 6, 38, 39]. The main difference is in their determinism. In fact, their basis are *MP grammars*, where multiset transformations are regulated by functions in a

deterministic way [19]. An MP system is an MP grammar equipped with a *temporal interval* τ, a conventional *mole size* ν, and substances masses, which specify the time and population (discrete) granularities respectively [19].

An MP grammar G can be considered as a generator of time series, determined by the following structure ($n, m \in \mathbb{N}$, the set of natural numbers):

$$G = (M, R, I, \Phi)$$

where:

1. $M = \{x_1, x_2, \ldots, x_n\}$ is a finite set of elements called *metabolites*, or *substances*. A *metabolic state* is given by a list of n values, each of which is associated to a metabolite.
2. $R = \{\alpha_j \to \beta_j \mid j = 1, \ldots, m\}$ is a set of *rules*, or *reactions*, with α_j and β_j multisets over M for $j = 1, \ldots, m$.
3. I are *initial values* of metabolites, that is, a list $x_1[0], x_2[0], \ldots, x_n[0]$ providing the *metabolic state at step 0*.
4. $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ is a list of functions, called *regulators*, one for each rule, such that, for $1 \leq j \leq m$, and for some k_j $(0 \leq k_j \leq n)$

$$\varphi_j : \mathbb{R}^{k_j} \to \mathbb{R}.$$

An MP grammar G is *parametric*, when a set P of parameters is added to G, and metabolic states include also elements of P (to which, the state assigns real values), therefore regulators may include parameters as their arguments . If G is parametric, also the time series of parameters has to be provided in order to specify G.

An MP grammar can be easily representable by an *MP graph* [22]. Moreover, the set of the rules of the system can be also represented by a *stoichiometric matrix* $\mathbb{A}$, which gives a sort of "matrix-like representation" of the stoichiometry (see Figure 1).

An MP grammar G defines, for any $x \in M$, a time series

$$(x[i] \mid i \in \mathbb{N}, i > 0)$$

in the following way. Let

$$s[i] = (x_1[i], x_2[i], \ldots, x_n[i])$$

the (row) *state vector* of G at step i, which can be seen as a function from the set of metabolites to $\mathbb{R}$, then the flux $\varphi_j(s_j[i])$ of rule r_j at step i, is given by applying the regulator φ_j to $s_j[i]$, a substate of $s[i]$ associated to r_j, and constituted by k_j components called the *tuners* of r_j.

If we consider the rule r_2 of the MP grammar given in Figure 1, for example, then the flux at step i is calculated by:

$$\begin{aligned} \varphi_2(s_2[i]) &= \varphi_2(A[i], B[i]) \\ &= c_2 A[i]^2 + c_3 B[i]^3 \end{aligned}$$

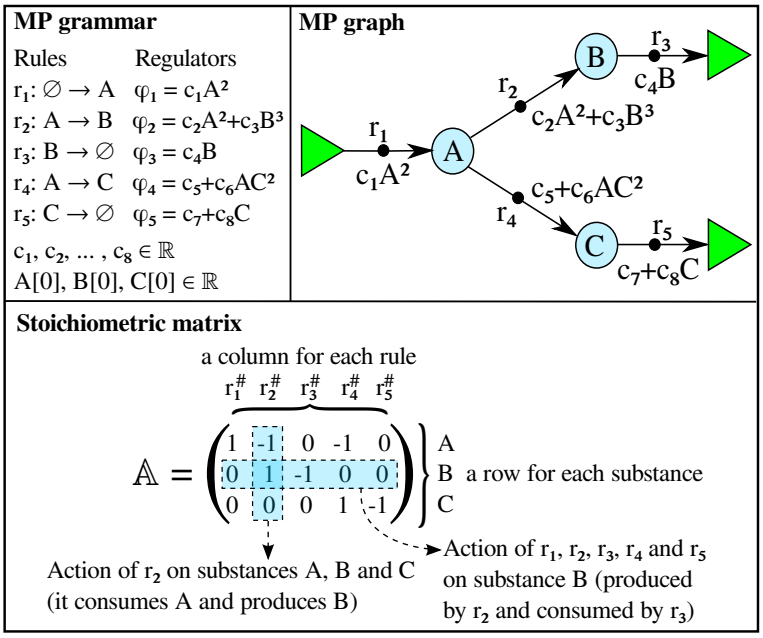


Fig. 1. An example of MP grammar (where $\emptyset$ denotes an empty multiset and substance symbols occurring in regulators denote the corresponding substance quantities), the stoichiometric matrix $\mathbb{A}$ is directly deduced by the MP grammar on the top left corner. The MP graph on the top right corner is obtained by translating the rules in the source-target-edge notation [26].

where c_2, c_3 are given real constants, and A and B are said to be the tuners of the rule r_2.

The value of $x[i+1]$, for each $x \in M$, is given by the following equation, where $\alpha_j(x)$ and $\beta_j(x)$ denote the multiplicities of x in the multiset α_j and β_j, respectively:

$$x[i+1] = x[i] + \sum_{j=1}^{m} [(\beta_j(x) - \alpha_j(x)) \cdot \varphi_j(s_j[i])].$$

More generally, if we denote by $\mathbb{A}$ the stoichiometric matrix of the system and by

$$\Phi[i] = (\varphi_1(s_1[i]), \varphi_2(s_2[i]), \ldots, \varphi_m(s_m[i]))$$

the row vector of fluxes at step i, it can be proved that [16]:

$$(s[i+1] - s[i])^T = \mathbb{A} \times \Phi^T[i] \tag{1}$$

that is, by transposition:

$$s[i+1] - s[i] = \Phi[i] \times \mathbb{A}^T. \tag{2}$$

These last two equations define equivalently the *Equational Metabolic Algorithm (EMA)*. In the following, the MP dynamics we will present are computed in MATLAB[1] by applying EMA. We refer to [19, 20, 18, 21] for a comprehensive presentation of the MP theory.

The dynamics which can be modelled by MP systems can be very complicated even by considering simple MP grammars (i.e. with few substances and linear regulators). In [23] MP systems were successfully applied to the field of real periodical function approximation. The complexity of the dynamics compared to the simplicity of the MP grammar which calculates it by EMA, suggests that MP system theory can be a suitable framework for modelling biological dynamics.

The procedure introduced in [23] to define the models has been widely extended in [25, 26] for defining the *LGSS (Log-Gain Stoichiometric Stepwise)* algorithm, which derives MP grammars generating time series of observed dynamics. LGSS can be applied independently from any knowledge about reaction rate kinetics and it represents the most recent solution, in terms of MP systems, of the *dynamical inverse problem*, that is, of the identification of (discrete) mathematical models of an observed dynamics and satisfying all the constraints required by the specific knowledge about the modelled phenomenon. The LGSS algorithm combines and extends the log-gain principles developed in the MP system theory [16, 17] with the classical method of Stepwise Regression [7], which is a statistical regression technique based on Least Squares Approximation and statistical F-tests [5].

LGSS has been implemented by Luca Marchetti in 2010 as a set of MATLAB functions. We refer to [24, 31, 32, 27] for some successful applications of LGSS and MP systems for discovering the internal regulation logic of phenomena relevant in systems biology.

The starting point of the LGSS algorithm was the search for the right regulators associated to the reactions of an MP grammar which provide the observed time series when dynamics is computed by means of EMA. If we consider the role of each regulator, we realize that it affects the variations of many substances. Therefore regulators are constrained to satisfy altogether, at each step, an algebraic system based on the stoichiometry of the observed phenomenon. The crucial point for regulator determination was a special kind of regression formulated as "stoichiometric expansion" of EMA by means of an initial set of basic functions called regressors.

In the next section we will introduce a new algebraic formulation of the stoichiometric expansion, based on MP grammars and Kronecker product, which better describes and motivates its adoption in LGSS for solving inverse dynamical problems.

[1] See `http://www.mathworks.it/index.html` for details on the MATLAB software.

2 Stoichiometric expansion

Given a system with n variables $x_1, x_2, \ldots, x_n$, let us suppose to know the time series of these variables along time points $0, 1, \ldots, t$. Let

$$s[i] = (x_1[i], x_2[i], \ldots, x_n[i])$$

the (row) state vector at time i, and

$$x_j[i+1] - x_j[i] = \Delta_j[i]$$

for $j = 1, 2, \ldots, n$, then

$$s[i+1] - s[i] = (\Delta_1[i], \Delta_2[i], \ldots, \Delta_n[i])$$

whence, from equation (2), we get

$$\Phi[i] \times \mathbb{A}^T = (\Delta_1[i], \Delta_2[i], \ldots, \Delta_n[i]). \tag{3}$$

For the determination of the regulators which provide the best approximate solution of the system (3), which has m unknowns (the m components of the flux vector $\Phi[i]$), LGSS applies a procedure called *stoichiometric expansion*. Let us assume that the regulators we are searching for can be expressed as linear combinations of some basic regressors $g_1, g_2, \ldots, g_d$ which usually include constants, powers, and products of substances, plus some basic functions which are considered suitable in the specific cases under investigation:

$$\begin{aligned}
\varphi_1 &= c_{1,1}g_1 + c_{1,2}g_2 + \ldots + c_{1,d}g_d \\
\varphi_2 &= c_{2,1}g_1 + c_{2,2}g_2 + \ldots + c_{2,d}g_d \\
\ldots &= \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\varphi_m &= c_{m,1}g_1 + c_{m,2}g_2 + \ldots + c_{m,d}g_d.
\end{aligned} \tag{4}$$

Let us consider the *t-expansion* $\Phi_1^t, \Phi_2^t, \ldots, \Phi_m^t$ of regulators as the vectors constituted by the right members of equations (4) evaluated along t steps (where the values of all the variables of the system are supposed to be known):

$$\begin{aligned}
\Phi_1^t &= c_{1,1}G_1^t + c_{1,2}G_2^t + \ldots + c_{1,d}G_d^t \\
\Phi_2^t &= c_{2,1}G_1^t + c_{2,2}G_2^t + \ldots + c_{2,d}G_d^t \\
\ldots &= \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\Phi_m^t &= c_{m,1}G_1^t + c_{m,2}G_2^t + \ldots + c_{m,d}G_d^t.
\end{aligned} \tag{5}$$

Now, let $C_1^d, C_2^d, \ldots, C_m^d$ be the unknown column vectors, of dimension d, constituted by the coefficients of the regressors providing the linear combinations of regulators $\varphi_1, \varphi_2, \ldots, \varphi_m$ we are searching for, and

$$\mathbb{C} = (C_1^d, C_2^d, \ldots, C_m^d)$$

the matrix having these vectors as columns. Moreover, let $\Delta_1^t, \Delta_2^t, \ldots, \Delta_n^t$ be the column vectors of dimension t constituted by substance variations of substances, from step i to step $i+1$, for $0 \leq i \leq t-1$, and

$$\Delta = (\Delta_1^t, \Delta_2^t, \ldots, \Delta_n^t)$$

the matrix having these vectors as columns. Let also Φ^t be the following matrix constituted by m column vectors of t elements:

$$\Phi^t = (\Phi_1^t, \Phi_2^t, \ldots, \Phi_m^t).$$

Finally, let

$$\mathbb{G} = (G_1^t, G_2^t, \ldots, G_d^t)$$

the matrix, of dimension $t \times d$, having as columns the vectors obtained by evaluating the regressors $g_1, g_2, \ldots, g_d$ on the t observed time points. With the notation above, the system of equations (5) becomes:

$$\mathbb{G} \times \mathbb{C} = \Phi^t. \tag{6}$$

Now, it easily follows from (3) that:

$$\Phi^t \times \mathbb{A}^T = \Delta \tag{7}$$

where the exponent T denotes the matrix transposition. Therefore, by combining equations (6) and (7), we finally obtain the *t-expansion* of the system (3) as:

$$\mathbb{G} \times \mathbb{C} \times \mathbb{A}^T = \Delta. \tag{8}$$

The coefficients of $\mathbb{C}$ are the unknowns which needs to be estimated by LGSS. We show now that they can be obtained by a Least Square Estimation deduced by equation (8), by using *direct product* $\otimes$ between matrices, also called *Kronecker product* [9, 10, 40, 41], which results a special case of *tensor product* used in linear algebra and in mathematical physics.

Given two real matrix A, B of dimension $n \times m$ and $t \times d$ respectively, then the *direct product*:

$$A \otimes B$$

is the matrix, of dimension $nt \times md$, constituted by nm blocks $B_{i,j}$, such that, if $A = (a_{i,j} \mid 1 \leq i \leq n,\ 1 \leq j \leq m)$, then $B_{i,j} = a_{i,j}B$ (in $B_{i,j}$ all the elements of B are multiplied by $a_{i,j}$, see Figure 2).
The Kronecker product is bilinear and associative, that is, it satisfies the following equations:

$$\begin{aligned}
A \otimes (B + C) &= (A \otimes B) + (A \otimes C) \\
(A + B) \otimes C &= (A \otimes B) + (A \otimes C) \\
(kA) \otimes B &= A \otimes (kB) = k(A \otimes B) \\
(A \otimes B) \otimes C &= A \otimes (B \otimes C).
\end{aligned}$$

$$\begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \otimes \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} = \begin{pmatrix} a\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} & b\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} & c\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \\ d\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} & e\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} & f\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \end{pmatrix}$$

Fig. 2. An example of Kronecker product of two matrices.

Moreover, matrix direct product verifies also the following equations:

$$\begin{aligned} (A \otimes B) \times (C \otimes D) &= (A \times C) \otimes (B \otimes D) \\ (A \otimes B)^T &= A^T \otimes B^T \\ (A \otimes B)^{-1} &= A^{-1} \otimes B^{-1} \end{aligned}$$

where the exponent T denotes transposition and the last equation holds only when the involved matrices are invertible.

Let us denote by $vec(W)$ the *vectorization* of the matrix W, obtained by concatenating in a unique column vector all the columns of W in their order. Then, a general property of matrix direct product asserts that [10]:

$$A \times X \times B = Y \quad iff \quad (B^T \otimes A) \times vec(X) = vec(Y). \tag{9}$$

Therefore, if we apply equivalence (9) to equation (8) we obtain:

$$(\mathbb{A} \otimes \mathbb{G}) \times vec(\mathbb{C}) = vec(\Delta) \tag{10}$$

where the *stoichiometric matrix* is multiplied, by Kronecker product, with the *regressor matrix* and the result is multiplied with the vectorization of the *regressor coefficient matrix*, and then equated to the vectorization of the *substance variation matrix*, by providing nt equations with md unknown values. The system of equations given in (10) is the *stoichiometric expanded system* calculated by LGSS.

According to the Least Square approximation method [43, 13], if $nt \geq md$, then the best approximation to $vec(C)$, minimizing the difference between the two members of equation (10), is given by the following vector:

$$\left((\mathbb{A} \otimes \mathbb{G})^T \times (\mathbb{A} \otimes \mathbb{G})\right)^{-1} \times (\mathbb{A} \otimes \mathbb{G})^T \times vec(\Delta). \tag{11}$$

Some constraints may be imposed to the fluxes provided by regulators, which may be of general nature, or may be specific to some classes of systems (for example, fluxes should not be negative, and the sum of fluxes of all reactions consuming a substance x cannot exceed the quantity of x). In Figure 3 are represented the regressor matrix $\mathbb{G}$ and the substance variation matrix Δ which are used by LGSS for least-squares approximating the coefficients $c_1, \ldots, c_8$ of the MP grammar given in Figure 1.

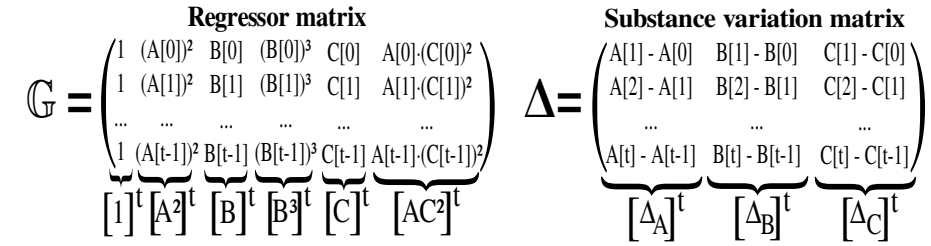


Fig. 3. The regressor matrix $\mathbb{G}$ and the substance variation matrix Δ used for approximating the coefficients $c_1, c_2, \ldots, c_8$ of the MP grammar given in Figure 1.

However, the approximation given by (11) cannot in general be considered the best way for solving the inverse dynamical problem. In fact, apart the computational cost of considering all the d regressors at same time, several reasons suggest to follow a gradual strategy in the determination of a subset of regressors and their corresponding coefficient which provide the best approximation to the given dynamics. There are two main requirements which are essential for an appropriate application of least squares method: the linear independence among the regressor expansions and the parsimony of the set of regressors. In other words, the best approximation is obtained by determining a *parsimonious* set of linearly independent regressors ensuring an error under a given threshold.

Linear independence is a requirement of least squares method and is solved by considering systems of equations which have been stoichiometric expanded. The parsimony of the model, instead, avoids problems of *overfitting*. In fact, the more regressors are considered in the model, the less is the degree of freedom left for the error [1]. This implies that solution fits very well with the dynamics on the observation points, but it is too constrained to them for behaving in a satisfactory way outside them (i.e. the model fits well the data, but it has not predictive power, see Figure 4 for an example).

In order to cope with the requirements explained above, LGSS integrates the least squares approximation of stoichiometric expanded systems with a regression strategy based on a step-wise approach as defined in [26]. Such kind of approach permits to define the model, step by step, by inserting into the model only those expanded regressors (among the columns of the matrix given by the direct product $\mathbb{A} \otimes \mathbb{G}$) which satisfy specific statistical tests. In this way, we can obtain MP models which fit the dynamics and that comprehends a small set of regressors.

3 Problems related to the regression in LGSS

The stepwise approach adopted in LGSS is based on the assumptions which are at the basis of the classical multiple regression model [1]. These assumptions concern with some properties of the expanded regressors (i.e. they must be linearly

independent and, possibly, not correlated[2]) and with the probability distribution of the errors associated to observations in considered time series (i.e. the errors should be normally distributed with mean zero). When one or more of these assumptions are not completely satisfied, some mistakes can occur in the definition of the regulators. In particular, there are several problems which we need to be aware of in the context of multiple regression. Some of them have been discussed in [26] and can be solved by substituting the ordinary least squares with other estimation methods based on the *weighted least squares* [42] or on the *generalized least squares* [12].

Here we focus on solving the problem of multicollinearity, which consists in having regressors that are highly correlated among them. This is the most common problem occurring in LGSS and also one of the most difficult to be solved [1]. When we develop a new MP model, we hope to have a strong correlation between each expanded regressor and the dependent variable $vec(\Delta)$, but we do not want to have expanded regressors correlated among them. In fact, this phenomenon may cause errors in the selection of the right set of regressors during the execution of the stepwise regression. In the case of perfect collinearity, the regression algorithm breaks down completely (because the matrix given by the direct product $\mathbb{A} \otimes \mathbb{G}$ has not maximum rank). Since in LGSS usually regulators are assumed to be linear combinations of polynomial regressors, then it is very common to meet multicollinearity problems.

[2] The correlation between regressors is intended to be calculated by means of the *Pearson's correlation coefficient* [37], which ranges from -1 to 1 and provides a measure of dependence between the behaviours of two magnitudes (-1: perfect anti-correlation; 0: no correlation; 1: perfect correlation).

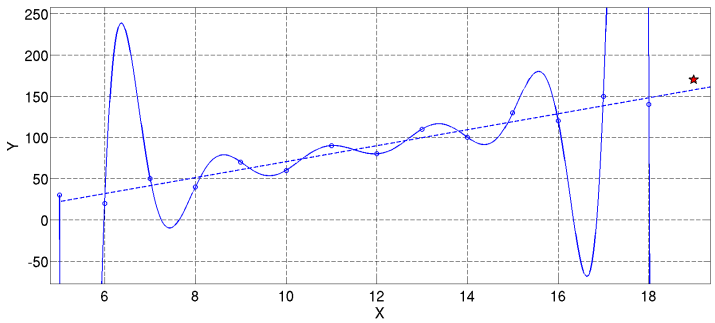


Fig. 4. Comparison between the predictive power of two regression models: a 13-degree polynomial $\widehat{Y} = c_0 + c_1 X + c_2 X^2 + \ldots + c_{13} X^{13}$ (depicted by the continuous line) and a least squares line (depicted by the dotted line). The dataset used to calculate the models are the 14 points depicted as blue circles, the last point represented by the red star is the value of Y we want to predict with our models. The 13-degree polynomial is a perfect example of model which overfits the data: in fact, it provides a perfect fit for all the points of the dataset, but it completely fails the prediction of Y in the 15th data point.

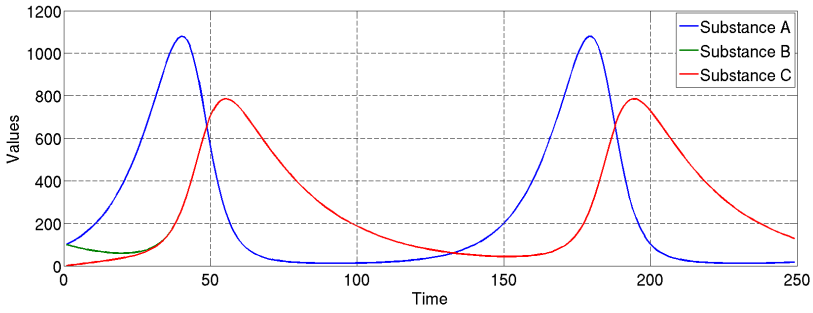


Fig. 5. Sirius' dynamics.

As an example, let us consider the dynamics given in Figure 5 related to a synthetic oscillator, introduced in [16] and very often considered in the MP theory, called *Sirius*. The oscillator is made of three substances A, B and C and five reactions whose regulators are supposed to depend on the set of substances given in Table 1.

For the metabolic oscillator Sirius, we want to apply LGSS for discovering the formulae of the regulators by assuming that they will be given by linear combinations of polynomial functions on substance quantities of degree less than or equal to 3. Since we do not know the right set of regressors which will be used, we should start LGSS by considering the set of all possible regressors given in Table 2. If we do this, however, the problem of multicollinearity arises since there

Set of the reactions	Dependence of the corresponding regulators
$r_1 : \emptyset \to A$	A, B and C
$r_2 : A \to B$	A and C
$r_3 : A \to C$	A and B
$r_4 : B \to \emptyset$	only B
$r_5 : C \to \emptyset$	only C

Table 1. The reactions (and the corresponding tuners) of the Sirius oscillator.

r_1 : Constant, A, B, C, A^2, B^2, C^2, AB, AC, BC, A^3, B^3, C^3, A^2B, A^2C, B^2C, AB^2, AC^2, BC^2, ABC.
r_2 : Constant, A, C, A^2, C^2, AC, A^3, C^3, A^2C, AC^2.
r_3 : Constant, A, B, A^2, B^2, AB, A^3, B^3, A^2B, AB^2.
r_4 : Constant, B, B^2, B^3.
r_5 : Constant, C, C^2, C^3.

Table 2. The set of possible regressors for each regulator of Sirius.

are many regressors which are highly correlated with each other (for example the two regressors A^2 and A^3, whose correlation coefficient is equal to 0.98).

In order to overcome the problem, LGSS computes the *variance inflation factor* (VIF) for each regressor [1], which gives an idea of the degree of multicollinearity introduced by a regressor, when some other regressors are already in the regression equation. In LGSS the user can select a threshold value for the variance inflation factor, in order to avoid the insertion of collinear regressors. This solution, however, may affect the performance of the algorithm since the computing of VIF requires many additional computations [26].

Of course, a way to overcome the problem of multicollinearity is to drop collinear variables before launching the regression phase of LGSS. In the following we define an algorithm, based on a hierarchical clustering technique [11], which permits to cluster the time series of the regressors associated to the same reaction and to select those which are less correlated and that best satisfy the log-gain principle [16, 17], a principle developed in the MP theory based on a general criterion concerning the variations of quantities involved in biological phenomena [2]. The algorithm is based on the following procedure (we refer to [25] for details concerning the calculation of the log-gain score used in the algorithm).
For each reaction r in the MP system:

1. start by associating the time series of regressors of reaction r to different clusters.
2. Compute distances (similarities) between clusters. We consider the distance between one cluster and another cluster to be equal to the average distance from any time series of one cluster to any time series of the other cluster. The distance $d(g_1, g_2)$ of two regressors g_1, g_2 is given by the following two equations, where $corr(G_1^t, G_2^t)$ denotes the value of the Pearson's correlation coefficient [37] between the time series obtained by evaluating the two regressors on the t observed time points:

   $$d(g_1, g_2) = 1 - |corr(G_1^t, G_2^t)|$$

   if we do not want to distinguish between positive and negative correlations, and

   $$d(g_1, g_2) = 1 - corr(G_1^t, G_2^t)$$

   when we need to consider this.
3. Find the closest (most similar) pair of clusters and merge them into a single cluster (so that now we have one cluster less).
4. Repeat steps 2 and 3 until all the distances between clusters are greater than a user defined threshold value.
5. For each cluster computed, calculate the log-gain score of each regressor included (as defined in [25]) and select the one which have higher log-gain score. This permits to discard those regressors whose time series express changes which are not realistic in biology. The set of regressors for r which will be considered during the regression phase of LGSS are those collected at this step.

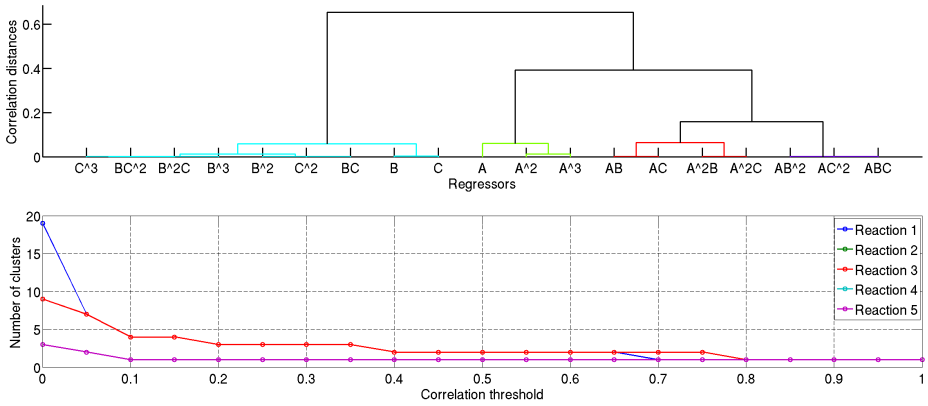


Fig. 6. On the top: the dendrogram which represents the clusters computed for the regressors of the rule r_1 of Sirius, by considering a threshold value of 0.1. On the bottom: the chart which displays the number of the computed clusters, for each reaction of Sirius, with respect to different threshold values of the maximum correlation distance between clusters.

r_1 : Constant, A, B, AB, AB^2.
r_2 : Constant, A, C, AC, AC^2.
r_3 : Constant, A, B, AB, AB^2.
r_4 : Constant, B.
r_5 : Constant, C.

Table 3. The set of possible regressors for each regulator of Sirius after the execution of the clustering algorithm. The total number of regressors is decreased of the 60% with respect to the set considered in Table 2.

The algorithm described above was included in LGSS and permits to solve the problem of multicollinearity without affecting the performance of the regression. In fact, LGSS launches this algorithm before starting the regression phase. The application of the algorithm, to the set of 48 regressors of Table 2, permits to reduce of more than the 60% the total set of regressors, by considering a threshold value of 0.1 (see Figure 6). The new set of regressors is given in Table 3.

The MP grammar computed by LGSS, starting from the set of regressors in Table 3, is given in Table 4 (see also Figure 7). This MP grammar is much better than the one given in Table 5, provided by LGSS with the initial set of possible regressors of Table 2. In fact, the new model uses less regressors (with lower degree) and permits a more clear comprehension of the regulative role of each substance of the system.

$r_1 : \emptyset \to A$	$\varphi_1 = 0.047 + 0.087A$
$r_2 : A \to B$	$\varphi_2 = 0.002A + 0.0002AC$
$r_3 : A \to C$	$\varphi_3 = 0.002A + 0.0002AB$
$r_4 : B \to \emptyset$	$\varphi_4 = 0.04B$
$r_5 : C \to \emptyset$	$\varphi_5 = 0.04C$

Table 4. The MP grammar of the Sirius oscillator, the dynamics is given in Figure 7.

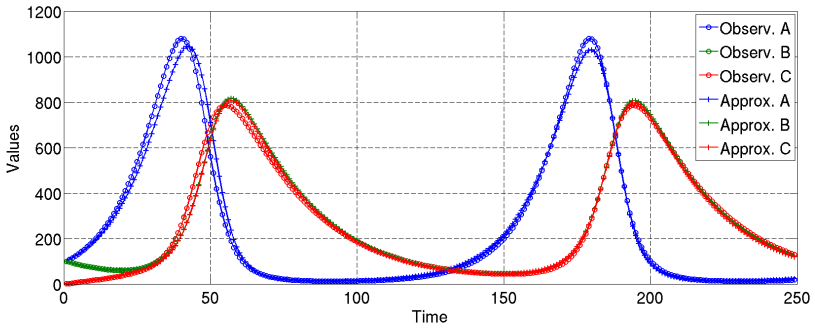


Fig. 7. Sirius' dynamics calculated by means of the MP grammar given in Table 4.

4 Conclusions

In this paper, a new algebraic formulation of inverse dynamical problems, based on MP grammars and Kronecker product, has been given, which provides, in general terms, the logic underlying the LGSS algorithm and proves its correctness in the approximate solution of inverse dynamical problems.

Even if computational tools are available for evaluating unknown parameters of ODE models [14, 8], LGSS seems to point out a more general methodology. In fact, LGSS not only discovers unknowns parameters, but suggests also the form of regulators as a combination of basic functions. This possibility could be very important in the case where the knowledge about the phenomenon under investigation is so poor that no clear idea is available about the kind of model underlying the observed behaviour.

The LGSS algorithm introduces a new perspective in the analysis of time series produced by the variables of a system evolving in time. This perspective can be defined as a generative one, where phenomena observed in time are reconstructed in terms of variable influences/transformations determined by the internal global states of the system. This approach is of course relevant in systems biology, but has a wide field of applications. In fact, in all the cases where some variables change according to some mutual relationship, due to mutual and systemic logic involving them, we are allowed to apply this general paradigm of discrete mathematical analysis.

$r_1 : \emptyset \to A \;\; \varphi_1 = 1.06 + 0.082A$
$r_2 : A \to B \; \varphi_2 = 3.6 \cdot 10^{-6} A^2 + 0.0002AC$
$r_3 : A \to C \; \varphi_3 = 0.004B + 8.9 \cdot 10^{-7} A^2 + 0.0001AB + 3.3 \cdot 10^{-8} A^2 B$
$r_4 : B \to \emptyset \;\; \varphi_4 = 0.04B$
$r_5 : C \to \emptyset \;\; \varphi_5 = 0.04C$

Table 5. The MP grammar of the Sirius oscillator computed by LGSS starting from the set of regressors given in Table 2. Due to the problem of the multicollinearity of regressors, the MP grammar is more complicated than the one given in Table 4.

References

[1] A. D. Aczel and J. Sounderpandian. *Complete Business Statistics*. Mc Graw Hill, International Edition, 2006.

[2] L.von Bertalanffy. *General Systems Theory: Foundations, Developments, Applications*. George Braziller Inc., New York, 1967.

[3] G. Ciobanu, Gh. Păun, and M.J. Pérez-Jiménez (Eds.), editors. *Applications of Membrane Computing*. Springer, 2006.

[4] D.W. Corne and P. Frisco. Dynamics of HIV infection studied with cellular automata and conformon-P systems. *Biosystems*, 91(3):531–544, 2008.

[5] N. Draper and H. Smith. *Applied Regression Analysis, 2nd Edition*. John Wiley & Sons, New York, 1981.

[6] M. Gheorghe, N. Krasnogor, and M. Camara. P systems applications to systems biology. *Biosystems*, 91(3):435–437, 2008.

[7] R.R. Hocking. The Analysis and Selection of Variables in Linear Regression. *Biometrics*, 32, 1976.

[8] S. Hoops, S. Sahle, R. Gauges, C. Lee, and J. Pahle. COPASI-a COmplex PAthway SImulator. *Bioinformatics*, 22(24), 2006.

[9] R.A. Horn and C.R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991.

[10] A.K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, 1989.

[11] S.C. Johnson. Hierarchical Clusterin Schemes. *Psychometrika*, 2:241–254, 1967.

[12] T. Karya and H. Kurata. *Generalized Least Squares*. Wiley, 2004.

[13] D.G. Luenberger. *Optimization by Vector Space Methods*. John Wiley & Sons Inc., 1969.

[14] T. Maiwald and J. Timmer. Dynamical modeling and multi-experiment fitting with PottersWheel. *Bioinformatics*, 24(18):2037–2043, 2008.

[15] V. Manca. String Rewriting and Metabolism: A logical perspective. In *Computing with Bio-Molecules*, pages 36–60. Springer-Verlag, 1998.

[16] V. Manca. The metabolic algorithm for P systems: Principles and applications. *Theoretical Computer Science*, 404:142–155, 2008.

[17] V. Manca. *Algorithmic Bioprocesses*, chapter 28: Log-Gain Principles for Metabolic P Systems, pages 585–605. Natural Computing. Springer-Verlag, 2009.
[18] V. Manca. From P to MP Systems. *WMC 2009, LNCS*, 5957:74–94, 2009.
[19] V. Manca. Fundamentals of Metabolic P Systems. In *[36]*, chapter 19, pages 475–498. Oxford University Press, 2010.
[20] V. Manca. Metabolic P Dynamics. In *[36]*, chapter 20, pages 499–528. Oxford University Press, 2010.
[21] V. Manca. Metabolic P systems. *Scholarpedia*, 5(3):9273, 2010.
[22] V. Manca and L. Bianco. Biological networks in metabolic P systems. *BioSystems*, 91(3):489–498, 2008.
[23] V. Manca and L. Marchetti. Metabolic approximation of real periodical functions. *The Journal of Logic and Algebraic Programming*, 79:363–373, 2010.
[24] V. Manca and L. Marchetti. Goldbeter's Mitotic Oscillator Entirely Modeled by MP Systems. *CMC 2010, LNCS 6501*, pages 273–284, 2010.
[25] V. Manca and L. Marchetti. Log-Gain Stoichiometic Stepwise regression for MP systems. *Int. Journal of Foundations of Computer Science*, 22(1):97–106, 2011.
[26] V. Manca and L. Marchetti. Solving Dynamical Inverse Problems by means of Metabolic P Systems. *BioSystems*, 2012. DOI:10.1016/j.biosystems.2011.12.006.
[27] V. Manca and L. Marchetti. Application of the MP theory to systems biology. In *Proceedings of the International Conference on Bio-inspired Systems and Signal Processing*, pages 303–308. SciTePress, 2012. DOI: 10.5220/0003852003030308.
[28] V. Manca and M.D. Martino. From String Rewriting to Logical Metabolic Systems. In *Grammatical Models of Multiagent Systems vol. 8*, pages 297–315. Gordon and Breach Science Publishers, 1999.
[29] V. Manca, L. Bianco, and F. Fontana. Evolutions and Oscillations of P systems: Theoretical Considerations and Application to biological phenomena. *WMC5 2004, LNCS*, 3365:63–84, 2005.
[30] V. Manca, G. Franco, and G. Scollo. State Transition Dynamics: basic concepts and molecular computing perspectives. In *Molecular Computational Models*, pages 32–55. IDEA Group INC., 2005.
[31] V. Manca, L. Marchetti, and R. Pagliarini. MP Modelling of Glucose-Insulin Interactions in the Intravenous Glucose Tolerance Test. *Int. Journal of Natural Computing Research*, 2(3):13–24, 2011.
[32] L. Marchetti and V. Manca. A methodology based on MP theory for gene expression analysis. *CMC 2011, LNCS 7184*, pages 300–313, 2012.
[33] Gh. Păun. Computing with membranes. *J. Comput. System Sci.*, 61(1): 108–143, 2000.
[34] Gh. Păun. *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
[35] Gh. Păun. A quick introduction to membrane computing. *Journal of Logic and Algebraic Programming*, 79(6):291–294, 2010.

[36] Gh. Păun, G. Rozenberg, and A. Salomaa, editors. *Handbook of Membrane Computing*. Oxford University Press, 2010.

[37] K. Pearson. Notes on the History of Correlation. *Biometrika*, 13(1):25–45, 1920.

[38] F.J. Romero-Campero and M.J. Pérez-Jiménez. Modelling gene expression control using P systems: The Lac Operon, a case study. *Biosystems*, 91(3): 438–457, 2008.

[39] A. Spicher, O. Michel, M. Cieslak, J.L. Giavitto, and P. Prusinkiewicz. Stochastic P systems and the simulation of biochemical processes with dynamic compartments. *Biosystems*, 91(3):458–472, 2008.

[40] W.H. Steeb. *Matrix Calculus and Kronecker Product with Applications and C++ Programs*. World Scientific Publishing, 1997.

[41] W.H. Steeb. *Problems and Solutions in Introductory and Advanced Matrix Calculus*. World Scientific Publishing, 2006.

[42] T. Strutz. *Data Fitting and Uncertainty. A practical introduction to weighted least squares and beyond*. Vieweg+Teubner, 2010.

[43] J. Wolberg. *Data Analysis Using the Method of Least Squares: Extracting the Most Information from Experiments*. Springer, 2005.

Parallel Simulation of Probabilistic P Systems on Multicore Platforms

Miguel A. Martínez-del-Amor[1], Ian Karlin[2], Rune E. Jensen[2], Mario J. Pérez-Jiménez[1], Anne C. Elster[2]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mdelamor@us.es, marper@us.es

[2] High Performance/Heterogeneous and Parallel Computing Lab
Department of Computer and Information Science
Norwegian University of Science and Technology
Sem Sælands vei 9, NO-7491, Trondheim, Norway
E-mail: Ian.Karlin@colorado.edu, runeerle@idi.ntnu.no, elster@idi.ntnu.no

Summary. Ecologists need to model ecosystems to predict how they will evolve over time. Since ecosystems are non-deterministic phenomena, they must express the likelihood of events occurring, and measure the uncertainty of their models' predictions. One method well suited to these demands is Population Dynamic P systems (PDP systems, in short), which is a formal framework based on multienvironment probabilistic P systems. In this paper, we show how to parallelize a Population Dynamics P system simulator, used to model biological systems, on multi-core processors, such as the Intel i5 Nehalem and i7 Sandy Bridge. A comparison of three different techniques, discuss their strengths and weaknesses, and evaluate their performance on two generations of Intel processors with large memory sub-system differences is presented. We show that P systems are memory bound computations and future performance optimization efforts should focus on memory traffic reductions. We achieve runtime gains of up to 2.5x by using all the cores of a single socket 4-core Intel i7 built on the Sandy Bridge architecture. From our analysis of these results we identify further ways to improve the runtime of our simulator.

Key words: Population Dynamics, P systems, Parallel Simulation, Multicore Computing, OpenMP

1 Introduction

Multienvironment probabilistic P systems are used to model species in real ecosystems, such as the bearded vulture in the Catalan Pyrenees [4] and zebra mussels in

the Ribarroja reservoir [3]. They conform a formal framework for ecological modelling called Population Dynamics P systems. These models are first validated by a software tool, and can reproduce actual measurements taken in a given number of years [4]. The goal of work is to be able to use P systems simulations to adopt *a priori* management strategies for the real system. However, P systems are computationally and data expensive with large systems, such as the one modelling the zebra mussel ecosystem, taking hours to run on a single set of input parameters on a single core processor.

Due to the probabilistic behaviour of these systems, ecological experts and model designers run many simulations on each set of input parameters to extract statistical information of the likelihood of certain behaviours occurring [4]. This makes the systems large. The more simulations run, the more confident they can be in the model's output. Also, the more input parameters they test, the greater certainty that the real-life experiments they run will yield useful knowledge. Therefore, the overall runtime of the simulations is critical.

This paper describes our initial parallelization work, which includes implementing a C/C++ version of the DCBA algorithm [7]. We have designed an implementation which saves on memory by avoiding the creation of a static table. We also choose C for its similarity to the common GPGPU (General Purpose computations on Graphics Processing Units) languages of OpenCL and CUDA, and the support of many parallel libraries. Our new implementation is parallelized in three ways: 1) simulations, 2) environments and 3) a hybrid approach. All them are implemented using the parallel standard library for multicore platforms, OpenMP [1]. From our analysis of these results we identify further ways to improve the runtime of our simulator, by minimizing memory and cache bottlenecks using data compression and GPU computing. Ideas and references on compression and GPU computing can be found in [2].

The rest of the paper is organized as follows: Section 2 gives an overview of the P systems framework that includes our simulator. Section 3 discusses the three forms of parallelism we introduced into the simulator and the advantages and disadvantages of each. Section 4 contains experimental results from testing out initial parallelization efforts. Finally, conclusions and future work are presented in Section 5.

2 System overview

The simulator we are optimizing and parallelizing is included in a P systems based modelling framework that contains four design levels. An overview of the framework is shown in Figure 1 along with the domain specific knowledge needed to implement each level. At the top level, ecological experts express living systems as inputs based upon observed data and/or conditions they wish to test. Example inputs include the number of organisms living in the ecosystem, temperature data and the probabilities of events happening. These inputs are then fed into proba-

bilistic P systems with the assistance of the model designers, which provide the rules used by the model.

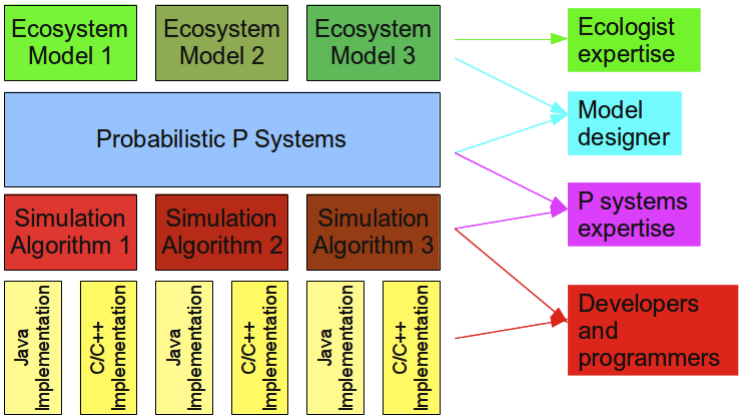


Fig. 1. Framework levels and expertise needed

The probabilistic P system software level currently uses a framework called PLinguaCore [6]. PLinguaCore is written in Java, and contains implementations of different types of P systems, along with a standard specification language for P systems known as P-Lingua. Both ecologists and model designers run their simulations using this framework through a software tool called MeCoSim [9] that provides an abstraction layer for non-experts in P system models.

The execution of the rules input by the model designer occurs in the simulation algorithms, which are implemented in common programming languages. Fast execution of the algorithms is important to ecological experts and model designers. Also, ecological experts often simulate many hypotheses before deciding which hypotheses to experimentally evaluate. The simulation algorithm and implementation levels are the most time consuming parts of the simulations, and thus, where we focus on improving performance.

Algorithm 1 Main loop of the simulator

```
1: for sim ← 0, ..., simulations do
2:    INITIALIZATION
3:    for step ← 0, ..., time_steps do
4:       for env ← 0, ..., environments do
5:          SELECTION OF RULES
6:       end for
7:       for env ← 0, ..., environments do
8:          EXECUTION OF RULES
9:       end for
10:   end for
11: end for
```

The last simulation algorithm for PDP systems is the DCBA algorithm [7]. A high level algorithm representation of our simulator is shown in Algorithm 1. The outermost loop runs the simulation multiple times as specified by the user. The next loop performs the discrete time steps that advance the simulation. The time steps are performed in two stages: selection and execution. During the first stage (selection) all environments, which each represent discrete parts of an ecosystem with different properties, have rules selected to be executed on them. All environments have the same rules, but the associated probability indicating the likelihood of them being executed varies between environments. In the second stage (execution) of a time step, the rules are executed. During a time step, environments evolve independently, but between time steps can communicate objects.

Rules are classified into rule blocks by their left-hand sides (consuming the multisets of objects in the same compartments, and according to the same charge of the active membrane) and the charge of the active membrane in the right-hand side (consistent blocks [7]). The probabilities of the rules within a block sum 1 (they have local meaning inside the blocks).

Selection stage is split into three micro-phases (see [7] for more details): phase 1 (object distribution), phase 2 (maximality) and phase 3 (probabilities). The DCBA algorithm uses a table for phase 1 in order to distributes the objects along the rule blocks. This table has one column per each rule block, and one row per each pair object and membrane (also considering the environment itself). Therefore, the size of the table is of order $O(|B| \cdot |\Gamma| \cdot (q+1))$, being $|B|$ the number of rule blocks, $|\Gamma|$ the size of the alphabet (total amount of different objects), and $q+1$ the number of membranes in the system plus the space for the environment in each one.

The implementation of this table can be inefficient in systems with a large number of rule blocks and/or objects. Therefore, our simulator does not really implement the table. The main baseline idea is to translate operations over the table to operations directly to the rule blocks:

- Operations over columns: they can be transformed to operations over the rule blocks and their left-hand sides (LHS in short).
- Operations over rows: they can be transformed to operations over the left-hand sides of rule blocks and storing the partial result in a global variable for each row.

Phase 1 can be implemented as shown in Algorithm 2. Phase 2, phase 3 and execution stage can be directly implemented following the corresponding definitions in [7].

Remark that in this implementation, instead of using a real table, we *virtually* implement it by using operations over the information of the rule blocks. Actually, two extra vectors are only used:

- *Activation vector*: We annotate the blocks that has not been filtered by a boolean value.
- *Addition vector*: We add the values of the rows by using this global vector, one per each pair object and membrane.

Algorithm 2 Selection Phase 1 (Distribution)

1: Apply *Filter 1*: for each block, if the charge in the LHS is different to the one presented in the configuration, then deactivate the block: $activationVector[b] = false$.
2: Apply *Filter 2*: for each block, if one of the objects involved in the left-hand side does not exist in C_t, then deactivate the block: $activationVector[b] = false$.
3: Check the mutually consistency of blocks.
4: **repeat**
5: For each active block, and for each object in the left-hand side, add the multiplicity k appearing in the block to a global variable for the corresponding object ($addition[object, membrane] += k$).
6: For each active block, calculate the minimum of the object distributions in the left-hand side: $N_b = Min_{[o^k]_m \in LHS(block)}(\frac{1}{k^2} * \frac{1}{addition[o,m]} * C[o,m])$. This is the number of applications for block b: $NumAppBlocks[b] += N_b$.
7: Delete objects in the configuration C, corresponding with N_b.
8: Apply *Filter 2*.
9: $a = a + 1$
10: **until** $a == A$ **or** *every* $N_b == 0$

3 Design and parallelism

Before we introduced parallelism, we first rewrote the simulator in C/C++ which is advantageous because OpenMP, PThreads and MPI all are supported. In this section, we describe the implementation of the three forms of parallelism added to our simulator. A discussion of the advantages and disadvantages of each is included.

Simulations are parallelized by using the *#pragma omp parallel for* OpenMP directive on the simulation loop from Algorithm 1. The advantage of running simulations in parallel is there are no data dependencies between simulations, and, therefore, the problem is embarrassingly parallel. Also, the users of our simulator typically run 50 to 100 simulations of each set of input parameters, so there are enough simulations to consume all cores. However, there are disadvantages of running simulations in parallel. Each simulation needs its own memory space increasing the amount of memory used. If the number of simulations is not divisible by the number of processors then load balancing issues can occur with the final simulations running while some cores are idle. Also, running simulations in parallel can result in resource conflicts as cores compete for shared resources.

Environments are parallelized by using the *#pragma omp parallel* to generate a thread pool for the simulation. Then the for loops in Algorithm 1 that iterate over environments are parallelized with *#pragma omp for*, which has an implicit barrier that enforces the dependencies between the stages in each time step. Using this design, creating new thread blocks for each for loop is avoided.

The advantage of parallelizing environments over simulations is that memory usage does not increase. However, dependencies occur twice in each time step requiring synchronization steps. Also, since most models use 5 to 30 environments, there are cases where modern machines have more cores than environments and

just parallelizing environments cannot take advantage of all computing resources. In addition, as with simulations, load balancing can be an issue if the number of environments is not divisible by the number of cores, or if the runtime of environments varies.

Hybrid parallelization is accomplished by combining parallel environments with parallel simulations. We accomplish hybrid parallelization through command-line flags that allow the specification of how many environments or simulations to run in parallel. By combining both forms of parallelism, we can balance the amount of each resource used. This will become more important as the number of cores within a node increases. For example, the number of simulations can be increased until available memory is used and then environments within each system can be parallelized.

4 Experimental Evaluation

In this section, we describe a series of tests performed on our implementation and the systems they were run on, along with the results from those experiments.

4.1 Test Environment and Methodology

The following experiments were run on the two machines shown in Table 1. The tests used random systems with similar amounts of data to real-life examples. Multiple configurations with environments and simulations varying from 10 to 50 were tested for the parallel environments, simulations and two hybrid combinations. Each of these tests were run on 1 to 8 cores for the Intel i5 machine, and 1 to 4 for the Intel i7. The measurements in this section, except when noted, correspond only to the parallelized part of the code.

Processor	**Speed**	**Bus speed**	**Cache**
i5 Nehalem (2x4)	2 Ghz	3x800 Mhz	2x4 MB
i7 Sandy Bridge (1x4)	3.4 Ghz	2x1333 Mhz	8 MB

Table 1. Specifications of the test machines.

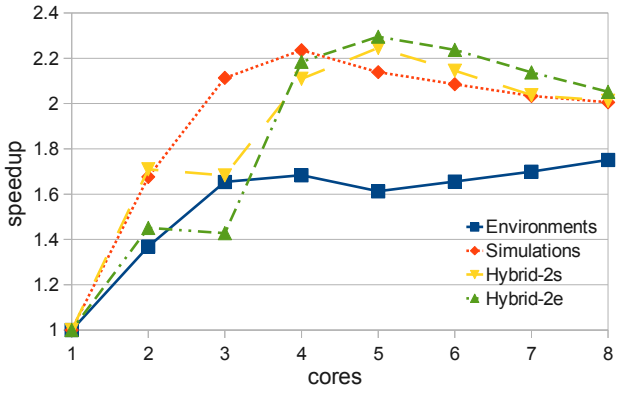
4.2 Results

The serial runtime on both of our test machines is shown in Table 2. *Setup* is the cost of running the serial portion of the code. The other two columns represent the runtime extremes of our test cases when run in serial. From the table, we can see that in serial the setup portion is a small part of the overall runtime, and that the Sandy Bridge processor is about 2.5 times faster than the Nehalem.

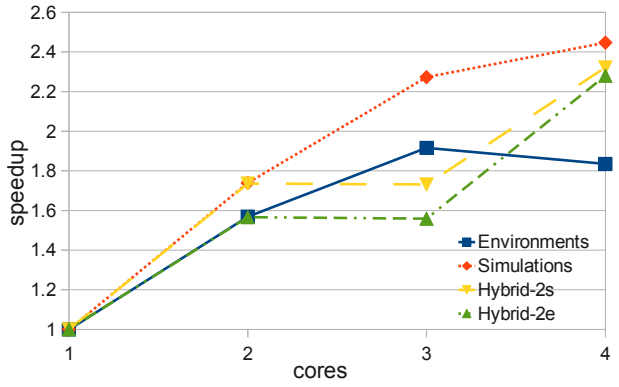
Processor	Setup	10 env & sim	50 env & sim
Nehalem	0.8s	48.0s	251.0s
Sandy Bridge	0.35s	19.9s	97.8s

Table 2. Serial Runtimes

Figures 2 and 3 show the performance improvements of parallelizing our system in various ways. The two figures are representative of the other tests we performed with the best performance either being parallelizing by simulations or the hybrid method (2s), which uses two simulations and then parallelizes by environments. Another trend shown is that as the number of simulations increases, the advantage of parallelizing by simulations increases. The same effect is observed for environments.



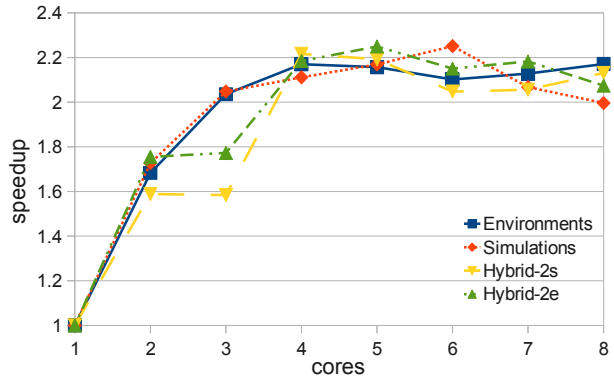
(a) Nehalem



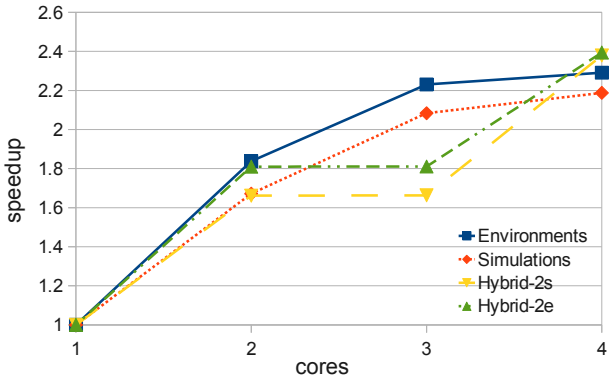
(b) Sandy Bridge

Fig. 2. Speedups running 50 simulations with 10 environments in the system

On the Sandy Bridge system the largest speedup of 2.5x occurs for 50 simulations and 50 environments. However, the maximum speedup on Sandy Bridge



(a) Nehalem



(b) Sandy Bridge

Fig. 3. Speedups running 10 simulations with 50 environments in the system

when going from 3 to 4 processors is only between 0.1 and 0.2, suggesting that the calculation is memory bound for larger core counts. On the Nehalem machine, the maximum parallel speedup was 2.3x for all tests, which is barely greater than the added available bandwidth from using the second socket. These results, led us to suspect we that the Nehalem system's performance was being limited our programming approach. In particular, we did not account for the Non Uniform Memory Access (NUMA) memory subsystem of the two sockets.

One final test was run to see if NUMA was hurting the performance of our code on the Nehalem machine. First one and then two instances of the code was run with 4 threads each (affinity locked to different sockets) on the machine. With two instances a 2x speedup was achieved over the best parallel results from running one instance of the OpenMP version. Confirming this result is that locking all 4 threads to a single socket performance results in a 50% performance increase when compared to locking 2 threads to each socket. For the current tests, however, affinity is controlled by the operating system and performance is similar to when

two threads were locked to each socket. These preliminary tests also indicates that the code is memory bound since overall speedups on 8 cores were less than 5x.

5 Conclusions and future work

In this paper, we showed how P systems simulations can take advantage of modern multi-core architectures. Our implementation included three forms of parallelism. Experiments ran to test the simulator indicate the simulations are memory bound and the portion of the code we parallelized consumes over 98% of the runtime in serial. From this initial work we conclude that parallelizing by simulations or hybrid techniques yields the largest speedups. Also, using hardware, such as Intel's Sandy Bridge, that has more memory bandwidth is an easy way for scientists to improve the speed of our simulator. It can also be concluded that performance tuning to decrease data movement is important for P-system simulators.

Future Work

This paper leaves open many research questions that we plan to explore by building on this work. We have experience overlapping communication and computation [8] and compressing data [2] both of which will be especially important on GPUs. We anticipate large speedups from using GPUs due to their increased memory bandwidth and computational capabilities. In addition, we plan to leverage our experience tuning shared memory systems to other optimizations for NUMA processors [5]. Open research questions on NUMA machines include how our hybrid parallel approach will best map to various processor configurations. A hybrid GPU and CPU code will be able to take advantage of all compute resources on a given system. Either an OpenCL or CUDA/C hybrid implementation will be used. While not a high priority because the simulator is usually run on scientists workstations an MPI version would offer large speedups due to simulations being embarrassingly parallel.

As core counts continue to increase, exploiting parallelism within the environments may be profitable. Also, of interest is eliminating or reducing the synchronization required at each time step. While dependencies exist between environments, not all environments depend on all other environments. For an environment to begin executing its next step, all environments from which organisms can migrate into it must be finished. We believe that for simulations with few migration paths between environments, or with a large number of environments, it is important balance workloads better.

Acknowledgments

M.A. Martínez-del-Amor and M.J. Pérez-Jiménez acknowledge the support of "Proyecto de Excelencia con Investigador de Reconocida Valía" of the "Junta de Andalucía" under grant P08-TIC04200, and the support of the project TIN2009-13192 of the "Ministerio de Educación y Ciencia" of Spain, both co-financed by FEDER funds.

References

1. The OpenMP specification. "`http://www.openmp.org`".
2. A. A. Aqrawi and A. C. Elster. Bandwidth reduction through multithreaded compression of seismic images. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, pages 1730 –1739, may 2011.
3. M. Cardona, M. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M. Pérez-Jiménez, and D. Sanuy. A computational modeling for real ecosystems based on p systems. *Natural Computing*, 10:39–53, 2011.
4. M. A. Colomer, A. Margalida, D. Sanuy, and M. J. Pérez-Jiménez. A bio-inspired computing model as a new tool for modeling ecosystems: The avian scavengers as a case study. *Ecological Modelling*, 222(1):33 – 47, 2011.
5. A. C. Elster and J. C. Meyer. A super-efficient adaptable bit-reversal algorithm for multithreaded architectures. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–8, Washington, DC, USA, 2009.
6. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and A. Riscos-Núñez. An overview of p-lingua 2.0. *Lecture Notes in Computer Science*, 5957:264–288, 2010.
7. M. Martínez-del Amor, I. Pérez-Hurtado, M. García-Quismondo, L. Macías-Ramos, L. Valencia-Cabrera, A. Romero-Jiménez, C. Graciani-Díaz, A. Riscos-Núñez, M. Colomer, and M. Pérez-Jiménez. Dcba: Simulating population dynamics P systems with proportional object distribution. In *this volume*.
8. T. Natvig and A. C. Elster. Run-time analysis and instrumentation for communication overlap potential. In *Proceedings of the 17th European MPI users' group meeting conference on Recent advances in the message passing interface*, EuroMPI'10, pages 42–49, Berlin, Heidelberg, 2010. Springer-Verlag.
9. I. Pérez-Hurtado, L. Valencia-Cabrera, M. J. Pérez-Jiménez, M. A. Colomer, and 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-inpired Computing: Theories and Applications (BIC-TA 2010)*, volume I, pages 637–643, Changsha, China, 2010. IEEE, Inc.

DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution

M.A. Martínez-del-Amor[1], I. Pérez-Hurtado[1], M. García-Quismondo[1], L.F. Macías-Ramos[1], L. Valencia-Cabrera[1], A. Romero-Jiménez[1], C. Graciani-Díaz[1], A. Riscos-Núñez[1], M.A. Colomer[2], M.J. Pérez-Jiménez[1]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mdelamor@us.es, perezh@us.es, mgarciaquismondo@us.es, lfmaciasr@us.es, lvalencia@us.es, romero.alvaro@us.es, cgdiaz@us.es, ariscosn@us.es, marper@us.es

[2] Department of Mathematics
University of Lleida
Avda. Alcalde Rovira Roure, 191, 25198 Lleida, Spain
E-mail: colomer@matematica.udl.es

Summary. Population Dynamics P systems refer to a formal framework for ecological modelling. The semantics of the model associates probabilities to rules, but at the same time, the model is based on P systems, so the rules are applied in a maximally parallel way. Since the success of the first model using this framework [5], initially called multienvironment probabilistic P systems, several simulation algorithms have been defined in order to better reproduce the behaviour of the ecosystems with the models.

BBB and DNDP are previous attempts, which define blocks of rules having the same left-hand side, but do not define a deterministic behaviour when different rules are competing for the same resources. That is, different blocks of rules present in their left-hand side common objects, being applicable at the same time. In this paper, we introduce a new simulation algorithm, called DCBA, which performs a proportional distribution of resources.

Keywords: Membrane Computing, Population Dynamics, Simulation Algorithm, Probabilistic P systems, DCBA, P-Lingua, pLinguaCore

1 Introduction

Membrane Computing has a far–reaching background on the modelling of biochemical phenomena, within the framework of Computational Systems Biology

[2, 7, 17, 19], being complementary and an alternative to more classical approaches (i.e. ODEs, Petri Nets, etc). However, in 2011 a Membrane Computing modelling framework for ecosystem dynamics was introduced [3]. Based on this framework, several ecosystem models have already been presented. Some examples are the population dynamics of *Gypaetus barbatus* [4] and *Rupicapra p. pyrenaica* [8] in the Catalan Pyrenees, as well as the population density of *Dreissena polymorpha* in Ribarroja reservoir [3]. Some of the assets of this framework are the ability to analyse the simultaneous evolution of a high number of species, as well as the management of a large number of auxiliary objects. These objects could represent, for instance, grass, biomass or animal bones.

The results obtained from the application of the framework on different ecosystems prove its versatility and adaptability. Thus, a straightforward interpretation of the results of the simulations of its models can be easily obtained by checking the states and multisets associated to each one of the membranes.

Although this framework allows a direct interpretation of the simulations of its models, the simulation itself is a complicated problem to solve from a practical point of view. Therefore, algorithms capable of capturing the semantics described by the framework are necessary. These algorithms should be able to select rules in the models according to their associated probabilities while keeping the maximal semantics of P systems. In this scenario, the concept of *rule block* takes form. A rule block is a set of rules whose left hand side (that is, the necessary and sufficient condition for them to be applied) is exactly the same. That is, given a P system configuration, either all or none of the rules in the block can be applied. According to the semantics associated to the modelling framework, one or more blocks are selected on each step of computation. The probability for a block to be selected is calculated out of the probabilities of its rules. Once a rule block is selected, its rules are applied a number of times in a probabilistic manner according to their associated probabilities, also known as *local probabilities*. Henceforth, the condition of the sum of the probabilities associated to all rules in each block being equal to 1 is imposed.

The way in which the blocks and rules in the model are selected depends on the specific simulation algorithm employed. These algorithms should be able to deal with issues such as the possible overlapping of left hand sides from different blocks, which might result on the competition of blocks and rules for objects. So far, several algorithms have been developed in order to capture the semantics defined by the modelling framework. Some of these algorithms are the Binomial Block Based algorithm (*BBB*) and the Direct Non Deterministic algorithm with Probabilities (*DNDP*). A comparison on the performance of these algorithms can be found on [9].

The algorithms mentioned above share a common drawback. This drawback involves the distortion of the way in which blocks and rules are selected. That is, instead of blocks and rules being selected according to its probabilities in a uniform manner, this selection process is biased towards those with the highest probabilities. This paper introduces a new algorithm, known as Direct distribution based on Consistent Blocks Algorithm (*DCBA*). This algorithm is introduced to solve the aforementioned distortion, thus not biasing the selection process towards the most likely blocks and rules.

The rest of the paper is structured as follows: Section 2 introduces preliminary concepts, such as the formal modelling framework of PDP systems and the DNDP algorithm. Section 3 describes the DCBA algorithm, together with a test example to show the differences with DNDP, and some details on the implementation in the PlinguaCore software framework. Section 4 shows the behaviour of DCBA when simulating a real ecosystem model. The simulated model has been adapted and improved from the original version. The paper ends with some conclusions and ideas for future work in Section 5.

2 Preliminaries

2.1 The P system based framework

Definition 1. *A Population Dynamics P system of degree* (q, m) *with* $q \geq 1$, $m \geq 1$, *taking* T *time units,* $T \geq 1$, *is a tuple*

$(G, \Gamma, \Sigma, T, R_E, \mu, R, \{f_{r,j} : \ r \in R, 1 \leq j \leq m\}, \{\mathcal{M}_{ij} : \ 0 \leq i \leq q-1, \ 1 \leq j \leq m\})$

where:

- $G = (V, S)$ *is a directed graph. Let* $V = \{e_1, \ldots, e_m\}$ *whose elements are called environments;*
- Γ *is the working alphabet and* $\Sigma \subsetneqq \Gamma$ *is an alphabet representing the objects that can be present in the environments;*
- T *is a natural number that represents the simulation time of the system;*
- R_E *is a finite set of communication rules between environments of the form*

$$(x)_{e_j} \xrightarrow{p_{(x,j,j_1,\ldots,j_h)}} (y_1)_{e_{j_1}} \ldots (y_h)_{e_{j_h}}$$

  where $x, y_1, \ldots, y_h \in \Sigma$, $(e_j, e_{j_l}) \in S$ $(l = 1, \ldots, h)$ *and* $p_{(x,j,j_1,\ldots,j_h)}(t) \in [0, 1]$, *for each* $t = 1, \ldots, T$. *If* $p_{(x,j,j_1,\ldots,j_h)}(t) = 1$, *for each* t, *then we omit the probabilistic function. These rules verify the following:*

  - ⋆ *For each environment* e_j *and for each object* x, *the sum of functions associated with the rules from* R_E *whose left-hand side is* $(x)_{e_j}$ *coincides with the constant function equal to 1.*

- μ *is a membrane structure consisting of* q *membranes, with the membranes injectively labeled by* $0, \ldots, q-1$. *The skin membrane is labeled by 0. We also associate electrical charges from the set* $\{0, +, -\}$ *with membranes.*

- *R is a finite set of evolution rules of the form $r : u[\,v\,]_i^\alpha \to u'[\,v'\,]_i^{\alpha'}$ where u, v, u', v' are multisets over Γ, $i \in \{0, 1, \ldots, q-1\}$, and $\alpha, \alpha' \in \{0, +, -\}$.*
- *For each $r \in R$ and for each j, $1 \leq j \leq m$, $f_{r,j}$ is a computable function whose domain is $\{1, \ldots, T\}$ and its range is $[0, 1]$, verifying the following:*
  - ⋆ *For each $u, v \in \Gamma^*$, $i \in \{0, \ldots, q-1\}$ and $\alpha, \alpha' \in \{0, +, -\}$, if $r_1, \ldots, r_z$ are the rules from R whose left-hand side is $u[v]_i^\alpha$ and the right-hand side have polarization α', then $\sum_{j=1}^{z} f_{r_j}(t) = 1$, for each t, $1 \leq t \leq T$.*
  - ⋆ *If $(x)_{e_j}$ is the left-hand side of a rule $r \in R_E$, then none of the rules of R has a left-hand side of the form $u[v]_0^\alpha$, for any $u, v \in \Gamma^*$ and $\alpha \in \{0, +, -\}$, having $x \in u$.*
- *For each j $(1 \leq j \leq m)$, $\mathcal{M}_{0j}, \ldots, \mathcal{M}_{q-1,j}$ are strings over Γ, describing the multisets of objects initially placed in the q regions of μ, within the environment e_j.*

In other words, a system as described in the previous definition can be viewed as a set of m environments $e_1, \ldots, e_m$ linked between them by the arcs from the directed graph G. Each environment e_j contains a P system, $\Pi_j = (\Gamma, \mu, R, \mathcal{M}_{0j}, \ldots \mathcal{M}_{q-1,j})$, of degree q, such that $\mathcal{M}_{0j}, \ldots, \mathcal{M}_{q-1,j}$ describe the initial multisets for this environment, and every rule $r \in R$ has a computable function $f_{r,j}$ (specific for environment j) associated with it.

The tuple of multisets of objects present at any moment in the m environments and at each of the regions of each Π_j, together with the polarizations of the membranes in each P system, constitutes a *configuration* of the system at that moment. At the initial configuration of the system we assume that all environments are empty and all membranes have a neutral polarization.

We assume that a global clock exists, marking the time for the whole system, that is, all membranes and the application of all rules (both from R_E and R) are synchronized in all environments.

The P system can pass from one configuration to another by using the rules from $R = R_E \cup \bigcup_{j=1}^{m} R_{\Pi_j}$ as follows: at each transition step, the rules to be applied are selected according to the probabilities assigned to them, and all applicable rules are simultaneously applied in a maximal way.

When a communication rule between environments

$$(x)_{e_j} \xrightarrow{p_{(x,j,j_1,\ldots,j_h)}} (y_1)_{e_{j_1}} \ldots (y_h)_{e_{j_h}}$$

is applied, object x passes from e_j to $e_{j_1}, \ldots, e_{j_h}$ possibly modified into objects $y_1, \ldots, y_h$, respectively. At any moment t, $1 \leq t \leq T$, for each object x in environment e_j, if there exist communication rules whose left-hand side is $(x)_{e_j}$, then one of these rules will be applied. If more than one communication rule can be applied to an object, the system selects one randomly, according to their probability which is given by $p_{(x,j,j_1,\ldots,j_h)}(t)$.

For each j $(1 \leq j \leq m)$ there is just one further restriction, concerning the consistency of charges: in order to apply several rules of R_{Π_j} simultaneously to the same membrane, all the rules must have the same electrical charge on their right-hand side.

2.2 DNDP simulation algorithm

In this section, the Direct Non-deterministic Distribution with probabilities algorithm (DNDP) [14, 13] is briefly described (algorithm 1). The aim of this algorithm is to perform a non-deterministic object distribution, so rules having common objects in their left-hand sides (object competition) will have the same opportunities to consume objects.

The input consists on a PDP system of degree (q, m), and a number T of time units. The algorithm simulates T transition steps of the PDP system. Therefore, it only simulates one computation of the PDP system, by selecting and executing rules in a non-deterministic maximal consistent parallel way.

Algorithm 1 DNDP MAIN PROCEDURE

Require: A PDP system of degree (q, m) with $q \geq 1$, $m \geq 1$, taking T time units, $T \geq 1$.

1: $C_0 \leftarrow$ initial configuration of the system
2: **for** $t = 0$ to $T - 1$ **do**
3: $C'_t \leftarrow C_t$
4: *Initialization*
5: *First selection phase (consistency).*
6: *Second selection phase (maximality).*
7: *Execution of selected rules.*
8: $C_{t+1} \leftarrow C'_t$
9: **end for**

Similarly to the previous algorithms [14], the transitions of the P system are simulated in two phases, selection and execution, to synchronize the consumption and production of objects. However, selection is divided in two micro-phases: the first one calculates a multiset of mutually consistent applicable rules, and the second assure maximal application by eventually increasing the multiplicity of some rules in the previous multiset, obtaining a multiset of maximal mutually consistent applicable rules. The algorithm is described below, but for more details refer to [13].

First of all, in order to simplify the selection and execution phases, the *initialization* process constructs two ordered set of rules, A_j and B_j, gathering only rules from R_E and R_Π applicable in environment e_j, in the sense of having the same charge in the left-hand side than the membranes in the configuration.

In the *first selection* phase, a multiset of consistent applicable rules, denoted by R_j^1 for each environment e_j, is calculated. Moreover, a multiset of possible applicable rules, denoted by R_j^0, is also created. We will say that two rules are consistent if they are associated to the same membrane, and they update it to the same charge. It is used in order to store rules having 0 as the number of applications when using the random number generator function. Hence, this multiset allows to have elements with multiplicity 0.

First, a random order is applied to $A_j \cup B_j$, and stored in an ordered set D_j. Moreover, a copy of the configuration C_t, called C'_t, is created and it is updated each time that a rule is selected (removing the left-hand side). Then, a rule r is applicable if the following holds: it is consistent with the previously (according to the order in D_j) selected rules in R_j^1, and the number of possible applications M in C'_t is greater than 0. If a rule r is applicable, a binomial distributed random number of applications n is calculated according to the probability.

On the one hand, since C'_t has been updated by the previously selected rules, the number n cannot exceed M to guarantee a correct object distribution. On the other hand, if the generated number n is 0, the corresponding rule is added to the multiset R_j^0, giving another chance to be selected in the next phase (maximality). Note that only rules from R_j^1 are considered for the consistency condition, since rules from R_j^0 are not applied in the first selection phase.

In the *second selection* phase, the consistent applicable rules are checked again in order to achieve maximality. Only consistent rules are considered, and they are taken from $R_j = R_j^0 \cup R_j^1$. If one rule $r \in R_j$ has a number of applications M greater than 0 in C'_t, then M will be added to the multiplicity of the rule. In order to fairly distribute the objects among the rules, they are iterated in order with respect to the probabilities. Moreover, one rule from the multiset R_j^0 can be checked, so it is possible that another rule from R_j^1, inconsistent to this one, have been previously selected. In this case, the consistent condition has to be tested again.

An example of several executions of the DNDP algorithm is showed in section 3.3, together with a comparison with the new algorithm introduced in this paper.

3 Direct distribution based on Consistent Blocks Algorithm (DCBA)

3.1 Definitions for blocks and consistency

The selection mechanism starts from the assumption that rules in R can be classified into blocks of rules having the same left-hand side, following the definitions 2, 3 and 4 below.

Definition 2. *The left and right-hand sides of the rules are defined as follows:*

(a) Given a rule $r \in R_\Pi$ of the form $r : u[v]_h^\alpha \to u'[v']_h^{\alpha'}$:

- *The left-hand side of r is defined as $LHS(r) = (h, \alpha, u, v)$, where $h \in L$, $\alpha \in \{0, +, -\}$ and $u', v' \in \Gamma^*$. This corresponds to multiset u in the parent membrane of h, multiset v in membrane h, and membrane h with charge α.*
- *The right-hand side of r is defined as $RHS(r) = (h, \alpha', u', v')$, where $h \in L$, $\alpha' \in \{0, +, -\}$ and $u', v' \in \Gamma^*$. This corresponds to multiset u' in the parent membrane of h, multiset v' in membrane h, and membrane h with charge α'.*

(b) Given a rule $r \in R_E$ of the form $r : (x)_{e_j} \rightarrow (y_1)_{e_{j_1}} \dots (y_k)_{e_{j_k}}$:

- *The left-hand side of r is defined as $LHS(r) = (e_j, x)$, corresponding to the multiset with only one occurrence of object x in environment e_j.*
- *The right-hand side of r is defined as $RHS(r) = (e_{j_1}, y_1) \dots (e_{j_k}, y_k)$, corresponding to the k multisets with single objects $y_1 \dots y_k$, for each environment $e_{j_1} \dots e_{j_k}$ respectively.*

Definition 3. *Rules from R_Π can be classified in blocks associated to (h, α, u, v) as follows: $B_{h,\alpha,u,v} = \{r \in R_\Pi : LHS(r) = (h, \alpha, u, v)\}$.*

Definition 4. *Rules from R_E can be classified in blocks associated to (e_j, a) as follows: $B_{e_j,a} = \{r \in R_E : \exists a \in \Sigma, LHS(r) \equiv (a)_{e_j}\}$.*

Recall that, according to the semantics of the model, the sum of probabilities of all the rules belonging to the same block is always equal to 1 – in particular, rules with probability equal to 1 form individual blocks. Note that rules with overlapping (but different) left-hand sides are classified into different blocks.

Definition 5. *A block $B_{h,\alpha,u,v}$ is consistent if and only if $\exists \alpha', \forall r \in B_{h,\alpha,u,v}$, $charge(RHS(r)) = \alpha'$.*

Definition 6. *A consistent block $B_{h,\alpha,\alpha',u,v}$, with $h \in H$, $\alpha, \alpha' \in \{0, +, -\}$, $u, v \in \Gamma^*$, is of the form $B_{h,\alpha,\alpha',u,v} = \{r \in R : \exists u', v' \in \Gamma^* : r \equiv u[v]_h^\alpha \rightarrow u'[v']_h^{\alpha'}\}$.*

Remark 1. Note that **all** the rules $r \in B_{h,\alpha,\alpha',u,v}$ are consistent, in the sense that each membrane h with charge α goes to the **same** charge α' when any rule of $B_{h,\alpha,\alpha',u,v}$ is applied.

Definition 7. *Two blocks $B_{h_1,\alpha_1,\beta_1,u_1,v_1}$ and $B_{h_2,\alpha_2,\beta_2,u_2,v_2}$ are mutually consistent with themselves, if and only if $(h_1 = h_2 \wedge \alpha_1 = \alpha_2) \Rightarrow (\beta_1 = \beta_2)$.*

Definition 8. *A set of blocks $\mathcal{B} = \{B^1, B^2, \dots, B^s\}$ is self consistent (or mutually consistent) if and only if $\forall i, j (i \neq j \Rightarrow B^i$ and B^j are mutually consistent).*

Remark 2. In such a context, a set of blocks has an associated set of tuples (h, α, α'), that is, a relationship of $H \times C$ in C. Then, a set of blocks is mutually consistent if and only if the associated relationship $H \times C$ in C is functional.

3.2 DCBA pseudocode

This new simulation algorithm for PDP systems has the same general scheme than its predecessor, DNDP (algorithm 1). The main loop (algorithm 2) is divided into two stages: selection and execution of rules, similarly to the DNDP algorithm.

Algorithm 2 DCBA MAIN PROCEDURE

Require: A Population Dynamics P system of degree (q, m), $T \geq 1$ (time units), and $A \geq 1$ (*Accuracy*). The initial configuration is then called C_0.

1: *INITIALIZATION* ▷ (Algorithm 4).
2: **for** $t \leftarrow 0$ to $T-1$ **do**
3: $C'_t \leftarrow C_t$
4: Calculate probability functions $f_{r,j}(t)$ associated to the rules.
5: *SELECTION* of rules. ▷ (Algorithm 3)
6: *EXECUTION* of rules. ▷ (Algorithm 8)
7: $C_{t+1} \leftarrow C'_t$
8: **end for**

Note that the algorithm selects and executes rules, but not blocks of rules. Blocks are used by DCBA in order to select rules, and this is made in three micro-stages as seen in algorithm 3. Phase 1 calculates a proportional object *distribution* to the blocks. Phase 2 assures the *maximality* by checking the maximal applications of each block. And finally, phase 3 passes from block applications to rule applications by calculating random numbers following the multinomial distribution with the corresponding *probabilities*.

Algorithm 3 SELECTION

1: Selection *PHASE 1*: distribution ▷ (Algorithm 5)
2: Selection *PHASE 2*: maximality ▷ (Algorithm 6)
3: Selection *PHASE 3*: probabilities ▷ (Algorithm 7)

Before starting to select and execute rules in the system, some data initialization is required (see algorithm 4). For instance, the selection stage uses a table in order to distribute the objects among the blocks. This table T, also called *static table*, is used in each time step, so it is initialized only once, at the beginning of the algorithm. The *static table* has one column per each consistent block of rules, and one row per each pair of object and compartment (i.e. each membrane and the environment in the skeleton). An expanded static table T_j is also constructed for each environment, to consider also blocks from environment communication rules. Finally, two multisets, B_j and R^j_{sel}, are initialized for selected blocks and rules, respectively.

Remark 3. The columns of the static table contain the information of their left-hand side of the blocks. The rows of the static table contain the information of the competitions for objects: each block competing for a given object will have a value different to $-$ in the corresponding row.

Algorithm 4 INITIALIZATION

1: Construction of the static distribution table T:
- Columns: consistent blocks of rules from R_Π: $B_{h,\alpha,\alpha',u,v}$
- Rows: pairs $(obj, membr)$ and (obj', e), for all object $obj \in \Gamma$, $obj' \in \Sigma$ and membrane $membr \in \mu$, being e a way to generically identify the environment of the skeleton of the P systems in the multienvironment system.
- Values: place $1/k$ in the element (x, y) of the table T, if the corresponding object to the row x is in their left-hand side of the block given by column y, with multiplicity k. Otherwise, keep unmarked with $-$.

2: **for** $j = 1$ **to** m **do** ▷ (Construct the expanded table T_j)
3: $T_j \leftarrow T$. ▷ (Initialize the table with the original T)
4: Add to table T_j a column for each communication rule block from R_E associated to the environment e_j, and place the value 1 in the corresponding row for (obj', e), being obj' the object appearing in the left-hand side.
5: **end for**
6: Initialize the multisets $B_j \leftarrow \emptyset$ and $R^j_{sel} \leftarrow \emptyset$

The distribution of objects among the blocks with overlapping left-hand sides is performed in selection phase 1 (algorithm 5). The expanded static table T_j is used for this purpose in each environment. Three filters are defined in order to adapt the T_j to the current state of the system. That is, to select which rule blocks are going to receive objects. The first filter will delete columns of the table corresponding to non applicable rule blocks due to the charges in the left-hand side. The second filter will delete the columns of the rule blocks with no applications in a configuration, because of the objects in the left-hand side. The goal of the third filter is to save space in the table, deleting rows with no correspondence with the non-filtered columns. These three filters are applied at the beginning of phase 1, and the result is a *dynamic table* T^t_j (for the environment j and time step t).

Filter functions for selection Phase 1

function FILTER 1(table T, configuration C) ▷ (By columns and charges)
Delete columns from table T, according to the charge of the membrane in the left-hand side of the corresponding block and in the configuration C.
return T
end function

function FILTER 2(table T, configuration C) ▷ (By columns and multiplicity)
Delete columns from table T, such that for any row $(obj, membr)$ or (obj', e), the multiplicity of that object in C multiplied by $1/k$ (value in the table), returns a number $\kappa, 0 \leq \kappa < 1$. If all the values for that column are $-$, it is also filtered.
return T
end function

function FILTER 3(table T, configuration C) ▷ (By rows and multiplicity)
Delete rows from T of pairs $(obj, membr), (obj', e)$ according to the multisets of C, those having multiplicity 0.
return T
end function

Remark that the *static table* T contains all consistent blocks in the columns. The set of all consistent blocks is unlikely to be mutually consistent. However, two non-mutually consistent blocks, $B_{h,\alpha,\alpha'_1,u,v}$ and $B_{h,\alpha,\alpha'_2,w,y}$ (assigning a different charge to the same membrane), can be filtered as follows:

- If $u \neq w$ or $v \neq y$, and if one of these multisets is not present in C_t, then one of the blocks is not applicable, and therefore will be filtered by filter 2. This situation is commonly handled by the model designers, in order to take control of the model evolution.

It is very important to have a set of mutually consistent blocks before distributing objects to the blocks. For this reason, there are two complementary methods to detect it. First, and after applying filters 1 and 2, a loop to check the mutually consistency is performed. If this method ends with an error, meaning that an inconsistency was encountered, the simulation process can finish, warning the designer with the reason. Nevertheless, it can be interesting to find a way to continue the execution by non-deterministically constructing a subset of mutually consistent blocks. Since this method can be exponentially expensive in time, it is optional for the user to whether activate it or not.

Once the columns of the *dynamic table* represent a set of mutually consistent blocks, the distribution process starts. This is carried out by browsing the rows of the table, in such a way that the values of the rows, different to $-$, will be the multiplication of:

- The normalized value respecting the row, that is, the value divided by the total sum of the row. This calculates a way to *proportionally* distribute the corresponding object along the blocks. Since it depends on the value k (multiplicity in the left-hand side), the distribution actually penalize the blocks requiring more copies of the same object, what is inspired in the amount of energy required to join individuals from the same species.
- The value in the original dynamic table (i.e. $\frac{1}{k}$). This indicates the number of possible applications of the block with the corresponding object.
- The corresponding multiplicity of the object in the current configuration C'_t. This performs the distribution of the number of copies of the object along the blocks.

After the object distribution process, the number of applications for each block is calculated by selecting the minimum value in each column. This number is then used for consuming the left-hand side from the configuration. However, this application could be not maximal. The distribution process can eventually deliver objects to blocks that are restricted by other objects. In view of that this situation may occur frequently, the distribution and the configuration update process can be A times, being A an input parameter referring to *accuracy*. The more the process is repeated, the more is the distribution accurate, but the performance of the simulation can be lower. We have experimentally see that $A = 2$ gives the best accuracy/performance ratio.

Algorithm 5 SELECTION PHASE 1: DISTRIBUTION

1: **for** $j = 1$ **to** m **do** ▷ (For each environment e_j)
2: Apply filters to table T_j using C_t, obtaining T_j^t. The filters are applied as follows:
a. $T_j^t \leftarrow T_j$
b. $T_j^t \leftarrow$ *Filter 1* (T_j^t, C_t).
c. $T_j^t \leftarrow$ *Filter 2* (T_j^t, C_t).
d. Check *mutual consistency* for the blocks remaining in T_j^t:
- Create a vector MC_j^t, of order q (number of membranes in Π), with $MC_j^t(i) = -1, 1 \leq i \leq q$.
- **for each** column $B_{h,\alpha,\alpha',u,v}$ in T_j^t, **do**
  - **if** $MC_j^t(h) = -1$ **then** $MC_j^t(h) \leftarrow \alpha'$.
  - **else if** $MC_j^t(h) = \alpha'$ **then** do nothing.
  - **else** store all the information about the inconsistency.
- **if** there was at least one inconsistency **then** report the information about the error, and optionally stop the execution (in case of not activating steps *2* and *3*.)

e. $T_j^t \leftarrow$ *Filter 3* (T_j^t, C_t).
3: *(ACTIVATE OR NOT)* Generate, from T_j^t, sub-tables formed by sets of *mutually consistent* blocks, in a maximal way in T_j^t (by the inclusion relationship). This will produce a set of sub-tables $T_{j,i}^t, i = 1, \ldots, s$.
4: *(ACTIVATE OR NOT)* Randomly select one table from $T_{j,i}^t, i = 1, \ldots, s$: $T_{j,z}^t$
5: $a \leftarrow 1$
6: **repeat**
7: Add up the values of each row in $T_{j,z}^t$. Filter the rows whose sum is 0.
8: Each element of the table is divided by the sum of the corresponding row.
9: For each pair $(obj, membr)$ and $(obj', e) \in C_t^a$, if the object in C_t^a has multiplicity $mult > 0$, all the elements of the corresponding row in $T_{j,z}^t$ are multiplied by $mult$, by the corresponding value in $T_{j,z}^t$ (calculated in the previous step), and by the original value in T_j. That is, if in the position (x, y) of the table T_j there is a value different than $-$, and the corresponding object in the row x has multiplicity $mult_{x,a,t}$ in C_t^a, then:

$$T_{j,z}^t(x,y) = \lfloor mult_{x,a,t} \cdot T_{j,z}^t(x,y) \cdot \frac{T_{j,z}^t(x,y)}{RowSum_{x,t}} \rfloor = \lfloor mult_{x,a,t} \cdot \frac{(T_{j,z}^t(x,y))^2}{RowSum_{x,t}} \rfloor$$

10: For each block b (i.e. column) appearing (i.e. not filtered) in $T_{j,z}^t$, calculate the *minimum* number of the previously calculated values, $N_b^a \geq 0$. This is the number of times the block is going to be applied. This value is accumulated to the total calculated through the iteration of the loop over a: $B^j \leftarrow B^j \cup \{< b, N_b^a >\}$
11: $C_t^{a+1} \leftarrow C_t^a - LHS(b) \cdot N_b^a$ ▷ (Delete the left-hand side of the block.)
12: $T_{j,z}^t \leftarrow$ *Filter 3*(*Filter 2* $(T_{j,z}^t, C_t^{a+1})$,C_t^{a+1}), that is, apply filters 2 and 3.
13: $a \leftarrow a + 1$
14: **until** $(a > A) \vee$ *(all the selected minimums in step 10 are* 0*)*
15: **end for**

In order to efficiently repeat the loop for A, and also before going to the next phase (maximality), it is interesting to apply again filter 2. In this way, blocks

updating the configuration and without more applications, are deleted from the table.

After phase 1, some objects may be left unevolved. It can come from the issue of having a low A value, or because the rounded value calculated in the distribution process. Due to the maximal property of P systems, after each computation step no object can be left unevolved. In order to sort out this problem, a maximality phase is applied. This phase consists of selecting those blocks whose rules can still be applied. Then, a random order on these blocks is obtained. Finally, these blocks are applied by following that order. In this phase, each rule block is applied on a maximal manner. That is, blocks consume all objects which can be consumed. In order to minimize the distortion towards the most probable blocks, this phase is left after phase 1, as a residual number of objects is expected to be consumed in this phase.

Algorithm 6 SELECTION PHASE 2: MAXIMALITY

1: **for** $j = 1$ **to** m **do** ▷ (For each environment e_j)
2: Set a random order to the blocks remaining in the last updated table $T^t_{j,z}$ in Phase 1, step *12*.
3: **for** each block b, following the previous random order **do**
4: Calculate the number of applications, N_b, of b in C^A_t (last updated configuration in Phase 1, step *11*).
5: Add N_b to the total number of applications calculated for b in each loop of phase 1, step *10*: $B^j \leftarrow B^j \cup \{< b, N_b >\}$
6: $C^A_t \leftarrow C^A_t - LHS(b) \cdot N_b$ ▷ (delete the objects in the left-hand side of block b, N_b times.)
7: $C'_t \leftarrow C^A_t$
8: **end for**
9: **end for**

After the application of the phases 1 and 2, a maximal multiset of selected (mutually consistent) blocks is computed. The output of the selection stage is, in fact, a maximal multiset of selected rules. Hence, phase 3 (algorithm 7) passes from blocks to rules, by applying the corresponding probabilities (at the local level of blocks). The rules belonging to a block are selected according to a multinomial distribution $M(N, p_1, \ldots, p_k)$, where N is the number of applications of the block, and $p_1, \ldots, p_k$ are the probabilities associated with the rules within the block $r_1, \ldots, r_k$, respectively.

Algorithm 7 SELECTION PHASE 3: PROBABILITY

1: **for** $j = 1$ **to** m **do** ▷ (For each environment e_j)
2: **for all** block $< b, N_b > \in B^j$ **do**
3: Calculate a random multinomial $M(N_b, p_{r_1}, \ldots, p_{r_{l_b}})$ with respect to the probabilities of the l_b rules $r_1, \ldots, r_{l_b}$ within the block b.
4: **for** $i = 1$ **to** l_b **do** Add the randomly calculated value n_{r_i}, using the multinomial distribution for rule r_i, to the multiset of selected rules $R^j_{sel,t} \leftarrow R^j_{sel,t} \cup \{< r_i, n_{r_i} >\}$.
5: **end for**
6: **end for**
7: Delete the multiset of selected blocks $B^j \leftarrow \emptyset$. This is useful for the next step over time T.
8: **end for**

Once the rules to be applied on the current simulation step are selected, the execution stage (algorithm 8) is applied. This stage consists on executing the previously selected multiset of rules. As the objects present on the left hand side of these rules have already been consumed, the only operation left is the application of the right-hand side of the selected rules. Therefore, for each selected rule, the objects present on the right-hand side to the corresponding membranes are added and the indicated membrane charge is set.

Algorithm 8 EXECUTION

1: **for** $j = 1$ **to** m **do** ▷ (For each environment e_j)
2: **for all** Rule $< r, n > \in R^j_{sel}$ **do** ▷ (Apply the right-hand side of the selected rules)
3: $C'_t \leftarrow C'_t + n \cdot RHS(r)$
4: Update the electrical charges of C'_t from $RHS(r)$.
5: **end for**
6: Delete the multiset of selected rules $R^j_{sel} \leftarrow \emptyset$. This is useful for the next step over time T.
7: **end for**

3.3 Running a test example

Let us consider a test example, without any biological meaning, in order to show the different behaviour of the algorithms. This test PDP system is of degree $(2, 1)$, and of the following form:

$$\Pi_{test} = (G, \Gamma, \mu, R, T, \{f_r : r \in R\}, \mathcal{M}_e, \mathcal{M}_1, \mathcal{M}_2)$$

where:

- G is an empty graph because $R_E = \emptyset$.

- $\Gamma = \{a, b, c, d, e, f, g, h\}$
- $\mu = [\ [\ ]_2\ ]_1$ is the membrane structure, and the corresponding initial multisets are:
  - $\mathcal{M}_e = \{\ b\ \}$ (in the environment)
  - $\mathcal{M}_1 = \{\ a^{60}\ \}$ (in membrane 1)
  - $\mathcal{M}_2 = \{\ a^{90}\ b^{72}\ c^{66}\ d^{30}\ \}$ (in membrane 2)
- $T = 1$, only one time step.
- The rules R to apply are:

$$r_{1.1} \equiv [\ a^4\ b^4\ c^2\ ]_2 \xrightarrow{0.7} e^2\ [\ ]_2$$

$$r_{1.2} \equiv [\ a^4\ b^4\ c^2\ ]_2 \xrightarrow{0.2} [\ e^2\ ]_2$$

$$r_{1.3} \equiv [\ a^4\ b^4\ c^2\ ]_2 \xrightarrow{0.1} [\ e\ f\ ]_2$$

$$r_2 \equiv [\ a^4\ d\ ]_2 \xrightarrow{1} f^2[\ ]_2$$

$$r_3 \equiv [\ b^5\ d^2\ ]_2 \xrightarrow{1} g^2[\ ]_2$$

$$r_4 \equiv b\ [\ a^7\ ]_1^- \xrightarrow{1} [\ h^{100}\ ]_1^-$$

$$r_5 \equiv a^3\ [\ ]_2 \xrightarrow{1} [\ e^3\ ]_2$$

$$r_6 \equiv a\ b\ [\ ]_2 \xrightarrow{1} [\ g^3\ ]_2^-$$

We can construct a set of six consistent rule blocks $B_{\Pi_{test}}$ (of the form $b_{h,\alpha,\alpha',u,v}$) from the set R of Π_{test} as follows:

- $b_1 \equiv b_{2,0,0,\emptyset,\{a^4,b^4,c^2\}} = \{r_{1.1}, r_{1.2}, r_{1.3}\}$
- $b_2 \equiv b_{2,0,0,\emptyset,\{a^4,d\}} = \{r_2\}$
- $b_3 \equiv b_{2,0,0,\emptyset,\{b^5,d^2\}} = \{r_3\}$
- $b_4 \equiv b_{1,-,-,\{b\},\{a^7\}} = \{r_4\}$
- $b_5 \equiv b_{2,0,0,\{a^3\},\emptyset} = \{r_5\}$
- $b_6 \equiv b_{2,0,-,\{a,b\},\emptyset} = \{r_6\}$

It is noteworthy that the set $B_{\Pi_{test}}$ is not mutually consistent. However, only the blocks b_1, b_2, b_3 and b_5 are applicable in the initial configuration, and they, in fact, conform a mutually consistent set of blocks. Block b_4 is not applicable since the charge of membrane 1 is neutral, and block b_6 cannot be applied because there are no b's in membrane 1.

Table 1 shows five different runs for one time step of Π_{test} using the DNDP algorithm. The values refers to the number of applications for each rule, which is

Rules	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Simulation 5
$r_{1.1}$	11	0	0	0	0
$r_{1.2}$	4	4	3	0	0
$r_{1.3}$	1	0	0	0	0
r_2	6	18	6	22	2
r_3	1	6	12	4	14
r_4	-	-	-	-	-
r_5	20	20	20	20	20
r_6	-	-	-	-	-

Table 1: Simulating Π_{test} using the DNDP algorithm

actually the output of the selection stage (and the input of the execution stage). Note that for simulation 1, the applications for $r_{1.1}$, $r_{1.2}$ and $r_{1.3}$ follows the multinomial distribution. The applications of these rules are reduced because they are competing with rules r_2 and r_3, . However, this competition leads to situations where the applications of the block b_1 does not follow a multinomial distribution. It comes from the fact of using a random order over the rules, but not over the blocks. Rules having a probability equals to 1 are more restrictive on the competitions because they are applied in a maximal way in their turn. This is the reason because on simulations 4 and 5, none of the rules $r_{1.i}, 1 \leq i \leq 3$ are applied.

This behaviour could create a distortion of the reality described in the simulated model. But it is usually appeased running several simulations and making a statistical study. Finally, rules not competing for objects are applied as is, in a maximal way. For example, rule r_5 is always applied 20 times because its probability is equal to 1.

In the following, the test example is executed using the DCBA. The main results of the different phases of the process is also detailed.

In the initialization phase, the static table is created, containing all the consistent blocks. The static table of the Π_{test} P system is showed in table 2. As shown, the values inside the cells of the table represents the inverse $(1/k)$ of the multiplicity of the object (in the membrane, as specified in the row) inside the block indicated in the header of the column.

Once the static table has been initialized, the simulation main loop runs for the stated steps. Then, for each step of computation, the selection and execution of rules runs, as illustrated in the following paragraphs.

The selection starts with the distribution phase. The needed filters are performed, causing some objects and blocks to be discarded, as they need not present charges and/or objects. Then the corresponding calculus take place, getting the minimum number of applications of each way. The result of the selection phase 1 of the step 1 is showed in table 3. The sum of the previously obtained values is showed in the last column. Then, the possible number of applications of a block is calculated for each object, considering its multiplicity in the current configuration

Objects	**Consistent Blocks**					
	$b_{2,0,0,\emptyset,\{a^4,b^4,c^2\}}$	$b_{2,0,0,\emptyset,\{a^4,d\}}$	$b_{2,0,0,\emptyset,\{b^5,d^2\}}$	$b_{1,-,-,\{b\},\{a^7\}}$	$b_{2,0,0,\{a^3\},\emptyset}$	$b_{2,0,-,\{a,b\},\emptyset}$
$< a$,2>	1/4	1/4	-	-	-	-
$< b$,2>	1/4	-	1/5	-	-	-
$< c$,2>	1/2	-	-	-	-	-
$< d$,2>	-	1/1	1/2	-	-	-
$< a$,1>	-	-	-	1/7	1/3	1/1
$< b$,1>	-	-	-		-	1/1
$< b$,e>	-	-	-	1/1	-	-

Table 2: Static table

and the block, and the relation with the sum of the row. This relation somehow captures the proportion of objects to be initially assigned to each block. Then, the minimum number of each block (given by the column) is calculated.

Objects	**Consistent Blocks**				**Sum**
	$b_{2,0,0,\emptyset,\{a^4,b^4,c^2\}}$	$b_{2,0,0,\emptyset,\{a^4,d\}}$	$b_{2,0,0,\emptyset,\{b^5,d^2\}}$	$b_{2,0,0,\{a^3\},\emptyset}$	
$< a$,2> * 90	0.25 | 11	0.25 | 11	-	-	0.5
$< b$,2> * 72	0.25 | 10	-	0.2 | 6	-	0.45
$< c$,2> * 66	0.5 | 33	-	-	-	0.5
$< d$,2> * 30	-	1.0 | 20	0.5 | 5	-	1.5
$< a$,1> * 60	-	-	-	0.33 | 20	0.33
Applications	10	11	5	20	

Table 3: Selection Phase 1 - Distribution

The next phase, maximality, starts from the remaining objects, selecting new applications of the blocks in a maximal way. The result of this phase is showed in table 4. This table presents the remaining objects (the ones not assigned in phase 1) and the possible blocks to be selected. The blocks are chosen in a random way, as shown in algorithm 6, and the possible applications of the block are calculated. This process guarantees a maximal set of blocks to be selected, with a maximal number of applications of each block. The last row, applications, shows that the block $b_{2,0,0,\emptyset,\{a^4,b^4,c^2\}}$ is applying 1 time, additional to the number of applications calculated in the distribution phase.

Then the phase 3, probability, take place. For each block selected in the previous phases, its number of applications is divided among the rules being part of the

Objects	**Consistent Blocks**		
	$b_{2,0,0,\emptyset,\{a^4,b^4,c^2\}}$	$b_{2,0,0,\emptyset,\{a^4,d\}}$	$b_{2,0,0,\emptyset,\{b^5,d^2\}}$
$< a,2 >$ * 6	-	-	-
$< b,2 >$ * 7	-	-	-
$< c,2 >$ * 46	-	-	-
$< d,2 >$ * 9	-	-	-
Applications	1	-	-

Table 4: Selection Phase 2 - Maximality

block, according to their probabilities. As a result, the number of applications of each rule is obtained, as showed in table 5.

Rules	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Simulation 5
$r_{1.1}$	7	10	7	6	7
$r_{1.2}$	3	0	4	1	2
$r_{1.3}$	1	1	5	3	1
r_2	11	11	11	12	12
r_3	5	5	5	6	6
r_4	-	-	-	-	-
r_5	20	20	20	20	20
r_6	-	-	-	-	-

Table 5: Simulating Π_{test} using the DCBA algorithm

It is noteworthy that the selection of rules belonging to block 1 $\{r_{1.i}, 1 \leq i \leq 3\}$, in table 5, always follows a multinomial distribution respecting the 3 probabilities. This solves the drawback we showed on table 1. Moreover, it can be seen that the maximality sometimes can give one more application to blocks 2 and 3, in spite of keeping the original 10 applications for block 1 from phase 1. In any case, the number of applications is proportionally distributed, avoiding the distortion of using a random order over the blocks (or rules), as made in the DNDP algorithm.

3.4 Implementation in pLinguaCore

In [11], a Java library called *pLinguaCore* was presented under GPL license. It includes parsers to handle input files and built–in simulators to handle different P System based models. It is not a closed product because developers with knowledge of Java can add new components to the library. Within the scope of this paper, pLinguaCore has been upgraded to provide an implementation of the DBCA,

thus extending its existing probabilistic model simulation algorithms support. Along with the inclusion of other extensions, regarding to models such as Spiking Neural P Systems and Numerical P Systems, current version of the library, named *pLinguaCore 3.0*, and featuring an implementation of the introduced DBCA can be downloaded from [21]. In what follows, details of the implementation of the DBCA in pLinguaCore are shown. Data structures, methods, code optimization and bug fixes are reviewed. Going Top-down, Java classes involved in the implementation:

- *DynamicMatrix.* It provides an implementation for the main operations of the DBCA.

  DynamicMatrix is built as a dynamic map indexed by MatrixKey class objects. MatrixKey objects are implemented as a pair of (MatrixRow, MatrixColumn) class objects. Associated to each MatrixKey object within the map, multiplicity k of the object specified by the *MatrixRow row* in the left hand side of the rule specified by the *MatrixColumn column* is stored. Note that k is stored instead of $1/k$ for accuracy reasons.

  As different filters are applied over the DynamicMatrix object, a couple of lists of MatrixRow and MatrixColumn objects respectively are associated to the matrix to keep track of its *valid cells*. Removal of elements from these lists is performed when filters are applied, while the DynamicMatrix object itself is reset in every step of the main loop of selection phase. Thus, DynamicMatrix object can be viewed as a *hash table* of multiplicities that allows a significant reduction of the required amount of memory for execution of the DBCA.

  Also, attributes that stores the sum of the multiplicities of the objects in the matrix by row as well as the minimum of the columns are included in the DynamicMatrix class. Inconsistent blocks are controlled by means of a list of pairs of MatrixColumn objects.

  DynamicMatrix class directly extends from StaticMatrix class. Methods in DynamicMatrix implements the DBCA different phases themselves, remarkably:

  - *initData()* initializes valid rows and columns lists in the DynamicMatrix object, clearing up them; also application of rules data structure is initialized.
  - *filterColumns1()* computes valid columns and associates them to the DynamicMatrix object; applies Filter 1 to these columns;
  - *filterColumns2()* applies Filter 2 over valid columns associated to the DynamicMatrix object, removing the required ones.
  - *checkMutualConsistency()* checks mutual consistency over blocks of the DynamicMatrix object; if any inconsistency is found, an exception is thrown and execution of the simulator is halted; a message listing the mutual inconsistent blocks found is shown to the user.

  - *initFilterRows()* computes valid rows and associates them to the DynamicMatrix object; applies Filter 3 to these rows.
  - *filterRows()* applies Filter 3 to valid rows, removing the required ones; this method is called inside the main loop of the selection phase, while the previous one is called outside, at the beginning of this phase.
  - *normalizeRowsAndCalculateMinimums()* implements the main loop of selection phase.
  - *maximality()* implements maximality phase.
  - *executeRules()* implements execution phase; remarkably, multinomial distribution is computed by computing binomial distributions, implemented through the specialized CERN Java library (`cern.jet.random.Binomial`).

- *StaticMatrix.* Provides an implementation for the static matrix used by the DBCA. Similarly to DynamicMatrix class, cells within the matrix are stored as a map indexed by MatrixKey class objects, each one of them associated to a multiplicity. A couple of immutable lists of MatrixRow and MatrixColumn class objects determines the structure of the matrix. Contents of the cells are fixed once initialized.

- *MatrixRow.* Provides an implementation for rows featured in DynamicMatrix, StaticMatrix and MatrixKey objects. Implemented by a pair of String objects representing *object and membrane label* respectively, it also provides a method for computing the validity of the row, i.e. to determine if the row has to be kept within the DynamicMatrix object with respect to a given environment.

- *MatrixColumn.* Provides an implementation for columns featured in DynamicMatrix, StaticMatrix and MatrixKey objects. An abstract class, its extended and implemented by a couple of classes representing the two kinds of rule blocks:

  - SkeletonRulesBlock, which implements blocks of skeleton rules.

  - EnvironmentRulesBlock, which implements blocks of environment rules.

  Both classes have the same structure: a *single* object to store the common left hand rule side of the rule, plus a *collection* to store the several right hand rule side objects that conforms the block. Also, each one provides an specific method for computing the validity of the corresponding column within the dynamic matrix.

To conclude, let us note that while conducting the DBCA implementation, several bugs have been fixed in pLinguaCore, notably some of them regarding to the way in which rules are parsed and stored, thus applying beyond the scope of

the DBCA an affecting to implementation of probabilistic models simulators as a whole:

- Multisets of objects are now taken into account while checking rule blocks. In previous versions of pLinguaCore, when checking of the consistency of probabilities of a rule block was conducted (i.e. checking that sum of probabilities of the rules must equal to one), multiplicities of objects in the left hand side of the rules were ignored.

- Issues with "intentional duplicate rules" solved assigning an unique identifier for every rule within the scope of probabilistic models. Issues found were:

  – Instantiation of parameters in syntactically different rule schemes for some models produced duplicated rules and caused the parser to throw an error and halt. As this duplicity proved intentional, the parser was modified subsequently to take it into account.

  – Probability was not taken into account when differencing rules. This made the parser to discard a rule syntactically identical, except for its probability, to a previous parsed one.

4 Validation

4.1 Improved model for the scavenger bird ecosystem

In this section, it is presented a novel model for an ecosystem related to the Bearded Vulture in the Pyrenees (NE Spain), by using PDP systems. This model is an improved model of which is provided in [5]. The Bearded Vulture (*Gypaetus barbatus*) is an endangered species in Europe that feeds almost exclusively on bone remains of wild and domestic ungulates. In this model, the evolution of six species is studied: The Bearded Vulture and five subfamilies of domestic and will ungulates upon which the vulture feeds.

The model consists of a PDP system of degree $(2, 1)$,

$$\Pi = (G, \Gamma, \mu, R, T, \{f_r : \ r \in R\}, \mathcal{M}_1, \mathcal{M}_2)$$

where:

- G is an empty graph because $R_E = \emptyset$.
- In the alphabet Γ, we represent the six species of the ecosystem (index i is associated with the species and index j is associated with their age, and the symbols X, Y and Z represent the same animal but in different states); it

also contains the auxiliary symbol B, which represents 0.5 kg of bones, and C, which allows a change in the polarization of the membrane labeled by 2 at a specific stage.

$$\Gamma = \{X_{i,j}, Y_{i,j}, Z_{i,j} : 1 \leq i \leq 7, 0 \leq j \leq k_{i,4}\} \cup \{B, C\}$$

The species are the following:
- Bearded Vulture ($i = 1$)
- Pyrenean Chamois ($i = 2$)
- Red Deer Female ($i = 3$)
- Red Deer Male ($i = 4$)
- Fallow Deer ($i = 5$)
- Roe Deer ($i = 6$)
- Sheep ($i = 7$)

- $\mu = [\ [\ ]_2\ ]_1$ is the membrane structure, and the corresponding initial multisets are:
  - $\mathcal{M}_1 = \{\ X_{i,j}^{q_{i,j}} : 1 \leq i \leq 7, 0 \leq j \leq k_{i,4}\}$
  - $\mathcal{M}_2 = \{\ C, B^{\alpha}\}$

    where $\alpha = \lceil \sum_{j=1}^{21} q_{1,j} \cdot 1.10 \cdot 682 \rceil$

    Value α represents an external contribution of food which is added during the first year of study so that the Bearded Vulture survives. In the formula, $q_{1,j}$ represents the number of j years of age of Bearded Vultures, the finality of constant factor 1.10 is to guarantee enough food for 10% population growth. At present, the population growth is estimated an average 4%, but this value can reach higher values. Thus, to avoid problems related with the underestimation of this value the first year we estimated the population growth (overestimated) at 10%. The constant value 682 represents the amount of food needed per year for a Bearded Vulture pair to survive.

- Each year in the real ecosystem is simulated by 3 computational steps, so $T = 3 \cdot Years$, where $Years$ is the number of years to simulate.

- The rules R to apply are:
  - Reproduction rules for ungulates

    Adult males

    $r_{0,i,j} \equiv [X_{i,j}]_1 \xrightarrow{1-k_{i,13}} [Y_{i,j}]_1 : k_{i,2} \leq j \leq k_{i,4}, 2 \leq i \leq 7$

    Adult females that reproduce

    $r_{1,i,j} \equiv [X_{i,j}]_1 \xrightarrow{k_{i,5}k_{i,13}} [Y_{i,j}, Y_{i,0}]_1 : k_{i,2} \leq j < k_{i,3}, 2 \leq i \leq 7, i \neq 3$

    Red Deer females produce 50% of female and 50% of male springs

    $r_{2,j} \equiv [X_{3,j}]_1 \xrightarrow{k_{3,5}k_{3,13}{}^{0.5}} [Y_{3,j}Y_{3,0}]_1 : k_{3,2} \leq j < k_{3,3}$

    $r_{3,j} \equiv [X_{3,j}]_1 \xrightarrow{k_{3,5}k_{3,13}{}^{0.5}} [Y_{3,j}Y_{4,0}]_1 : k_{3,2} \leq j < k_{3,3}$

Fertile adult females that do not reproduce

$r_{4,i,j} \equiv [X_{i,j}]_1 \xrightarrow{(1-k_{i,5})k_{i,13}} [Y_{i,j}]_1 : k_{i,2} \leq j < k_{i,3}, 2 \leq i \leq 7$

Not fertile adult females

$r_{5,i,j} \equiv [X_{i,j}]_1 \xrightarrow{k_{i,13}} [Y_{i,j}]_1 : k_{i,3} \leq j \leq k_{i,4}, 2 \leq i \leq 7$

Young ungulates that do not reproduce

$r_{6,i,j} \equiv [X_{i,j}]_1 \xrightarrow{1} [Y_{i,j}]_1 : 0 \leq j < k_{i,2}, 2 \leq i \leq 7$

– Growth rules for the Bearded Vulture

$r_{7,j} \equiv [X_{1,j}]_1 \xrightarrow{k_{1,6}+k_{1,10}} [Y_{1,k_{1,2}-1}Y_{1,j}]_1 : k_{1,2} \leq j < k_{1,4}$

$r_{8,j} \equiv [X_{1,j}]_1 \xrightarrow{1-k_{1,6}-k_{1,10}} [Y_{1,j}]_1 : k_{1,2} \leq j < k_{1,4}$

$r_9 \equiv [X_{1,k_{1,4}}]_1 \xrightarrow{k_{1,6}} [Y_{1,k_{1,2}-1}Y_{1,k_{1,4}}]_1$

$r_{10} \equiv [X_{1,k_{1,4}}]_1 \xrightarrow{1-k_{1,6}} [Y_{1,k_{1,4}}]_1$

– Mortality rules for ungulates

Young ungulates which survive

$r_{11,i,j} \equiv Y_{i,j}[\,]_2 \xrightarrow{1-k_{i,7}-k_{i,8}} [Z_{i,j}]_2 : 0 \leq j < k_{i,1}, 2 \leq i \leq 7$

Young ungulates which die

$r_{12,i,j} \equiv Y_{i,j}[\,]_2 \xrightarrow{k_{i,8}} [B^{k_{i,11}}]_2 : 0 \leq j < k_{i,1}, 2 \leq i \leq 7$

Young ungulates which are retired from the ecosystem

$r_{13,i,j} \equiv Y_{i,j}[\,]_2 \xrightarrow{k_{i,7}} [\,]_2 : 0 \leq j < k_{i,1}, 2 \leq i \leq 7$

Adult ungulates that do not reach the average life expectancy

Those which survive

$r_{14,i,j} \equiv Y_{i,j}[\,]_2 \xrightarrow{1-k_{i,10}} [Z_{i,j}]_2 : k_{i,1} \leq j < k_{i,4}, 2 \leq i \leq 7$

Those which die

$r_{15,i,j} \equiv Y_{i,j}[\,]_2 \xrightarrow{k_{i,10}} [B^{k_{i,12}}]_2 : k_{i,1} \leq j < k_{i,4}, 2 \leq i \leq 7$

Ungulates that reach the average life expectancy

Those which die in the ecosystem

$r_{16,i} \equiv Y_{i,k_{i,4}}[\,]_2 \xrightarrow{k_{i,9}+(1-k_{i,9})k_{i,10}} [B^{k_{i,12}}]_2 : 2 \leq i \leq 7$

Those which die and are retired from the ecosystem

$r_{17,i} \equiv Y_{i,k_{i,4}}[\,]_2 \xrightarrow{(1-k_{i,9})(1-k_{i,10})} [\,]_2 : 2 \leq i \leq 7$

– Mortality rules for the Bearded Vulture

$r_{18,j} \equiv Y_{1,j}[\,]_2 \xrightarrow{1-k_{1,10}} [Z_{1,j}]_2 : k_{1,2} \leq j < k_{1,4}$

$r_{19,j} \equiv Y_{1,j}[\,]_2 \xrightarrow{k_{1,10}} [\,]_2 : k_{1,2} \leq j < k_{1,4}$

$r_{20} \equiv Y_{1,k_{1,4}}[\,]_2 \xrightarrow{1} [Z_{1,k_{1,2}-1}]_2$

$r_{21} \equiv Y_{1,k_{1,2}-1}[\,]_2 \xrightarrow{1} [Z_{1,k_{1,2}-1}]_2$

– Feeding rules

$r_{22,i,j} \equiv [Z_{i,j}B^{k_{i,14}}]_2 \xrightarrow{1} X_{i,j+1}[\,]_2^+ : 0 \leq j \leq k_{i,4}, 1 \leq i \leq 7$

– Balance rules

Elimination of remaining bones

$r_{23} \equiv [B]_2^+ \xrightarrow{1} [\,]_2$

Adult animals that die because they have not enough food

$r_{24,i,j} \equiv [Z_{i,j}]_2^+ \xrightarrow{1} [B^{k_{i,12}}]_2 : k_{i,1} \leq j \leq k_{i,4}, 1 \leq i \leq 7$

Young animals that die because the have not enough food

$r_{25,i,j} \equiv [Z_{i,j}]_2^+ \xrightarrow{1} [B^{k_{i,11}}]_2 : 0 \leq j < k_{i,1}, 1 \leq i \leq 7$

Change the polarization

$r_{26} \equiv [C]_2^+ \xrightarrow{1} [C]_2$

- The constants associated with the rules have the following meaning:
  - $k_{i,1}$: Age at which adult size is reached. This is the age at which the animal consumes food as an adult does, and at which, if the animal dies, the amount of biomass it leaves behind is similar to the total left by an adult. Moreover, at this age it will have surpassed the critical early phase during which the mortality rate is high.
  - $k_{i,2}$: Age at which it begins to be fertile.
  - $k_{i,3}$: Age at which it stops being fertile.
  - $k_{i,4}$: Average life expectancy in the ecosystem.
  - $k_{i,5}$: Fertility ratio (number of descendants by fertile females).
  - $k_{i,6}$: Population growth (this quantity is expressed in terms of 1).
  - $k_{i,7}$: Animals retired from the ecosystem in the first years, age $< k_{i,1}$ (this quantity is expressed in terms of 1).
  - $k_{i,8}$: Natural mortality ratio in first years, age $< k_{i,1}$ (this quantity is expressed in terms of 1).
  - $k_{i,9}$: 0 if the live animals are retired at age $k_{i,4}$, in other cases, the value is 1.
  - $k_{i,10}$: Mortality ratio in adult animals, age $\geq k_{i,1}$ (this quantity is expressed in terms of 1).
  - $k_{i,11}$: Amount of bones from young animals, age $< k_{i,1}$.
  - $k_{i,12}$: Amount of bones from adult animals, age $\geq k_{i,1}$.
  - $k_{i,13}$: Proportion of females in the population (this quantity is expressed in terms of 1).

  - $k_{i,14}$: Amount of food necessary per year and breeding pair (1 unit is equal to 0.5 kg of bones).

- In [5], they can be found actual values for the constants associated with the rules as well as actual values for the initial populations $q_{i,j}$ for each species i with age j. There are two sets of initial populations values, one beginning on year 1994 and another one beginning on year 2008.

4.2 Simulation results

In [5], a simulator for the model was presented. The authors show a comparison of the results provided by the simulator and actual data obtained from the ecosystem. That simulator was written in C++ and the rules were implemented directly on the source code. So, that is a simulator implemented *ad hoc* for the model. The simulator does not implement any described simulation algorithm for P systems and does not implement any generic method to define P systems. We have found that *ad hoc* simulators like the one presented in [5] have a strong coupling design and it is a problem for debugging. So, if the simulator does not reproduce the expected behaviour of the model, what is causing the problem?. In that situation, we could think that:

1. The model is wrong.
2. The rules are not correctly written in the source code.
3. The semantics of the model is not correctly implemented in the source code.

It is very difficult to find the cause of the problem with a strong coupling software design. Moreover, if we think that the cause of the problem is, for instance, 2, but it is really 1 or 3, then we can introduce new errors trying to correct it.

From a software engineering point of view it is very important to decouple software components, that is the point of view of P-Lingua and pLinguaCore [21]:

- The model is designed on a paper.
- The rules are written on a P-Lingua file. So, the parser checks the syntactical/semantics errors.
- The semantics of the model is implemented on the pLinguaCore library following a good described simulation algorithm.

PLinguaCore is a simulation library that accepts the input written in P-Lingua and provides simulations of the defined P systems. For each type of P system, there are one or more simulation algorithms implemented in pLinguaCore. It is a software framework, so it can be expanded with new simulation algorithms.

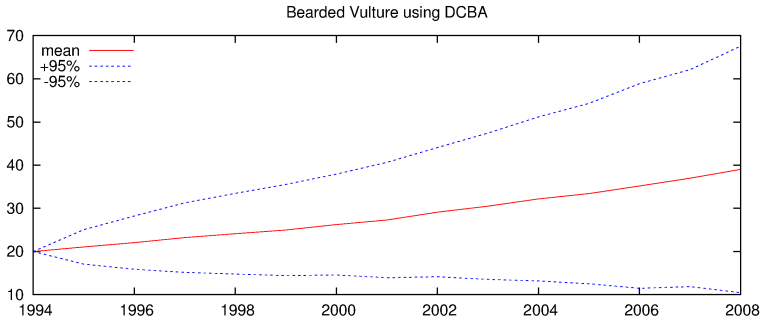
Thus, we have expanded the pLinguaCore library to include the DCBA simulation algorithm for PDP systems, the current version of pLinguaCore is 3.0 and it can be downloaded from [21].

In this section, we use the model of the Bearded Vulture described above to compare the simulation results produced by the pLinguaCore library using two different simulation algorithms: DNDP [14] and DCBA. We also compare the

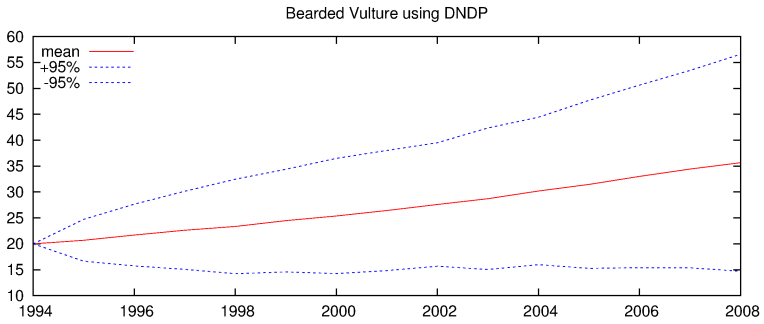
results of the implemented simulation algorithms with the results provided by the C++ *ad hoc* simulator and with the actual ecosystem data obtained from [5]. In [22] it can be found the P-Lingua file which defines the model and instructions to reproduce the comparisons.

We have set the initial population values with the actual ecosystem values for year 1994. For each simulation algorithm we have made 1000 simulations of 14 years, that is, 42 computational steps. The simulation workflow have been implemented on a Java program that runs over the pLinguaCore library (this Java program can be downloaded from [22]). For each simulated year (3 computational steps), the Java program counts the number of animals for each species i, that is: $X_i = \sum_{j=0}^{k_{i,4}} X_{i,j}$. After 1000 simulations, the Java program calculates average values for each year and species and writes the output to a text file. Finally, we have used the GnuPlot software [20] to produce population graphics.

In figures 1, 2, 3, 4 ,5, 6 and 7 the population graphics for each species and simulation algorithm are represented.



(a) Using DCBA



(b) Using DNDP

Fig. 1: Evolution of the Bearded Vulture birds

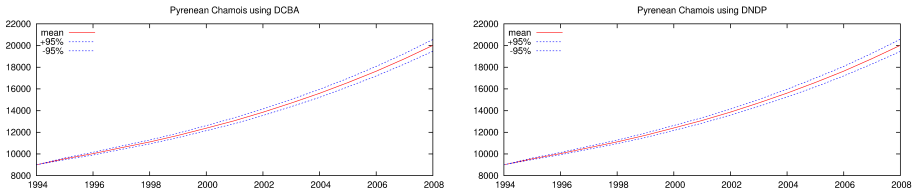


Fig. 2: Evolution of the Pyrenean Chamois

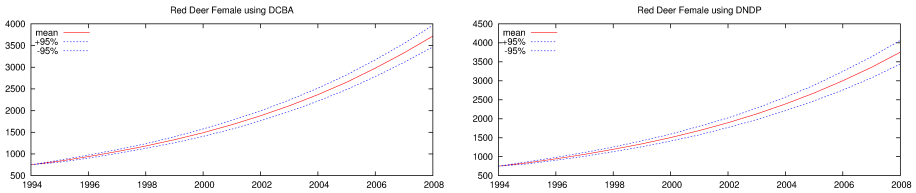


Fig. 3: Evolution of the female Red Deer

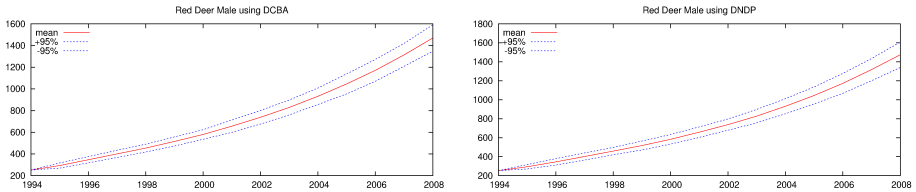


Fig. 4: Evolution of the male Red Deer

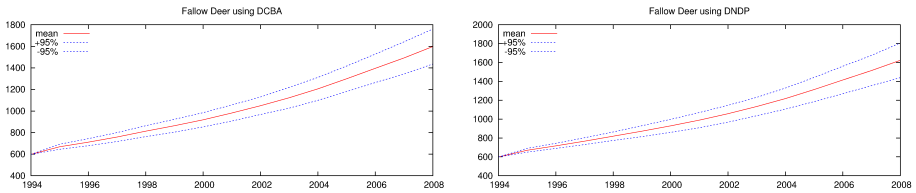


Fig. 5: Evolution of the Fallow Deer

The main difference between DNDP and DCBA algorithms is the way the algorithms distribute the objects between different rule blocks that compete for

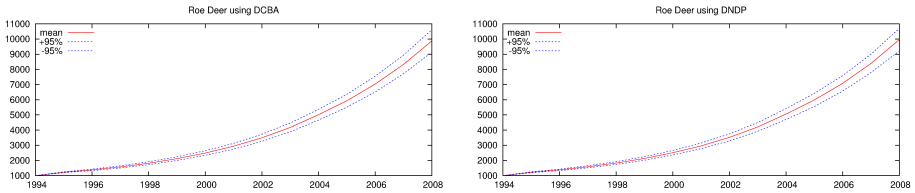


Fig. 6: Evolution of the Roe Deer

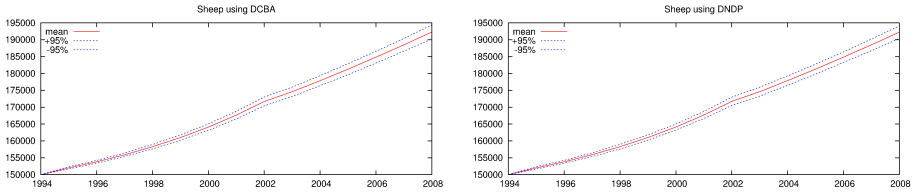


Fig. 7: Evolution of the Sheep

the same objects. In the model, the behavior of the ungulates are modeled by using rule blocks that do not compete for objects. So, the simulator provides similar results for both DCBA and DNDP algorithms. In the case of the Bearded Vulture, there are a set of rules $r_{22,i,j}$ that compete for B objects because $k_{1,14}$ is not 0 (the Bearded Vulture needs to feed on bones to survive). The $k_{i,14}$ constants are 0 for ungulates, $2 \leq i \leq 7$, because they do not need to feed on bones to survive. The initial amount of bones and the amount of bones generated during the simulation is enough to support the Bearded Vulture population regardless the way the simulation algorithm distributes the bones between vultures of different ages (rules $r_{22,1,j}$). By the way, there are a small initial population of bearded vultures (20 pairs), because of that, we can see differences between the results with DCBA, DNDP, C++ simulator and actual ecosystem data for the Bearded Vulture (39 bearded vultures with DCBA for year 2008, 36 with DNDP, 38 with the C++ simulator and 37 on the actual ecosystem).

In figure 8 it is showed the comparison between the actual data for year 2008 and the simulation results by using the C++ *ad hoc* simulator, the DNDP algorithm and the DCBA algorithm implemented in pLinguaCore. In the case of the Pyrenean Chamois, there is a difference between the actual population data on the ecosystem (12000 animals) and the results provided by the other simulators (above 20000 animals), this is because the population of Pyrenean Chamois was restarted on year 2004 [5]. Taking this into account, we can see that all the simulators behave in a similar way for the above model and they can reproduce the actual data after 14 simulated years. So, the DCBA algorithm is able to reproduce the semantics of PDP systems and it can be used to simulate the behavior of actual ecosystems by means of PDP systems.

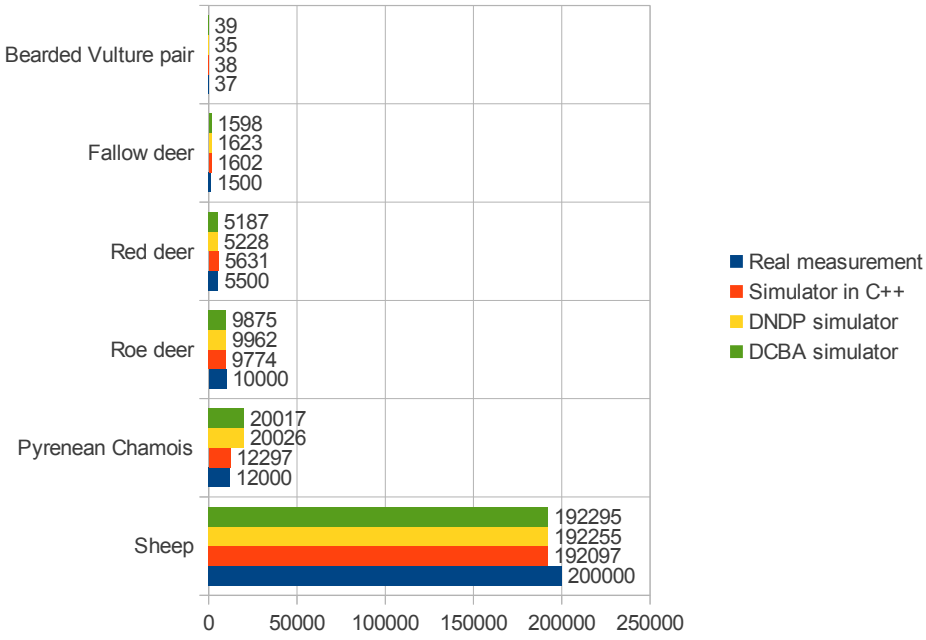


Fig. 8: Data of the year 2008 from: real measurements of the ecosystem, original simulator in C++, simulator using DNDP and simulator using DCBA.

5 Conclusions and Future Work

In this paper we have introduced a novel algorithm for Population Dynamics P systems (PDP systems), called Direct distribution based on Consistent Blocks Algorithm (DCBA). This new algorithm performs an object distribution along the rules that eventually compete for objects. The main procedure is divided into two stages: selection and execution. Selection stage is also divided into three micro-phases: phase 1 (distribution), where by using a table and the construction of rule blocks, the distribution process takes place; phase 2 (maximality), where a random order is applied to the remaining rule blocks in order to assure the maximality condition; and phase 3 (probability), where the number of application of rule blocks is translated to application of rules by using random numbers respecting the probabilities.

By using a test example, it is shown how this new algorithm solves some drawbacks in its predecessor, the DNDP algorithm. Moreover, both algorithms are validated towards a real ecosystem model (the bearded vulture birds), showing that they reproduce the same results than the original simulator written in C++.

Finally, some details and updates of its implementation in the pLinguaCore framework are provided.

The accelerators in High Performance Computing offers new approaches to accelerate the simulation of P systems and Population Dynamics [6]. An initial parallelization work of the DCBA by using multi-core processors is described in [12]. The analysis of the two parallel levels (simulations and environments), and the speedup achieved by using the different cores, make interesting the search for more parallel architectures. For the near future work, we will use the massively parallel architectures inside the graphics cards (GPUs) using CUDA. We will adapt and scale the DCBA algorithm using the CUDA programming model, and develop a parallel simulator for GPU based systems.

Acknowledgments

The authors acknowledge the support of "Proyecto de Excelencia con Investigador de Reconocida Valía" of the "Junta de Andalucía" under grant P08-TIC04200, and the support of the project TIN2009-13192 of the "Ministerio de Educación y Ciencia" of Spain, both co-financed by FEDER funds.

References

1. D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri. Modelling metapopulations with stochastic membrane systems, Biosystems, 91 (2008), 499–514.
2. L. Bianco, V. Manca, L. Marchetti, M. Petterlini. Psim: a simulator for biomolecular dynamics based on P systems, IEEE Congress on Evolutionary Computation (2007), 883–887
3. 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, Natural Computing, 10, 1 (2011), 39-53.
4. 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, Membrane Computing, 10th International Workshop, LNCS 5957 (2010), 182–195.
5. M. Cardona, M.A. Colomer, M.J. Pérez-Jiménez, D. Sanuy, A. Margalida. Modeling ecosystem using P systems: The bearded vulture, a case study. Membrane Computing, 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers. Lecture Notes in Computer Science, 5391 (2009), 137-156.
6. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Simulation of P systems with active membranes on CUDA, Briefings in Bioinformatics, 11, 3 (2010), 313–322
7. S. Cheruku, A. Păun, F.J. Romero-Campero, M.J. Pérez-Jiménez, O.H. Ibarra. Simulating FAS-induced apoptosis by using P systems, Progress in Natural Science, 17, 4 (2007), 424–431.

8. M.A. Colomer, S. Lavín, I. Marco, A. Margalida, 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, Membrane Computing, 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010, Revised Selected Papers. LNCS, 6501 (2011), 144–159.
9. M.A. Colomer, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos. Comparing simulation algorithms for multienvironment probabilistic P system over a standard virtual ecosystem. Natural Computing, online version (http://dx.doi.org/10.1007/s11047-011-9289-2).
10. M.A. Colomer, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. Simulating Tritrophic Interactions by Means of P Systems. *Proceedings of the 5^{th} IEEE International Conference on Bio-Inspired Computing: Theories and Applications*, vol. 2 (2010), 1621–1628.
11. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, Agustín Riscos-Núñez. An overview of P-Lingua 2.0, Membrane Computing, 10th International Workshop, LNCS 5957 (2010), 264–288.
12. M.A. Martínez-del-Amor, I. Karlin, R.E. Jensen, M.J. Pérez-Jiménez, A.C. Elster. Parallel Simulation of Probabilistic P Systems on Multicore Platforms. In this volume.
13. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, F. Sancho-Caparrini. A simulation algorithm for multienvironment probabilistic P systems: A formal verification. International Journal of Foundations of Computer Science, 22, 1 (2011), 107–118.
14. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, M.A. Colomer. A new simulation algorithm for multienvironment probabilistic P systems,*Proceedings of the 5^{th} IEEE International Conference on Bio-Inspired Computing: Theories and Applications*, vol. 1 (2010), 59–68,
15. V. Nguyen, D. Kearney, G. Gioiosa. An algorithm for non-deterministic object distribution in P systems and its implementation in hardware, Membrane Computing, 9th International Workshop, LNCS 5391 (2009), 325–354.
16. G. Păun. Computing with membranes. Journal of Computer and System Sciences, 61, 1 (2000), pp. 108–143, and Turku Center for Computer Science-TUCS Report No 208.
17. Gh. Păun, F.J. Romero-Campero. Membrane Computing as a Modeling Framework. Cellular Systems Case Studies, Formal Methods for Computational Systems Biology, LNCS, 5016 (2008), 168–214.
18. M.J. Pérez-Jiménez, F.J. Romero-Campero. P systems, a new computational modelling tool for systems biology, Transactions on Computational Systems Biology VI, LNCS 4220 (2006), 176–97.
19. G. Terrazas, N. Krasnogor, M. Gheorghe, F. Bernardini, S. Diggle, M. Cámara. Environment aware P-System model of quorum sensing, New Computational Paradigms, LNCS 3526 (2005), 473–485.
20. The GNUplot web page. `http://www.gnuplot.info`
21. The P-Lingua web page. `http://www.p-lingua.org`
22. The Bearded Vulture ecosystem model in P-Lingua. `http://www.p-lingua.org/wiki/index.php/bvBWMC12`

Two Topics Ahead Membrane Computing

Adam Obtułowicz

Institute of Mathematics, Polish Academy of Sciences
Śniadeckich 8, P.O.B. 21, 00-956 Warsaw, Poland
e-mail: A.Obtulowicz@impan.pl

Summary. Two topics from the area of probability theory and randomness challenging membrane computing together with open problems and research proposals are discussed.

1 Introduction

We outline in the paper the following two topics challenging membrane computing [16]:

1) new mathematical approaches to causes and origins of uncertainty resulting from critique of old approaches to probability and randomness,
2) western polyphony musical scores as predecessor of an exact (as mathematical) approach to concurrency and parallelism in computer science: looking for mutual inspiration.

These topics together with new open problems and research proposals for membrane computing are discussed in subsequent Sections 2 and 3, respectively.

2 New mathematical approaches to uncertainty with a regard to membrane computing

The critical discussion of foundations of probability theory from the points of view of quantum theory[a], computation theory[b], and philosophy (general methodology) of exact sciences[c] inspired the following new mathematical approaches to the causes and origins of uncertainty:

[a] see, e.g., the papers and book by L. Accardi, D. Aerst and I. Pitowsky quoted in Subsection 1.2 of [1].

[b] see [18] for some brief and comprehensive summary of the discussion of a concept of randomness in a context of algorithmic information by various scientists.

[c] see, e.g., the paper [5] due to M. Bunge and his other papers quoted in [5].

A1) measurement (quantum) uncertainty approached by nonclassical (non-Bayesian, non-Markovian) probability theory proposed among others in [1], where also some new non-physical applications of quantum formalism are presented,
A2) algorithmic approaches to randomness, see [18] for some their survey,
A3) interactive randomization (e.g., via oracles), cf. [2], versus classical and massively parallel Monte Carlo randomizations used e.g. in [7], [12] in classical case, and in [14], [15] in massively parallel case, where the massively parallel randomization was inspired also by the critical discussion of massive quantum parallelism in [17], [6].

The attempts to fill the gap between formal reasoning about correctness of programs and heuristic estimations of error probability and computation time of randomized algorithms (cf., e.g., [12], [13]) resulted in an invention of

A4) probabilistic functional programming systems approached by monad theory and related systems of proving correctness of probabilistic programs (e.g. in Coq), cf. [3], [8].

The approaches outlined in A1)–A4) give rise to the following open problems and research proposals for membrane computing:

P1) searching for non-classical probabilistic P systems respecting measurement interactions and eventually simulating quantum computers (see A1)), where the P systems in [14], [15] may be treated as an attempt of this simulation respecting quantum massive parallelism,
P2) establishing the relationships (eventually an equivalence) between interactive randomization and Monte Carlo massively parallel randomization (see A2), A3)), where some ideas from membrane computing, like assembly of massively parallel computing device by membrane division, are applied, see [14], [15],
P3) modifications of P lingua [19] programming environment by introducing probabilistic aspects like in the case of probabilistic functional programming, see A4).

Similar ideas to P3) have been already discussed in [10].

3 Western polyphony musical scores as a predecessor of an approach to concurrency and parallelism in computer science

Western polyphony from Middle Ages, through J. S. Bach's (implied) polyphony, to G. Ligeti's sound-mass music contains an exact (as mathematical) approach to concurrency and parallelism appearing in performance according to musical scores for many voices.

Writing a musical score for many simultaneously appearing voices resembles writing a program (e.g. in NESL [4]) which respects simultaneously working processors, where e.g. the restrictions for sharing an access to central memory could correspond to counterpoint rules of polyphony.

The following quotation:

> *Ligeti's goal was apparently to entangle the voices to such a degree that they become imperceptible as individual entities ...*

together with remarks about randomization of beats in a bar according to K. Stockhausen from J. J. Iverson's Ph.D. thesis [11] suggests that

- Ligeti's sound-mass music could serve as a metaphor for quantum massive parallelism,
- randomized spiking neural P systems [9] (respecting the timing of spikes by counting time by beats in the bars) could be mathematical models for this metaphor, where membranes—neurons could correspond to voices.

References

1. Aerst, D., Broekaert, J., Gabora L., *A case for applying an abstract quantum formalism to cognition*, New Ideas in Psychology 29 (2011), pp. 136–146.
2. Arora, S., Barak, B., *Computational Complexity. A Modern Approach*, Cambridge Univ. Press, Cambridge, 2009.
3. Audebaud, P., Paulin-Mohring, C., *Proofs of randomized algorithms in Coq*, Sci. Comput. Programming 74 (2009), pp. 568–589.
4. Blelloch, G. E., *NESL: a Nested Data-Parallel Language* (Version 3.1), Technical Report CMU-CS-95-170, Carnegie-Mellon University, 1995.
5. Bunge, M., *Bayesianism: Science or Pseudoscience*, International Review of Victimology 15 (2008), pp. 165–178.
6. Fortnow, L., *One complexity theorist's view of quantum computing*, Theoret. Comput. Sci. 292 (2003), pp. 597–610.
7. Hofmeister, T., et al., *Randomized algorithms for 3-SAT*, Theory of Comput. Syst. 40 (2007), pp. 249–262.
8. Hurd, J., *Verification of the Miller–Rabin probabilistic primality tests*, J. Log. Algebr. Program. 56 (2003), pp. 3–21.
9. Ionescu, M., Păun, Gh., Yokomuri, T., *Spiking neural P systems*, Fund. Inform. 71 (2006), pp. 279–308.
10. Ipate, F., Turcanu, A., *Modeling, verification, and testing of P systems using Rodin and Prob*, in: 9th BWMC, Sevilla, January 31 – February 4, 2011, ed. Martinez-del-Amour, M. A. et al., Sevilla Univ., 2011, pp. 209–219.
11. Iverson, J. J., *Historical memory and György Ligeti's sound-mass music* 1958–1968, Ph.D. Dissertation, The University of Texas in Austin, 2009.
12. Iwama, K., et al., *Improved randomized algorithms for 3-SAT*, in: Algorithms and Computation, Part I, Lecture Notes in Comput. Sci. 6506, Springer, Berlin, 2010, pp. 73–84.
13. Lenstra, A. K., Lenstra, H. W., Jr. (eds.), *The Development of the Number Field Sieve*, Lecture Notes in Math. 1554, Springer, Berlin, 1993.

14. Obtułowicz, A., *Probabilistic P systems*, in: Membrane Computing, Lecture Notes in Comput. Sci. 2597, Springer, Berlin, 2003, pp. 377–387.
15. Obtułowicz, A., *Randomized Gandy–Păun–Rozenberg machines*, in: Membrane Computing (Jena, 2010), Lecture Notes in Comput. Sci. 6501, Springer, Berlin, 2011, pp. 305–324.
16. Păun, Gh., Rozenberg, G., Salomaa, A., *The Oxford Handbook of Membrane Computing*, Oxford, 2009.
17. Steane, A. M., *A quantum computer only needs one universe*, Studies in History and Philosophy of Physics 34 (2003), pp. 469–478.
18. Volchan, S. B., *What is a random sequence?*, Amer. Math. Monthly 109 (2002), pp. 46–63.
19. *The P-Lingua website*, `http://www.p-lingua.org`

Languages and P Systems: Recent Developments

Gheorghe Păun[1,2], Mario J. Pérez-Jiménez[2]

[1] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania

[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, marper@us.es

Summary. Languages appeared from the very beginning in membrane computing, by their length sets or directly as sets of strings. We briefly recall here this relationship, with some details about certain recent developments. In particular, we discuss the possibility to associate a control word with a computation in a P system. An improvement of a result concerning the control words of spiking neural P systems is given: regular languages can be obtained as control words of such systems with only four neurons (and with usual extended rules: no more spikes are produces than consumed). Several research topics are pointed out.

1 Introduction

Basically, membrane computing is associated with multiset processing in the compartments defined by a membrane structure, hence with handling *numbers* encoded in a *unary* manner, by means of the multiplicity of given objects, represented by symbols of an alphabet. However, from the very beginning, [22], also P systems were considered whose objects are strings. While the multisets of objects are processed by biochemical or biological inspired rules (similar to reactions taking place among the chemicals in a cell, or by other operations, such as symport and antiport), the string objects should be processed by specific rules, such as rewriting, splicing (from DNA computing), replication. However, also in the case of symbol objects we can "compute" (generate, accept or translate) strings and languages, and we find this case particularly interesting, taking into account the qualitative difference between the "internal data structure", the multiset, and the "external data structure", the string (hence with a positional information). That is why in what follows we only discuss this case, of symbol objects P systems handling languages.

For P systems with string objects we refer to the corresponding chapter of [31] and to the current bibliography of membrane computing from [38]. It is important

to note, however, that P systems with string objects can have interesting applications in natural language processing; we refer only to [1], but researches of the same group should be followed in this respect.

In what follows, we assume that the reader is familiar with basic facts in membrane computing, including definitions of the main classes of P systems: cell-like P systems with symbol objects (called here *transition P systems*, P systems with active membranes, symport-antiport P systems, spiking neural P systems (in short, SN P systems). Details can be found in [23], [31], and at [38]. We also assume some familiarity with basic elements of formal language theory, e.g., from [34]. Some notations will be also given below; we only mention now that REG, LIN, CF, CS, RE denote the families of regular, linear, context-free, context-sensitive, and recursively enumerable languages, respectively, and that V^* is the set of all strings over the alphabet V, the empty string, λ, included.

Informal presentations of the four classes of P systems are given below, in order to facilitate the understanding of the subsequent sections.

A *transition P system* uses rules of the form $u \to v$, where u and v are strings over a given alphabet O of objects, representing multisets; the intuition is that the objects in the multiset (represented by) u are consumed and those in v are produced, like in a (bio)chemical reaction. The objects in v can have associated *target indications*, in the forms $(a, here), (a, in), (a, out)$; the meaning is that the object a produced by applying the rule remains in the same compartment of the membrane structure if *here* is associated with it, it goes to a membrane immediately inside the compartment where the rule is used, or it goes outside this compartment, in the surrounding compartment, if the indications *in* or *out* are associated, respectively. Note that the objects are processed inside compartments, by local rules, but they can travel across membranes, due to the target indications. In particular, an object (a, out) produced in the external membrane of a P system (also called skin membrane) leaves the system and it "gets lost" in the environment.

Rules of the general form $u \to v$ are called *cooperative*. If u consists of a single object, then the rule is said to be *non-cooperative*. The intermediate case of rules $ca \to cv$, where a and c are objects, with c taken from a distinguished subset C of O, is the *catalytic* case.

In *P systems with active membranes*, the membranes themselves are part of rules and can evolve during a computation. The objects can evolve inside compartments (by cooperative, catalytic or non-cooperative rules) and can pass across membranes, while membranes can get divided, dissolved, separated, etc.

In P systems with symport-antiport rules the objects pass across membranes by rules of the forms $(u, in), (u, out)$ (symport rules), and $(u, out; v, in)$ (antiport rules), where u, v are strings in O^* (representing multisets of objects). The rules are associated with the membranes, the objects are never modified, they are just moved from a compartment to another one.

Starting from an initial configuration (the membrane membrane structure and the multisets placed in its compartments), and using the rules in a specified way

(synchronously or unsynchronously, in the maximally parallel way, sequentially, etc.), we get *transitions* among configurations; a sequence of transitions forms a *computation*; a computation which reaches a configuration where no rule can be applied is said to be *halting*. In all the previous cases, the most natural result of a computation is a number, for instance, of objects present in the halting configuration in a specified membrane.

In what follows, always the P systems work in the maximally parallel manner.

Finally, *an SN P system* consists of a set of *neurons* placed in the nodes of a directed graph and sending signals (*spikes*, denoted in what follows by the symbol a) along *synapses* (arcs of the graph). The objects evolve by means of *spiking rules*, which are of the form $E/a^c \to a; d$, where E is a regular expression over $\{a\}$ and c, d are natural numbers, $c \geq 1, d \geq 0$. The meaning is that a neuron containing k spikes such that $a^k \in L(E), k \geq c$, can consume c spikes and produce one spike, after a delay of d steps. This spike is sent to all neurons to which a synapse exists outgoing from the neuron where the rule was applied. There also are *forgetting rules*, of the form $a^s \to \lambda$, with the meaning that $s \geq 1$ spikes are forgotten, provided that the neuron contains exactly s spikes. If rules can produce more than one spike, i.e., they are of the form $E/a^c \to a^p; d$, with E, c, d as above and $1 \leq p \leq c$, then the system is said to be *extended*. (Note that the number p of produced spikes cannot be greater than the number c of consumed spikes.) In the initial configuration, each neuron contains a given number (it can be zero) of spikes.

The system works in a synchronized manner, i.e., in each time unit, the rule to be applied in each neuron is non-deterministically chosen, each neuron which can use a rule should do it, but the work of the system is sequential in each neuron: only (at most) one rule is used in each neuron. One of the neurons is considered to be the *output neuron*, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* of the system – it might be infinite if the computation does not stop. The *result of a computation* can be the spike train itself (a binary string if the computation halts, or an infinite sequence otherwise) or a number (e.g., the distance between the first two spikes sent into the environment by the output neuron of the system).

If a spiking rule $E/a^c \to a * p$ has $L(E) = a^c$, then we write it in the simpler form $a^c \to a^p$ (and we call it *finite*).

Four ways to associate a language with a P system were considered so far:

1. external output,
2. using a P system in the accepting mode,
3. following the trace of a distinguished object through the membrane structure,
4. control words.

We shortly present them below, with some details in the case of control words, and then we propose some ideas for further research.

It should be noted that the references we give here are not meant to be complete or to indicate the first place where a notion was introduced, but only to offer a good introduction to this research area.

A *general research topic* can already be formulated here: consider systematically the 4×4 combinations of (basic) types of P systems and ways to associate a language with a P system. Not all of these 16 possibilities were explored (but we cannot say in advance that any of them is of no interest). In particular, equivalences between some of the 16 combinations would be nice to be found.

2 External Output

Introduced already in [30], for transition P systems, the idea is simple: because objects can exit a P system (of any type), we (the user, the observer) can "wait in the environment" and arrange the symbols which leave the system in a sequence. If the computation halts, then we obtain a string, if not, we obtain an infinite sequence. An important detail: we have to decide what to do in the case when several objects leave the system at the same time. In [30] and several subsequent papers, all permutations of the symbols are allowed, hence several strings are associated with the same computation. An interesting possibility is to disregard certain symbols and/or to associate a single symbol to a multiset (by means of a given "interpretation mapping"), like in [12].

Somewhat surprisingly, in spite of its simple definition, defining a language in the external output manner was not too much investigated – at least not until last years, when a systematic study was started in [3], [4], mainly for transition P systems with non-cooperative rules (and no further ingredients; in [30], catalytic rules and membrane dissolution rules are used, as well a priority relation among them). The obtained family lies in between REG and CS and has interesting (combinatorial) properties.

The spike train of an SN P system can also be considered as the result of a computation defined in the external mode, but, having only one object, we have to assign different symbols to the time units when (at least) a spike exits the system and to the time units when no spike is emitted. In this way, a binary string (or sequence, when the computation does not stop) is obtained. There are several papers in the SN P systems area dealing with such languages.

The external output is not very much investigated for symport-antiport systems, and we know no paper of this kind dealing with P systems with active membranes. Also, as far as we know, the case when only computations which send out at most one object in each step was not investigated (this condition imposes a restriction on the accepted computations, hence the computing power of P systems can be altered in this way).

3 P Automata

This is indeed a much investigated topic in membrane computing – but mainly for the symport-antiport case. The idea is simple (symmetric to the external output): we arrange in a sequence the symbols which enter a P system, again with the two possibilities, to consider all permutations of symbols which enter in the same step (see [21] and its bibliography), or to consider an encoding of multisets by symbols (see a survey and references in [12]).

For symport-antiport P systems the "reading" of symbols from the environment is naturally defined by means of symport and antiport rules associated with the skin membrane. This is also provided by rules with active membranes, but we know no study about this issue for such systems. For transition P systems and for SN P systems we have to input symbols in an "external manner" (an external user provides a string, symbol by symbol, according to its wish). In most cases, a string is accepted if the computation halts (there are also other ways to define successful computations, such as local halting, reaching final configurations, but we do not discuss them here).

This way of using P systems is also related to the use of P systems to solve decidability problems, were an input is introduced in the system and the problem (an instance of it) has a positive answer if the computation halts (and a special object is sent to the environment), however, in this case the input (an encoding of the instance of the problem) is introduced in the form of a multiset, placed in a distinguished membrane. Details can be found in [32].

A recent variant of P automata was introduced in [26], called *dP automata*: several (symport-antiport) systems are connected to each other by means of antiport-like rules; they read separately strings from the environment, process them, also communicating, and if the computation halts, then the concatenation of the input strings is accepted. This is a way to introduce more distribution in P systems, making explicit the splitting of a problem among the components of the dP automaton. There are several papers devoted to this topic, see, e.g., [27], [28], [37]. The idea was extended also to SN P systems, in [19]; in this context, also a dual of spiking rules is introduced, in the form of *request rules* (depending on the contents of a neuron, spikes can be brought in from the environment, that is, the spikes come in *by request*, not introduced by an external user).

4 Traces

The idea was introduced in [18] for symport-antiport P systems, investigated in a couple of papers (see [17] and its bibliography), and extended to SN P systems in [7]: distinguish an object and follow its path across membranes; the sequence of membrane labels visited by that object provides a string (in the case of SN P systems, one single spike is distinguished, it it always used by a spiking rule applied in the neuron where the marked spike resides and one of the produced

spikes become marked). We know no paper dealing with traces in transition P systems and in P systems with active membranes.

5 Control Words

Finally, the fourth way to associate a string with a computation is to consider *control words*, as sequences of labels of rules used in the steps of a computation.

This is a well investigated topic in formal language theory, especially for Chomsky grammars, because in each step such grammars use only one rule. Each derivation produces a control word; the set of all control words associated with all terminal derivations in a grammar is called the *Szilard* language associated with (generated by) the grammar. The things become more complicated in the case of parallel computing devices, when several rules are used simultaneously.

This is the case also in membrane computing, and probably this is the reason why control words were, up to our knowledge, never considered in this area (until the special case proposed in [33]). However, a sort of bidimensional control word was introduced already in [10], under the name of *Sevilla carpet*, as a way to describe the rules used in a computation and their multiplicity in each step, but not as a way to define a control language associated with the computations in a P system.

A possible solution to the above difficulty is to consider a sequence of multisets of labels, those labels associated with all rules applied in a given step. Then, a string of symbols can be obtained following the ideas also used for accepting P systems: take a function from multisets to strings and build the string(s) obtained by concatenating the strings associated with the multisets. For instance, all permutations of the labels in a multiset can be considered, as in [21], or only one specific string (maybe a symbol) associated with the multiset, like in [12].

Another idea was recently introduced in [33], starting from the following restriction: all rules used in a computation step should have the same label, or they can also be labeled with λ.

The definition in [33] is given for SN P systems, but it works for any type of P systems.

Indeed, let us consider a P system Π, of any type, with the total set of rules (the union of all sets of rules associated with compartments, membranes, neurons – as it is the case) denoted with R. Consider a labeling mapping $l : R \to B \cup \{\lambda\}$, where B is an alphabet. We consider only transitions $s \Longrightarrow_b s'$, between configurations s, s' of Π, which use only rules with the same label b and rules labeled with λ. We say that such a transition is *label restricted*. With a label restricted transition we associate the symbol b if at least one rule with label b is used; if all used rules have the label λ, then we associate λ to this transition. Thus, with any computation in Π starting from the initial configuration and proceeding through label restricted transitions we associate a (control) word. The language of control words associated with all label restricted halting computations in Π is denoted

by $Sz_\lambda(\Pi)$. The subscript indicates the fact that λ steps are permitted; in the opposite case, we write $Sz(\Pi)$ (the label restricted transitions which cannot use only rules with label λ are called λ-*label restricted*).

We give here two results for symport-antiport P systems. The family of languages $Sz(\Pi)$ associated with symport-antiport P systems with at most m membranes is denoted with $SzSAP_m$; when λ moves are allowed, we write $Sz_\lambda SAP_m$, and if the number of membranes is not bounded, then the subscript m is replaced with $*$.

In what follows we need the characterizations of regular languages by means of *regular grammars*. Such a device is a construct $G = (N, T, S, P)$, where N, T are disjoint alphabets (the nonterminal and the terminal one, respectively), $S \in N$ (the axiom), and P is a finite set of rewriting rules of the forms $A \to aB, A \to a$, where $A, B \in N$ and $a \in T$; a rule $S \to \lambda$ can be added, if we also want to generated the empty word. The language generated by G is denoted with $L(G)$. Without any loss of generality we may assume that the grammar is *reduced*: each $A \in N$ can be reached from S and can derive a terminal string.

When comparing two language generating or accepting devices G_1, G_2, the empty string is ignored, that is, $L(G_1)$ is considered equal to $L(G_2)$ as soon as $L(G_1) - \{\lambda\} = L(G_2) - \{\lambda\}$. Thus, no λ-rule is necessary in our regular grammars.

Theorem 1. $REG \subset SzSAP_1$.

Proof. The inclusion is easy to prove: for a regular grammar $G = (N, T, S, P)$ with $N = \{A_1 = S, A_2, \ldots, A_n\}$, we consider the antiport rules $b : (A_i, out; A_j, in)$ associated with $A_i \to bA_j \in P$ and the symport rules $b : (A_i, out)$ associated with $A_i \to b \in P$. Initially, the single membrane of the system contains the object A_1. Clearly, each terminal derivation in G corresponds to a halting computation in the system we have constructed, and conversely.

The inclusion is strict; actually, we have a stronger result: $SzSAP_1 - CF \neq \emptyset$. A P system proving this assertion is

$$\begin{aligned}
\Pi &= (O, [\ \ ]_1, e, O, R_1), \text{ where:}\\
O &= \{a_1, a_2, e, f, g, h\},\\
R_1 &= \{a : (e, out; ea_1a_2, in),\ a : (e, out; fa_1a_2, in),\\
&\quad b : (fa_1, out; f, in),\ b : (fa_1, out; g, in),\\
&\quad c : (ga_2, out; g, in),\ c : (ga_1, out; h, in),\\
&\quad d : (ha_1, out; ha_1, in),\ d : (ha_2, out; ha_2, in)\}.
\end{aligned}$$

The "carrier" e bring inside $n \geq 1$ copies of a_1 and a_2, then f and g remove copies of a_1 and a_2, respectively. Eventually, the object h is introduced in the system. If any copy of a_1 or a_2 is still present in the system, then the computation never halts, because the rules with label d can be used forever. Therefore, the control words associated with terminal computations are of the form $a^n b^n c^n$, for some $n \geq 1$, hence $Sz(\Pi)$ is not context-free. □

If steps when only rules with label λ are allowed, then all one-letter recursively enumerable languages can be generated.

Theorem 2. *If $L \subseteq a^*, L \in RE$, then $L \in Sz_\lambda SAP_1$.*

Proof. A language $L \in a^*$ is in RE if and only if its length set is a recursively enumerable set of numbers. Symport-antiport P systems with one membrane (and rules with no restricted complexity) can generate all recursively enumerable sets of numbers, [31]. Take such a system Π, namely, one which simulates a register machine $M = (n, H, l_0, l_h, I)$ (the number of registers, the set of instruction labels, the label of the initial instruction, the label of the halt instruction, the set of instructions, labeled with elements of H; simulating register machines is the usual way to prove the universality of symport-antiport P systems, so the reader is assumed to be familiar with such proofs). In the halting configuration, the system contains k copies of a symbol a_1, which encodes the contents of register 1 of M, the one where the number is generated, as well as the object l_h, for $k \in N(M)$. Assume that all rules of Π are labeled with λ, and add the following rules $a : (l_h a_1, out; l_h, in)$. This rule must be used for each copy of a_1 present in the system, hence the control word of the computation in the augmented system – let us denote it by Π' – is a^k. The halting label l_h is introduced only in the last step of a computation in Π. Consequently, $L = Sz_\lambda(\Pi')$. □

In the previous results we have imposed no restriction on the length of the symport and antiport rules; if such restriction are considered, then a larger number of membranes is expected to be necessary.

The control words associated with transition P systems and with systems with active membranes remain to be investigated. In what follows we consider the case of SN P systems.

6 Control Words for SN P Systems

The fact that λ steps increase the power of systems is also confirmed for the control words associated with SN P systems, a case which is investigated in [33]. Let $SzSNP_m$, $Sz_\lambda SNP_m$ be the families of all languages $Sz(\Pi)$, $Sz_\lambda(\Pi)$, respectively, associated with SN P systems Π (with extended rules) with at most m neurons; if the number of neurons is not restricted, then we replace the subscript m by $*$. In [33] it is proved that $Sz_\lambda SNP_* = RE$, but $SzSNP_* \subset CS$, strict inclusion (an example of a language not in $SzSNP_*$ is the linear language $\{xx^R \mid x \in V^*\}$, where V is an alphabet with at least two symbols and x^R is the reversal/mirror image of the string x). Moreover, a theorem in given in [33] stating that each regular languages L is the λ-label restricted Szilard language of an SN P system Π – with the mentioning that the system Π uses extended rules of the form $E/a^c \to a^p$ without the restriction $p \leq c$ and it has arbitrarily many neurons. This result will be improved in the next theorem.

We give first an example, also improving a result from [33], where it is shown that $SzSNP_6$ contains non-context-free languages. We prove that four neurons suffice.

Consider the SN P system (with four neurons, $\sigma_1, \sigma_2, \sigma_3, \sigma_4$)

$$\begin{aligned}
\Pi &= (\{a\}, \sigma_1, \sigma_2, \sigma_3, \sigma_4, syn), \text{ where:}\\
\sigma_1 &= \sigma_2 = (2, \{r_1 : a^2 \to a^2,\ r_2 : a^2 \to a\}),\\
\sigma_3 &= (1, \{r_2 : (a^4)^+ a/a \to a,\ r_3 : (a^4)^+ a^2/a^4 \to a\},\\
\sigma_4 &= (1, \{r_2 : (a^4)^+ a/a \to a,\ r_4 : (a^4)^+ a^2/a^4 \to a\},\\
syn &= \{(1,2), (2,1), (1,3), (1,4), (2,3), (2,4)\}.
\end{aligned}$$

The system is given in a graphical form in Figure 1. Each neuron contains initially one or two spikes, but only σ_1 and σ_2 can fire. If the rules r_2 are used in σ_1 and σ_2 (not also in σ_3, σ_4, because we do not have here enough spikes), then the computation halts. Let us assume that for a number n of steps we use the rule r_1 in σ_1 and σ_2. Neurons 1 and 2 exchange spikes to each other and, together, they send four spikes to each of σ_3, σ_4. These neurons cannot use the rules r_3, r_4 until getting inside an even number of spikes, and this means that the rules r_2 in σ_3, σ_4 were used. This however supposes that also σ_1, σ_2 use the rules r_2 (these rules are applicable, hence they must be applied), and this ends the work of these neurons. After using the rules r_2, neurons 3 and 4 can fire nondeterministically, but not both at the same time: they have to use the rules r_3 and r_4, which have different labels. After using the rules r_2, each of σ_3 and σ_4 contains the same number of spikes, namely $4n + 2$, hence, besides the string r_2, $Sz(\Pi)$ contains strings of the form $r_1^n r_2 w$, with $w \in \{r_3, r_4\}^*$ containing the same number of r_3 and r_4. This language is not context-free, hence $SzSNP_4 - CF \neq \emptyset$.

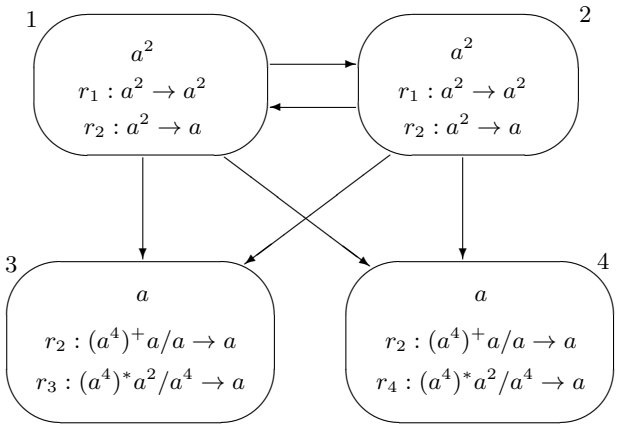


Fig. 1. An SN P system whose Szilard language is not context-free.

We give now the improvement of the mentioned result from [33].

Theorem 3. $REG \subset SzSNP_4$.

Proof. In view of the previous example, it is enough to prove the inclusion $REG \subseteq SzSNP$. To this aim, let us consider a regular language L generated by a reduced regular grammar $G = (N, T, S, P)$ with $N = \{S = A_1, A_2, \ldots, A_n\}$ and the rules in P of the forms $A_i \to bA_j$, $A_i \to b$, for some $A_i, A_j \in N$ and $b \in T$. Let us denote $J = \{1, 2, \ldots, n\}$.

We construct the following SN P system of degree 4 (together with the rules we also specify their labels):

$$\begin{aligned}
\Pi &= (\{a\}, (a^{2n+1}, R_{12}), (0, R_{12}), (a^{2n+1}, R_{34}), (0, R_{34}), syn),\\
R_{12} &= \{b : a^{2n+i} \to a^j \mid A_i \to bA_j \in R,\ i, j \in J\}\\
&\quad \cup \{b : a^{2n+i} \to a^{2n} \mid A_i \to b \in R,\ i \in J\},\\
R_{34} &= \{a^{2n+k} \to a^{2n} \mid k \in J\},\\
syn &= \{(1,2), (2,1), (1,4), (4,1), (2,3), (3,2), (3,4), (4,3)\}.
\end{aligned}$$

The system is also given in a graphical form in Figure 2. Note that it is finite and uses no forgetting rule.

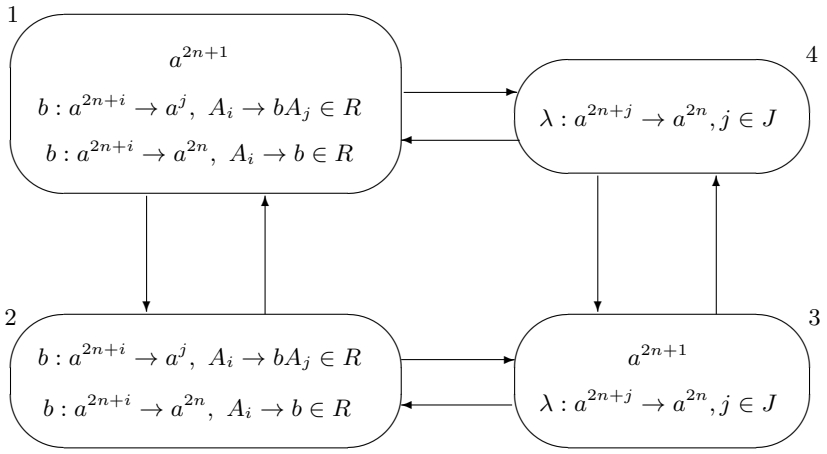


Fig. 2. An SN P system whose control language is a given regular language

In the first step, neurons 1 and 3 can fire; in the next step, neurons 2 and 4 fire – and the computation proceeds in steps which alternate the previous pairs of neurons. When a pair of neurons fires, then no spike remains inside these neurons, but the other pair receives spikes. This means that in each step a rule with a label $b \in T$ and one with the label λ are used (hence the computation is λ-label restricted).

With each nonterminal $A_i, 1 \leq i \leq n$, we have associated $2n+i$ spikes; initially, we have in neurons 1 and 3 spikes which identify the nonterminal $S = A_1$.

Assume that either σ_1, σ_3, or σ_2, σ_4 contain spikes, namely $2n+i$ in σ_1, σ_2, for a rule $A_i \to bA_j \in R$, and $2n+k$ in σ_3, σ_4, for some $k \in J$. The rule $A_i \to bA_j$ is simulated by using the rule (with label b) $a^{2n+i} \to a^j$ in σ_1, σ_2, simultaneously with using the rule (with label λ) $a^{2n+k} \to a^{2n}$ in σ_3, σ_4. The symbol b is added to the control word, and the process is continued with the simulation of a rule $A_j \to u \in R, u \in T \cup TN$.

In the moment when a (terminal) rule $A_i \to b \in R$ is simulated, the active σ_1 or σ_2 introduces $2n$ spikes, at the same time with $2n$ spikes produced by the paired neuron σ_3, σ_4. Two neurons are empty, the other two contains $4n$ spikes, hence no rule can be applied in any neuron. The computation halts, having as its control word the word generated by the derivation in G. Consequently, $Sz(\Pi) = L(G)$, which concludes the proof. □

We do not know whether the number of neurons in the previous theorem can be decreased.

7 Controlled P Systems

In the previous sections, the control words were collected in order to have a new way of producing a language starting from a P system. The computations cannot proceed freely, but they should be label restricted or λ-label restricted. This restriction has an influence on the computing power of P systems, considered as number computing devices. Indeed, let us consider the following systems:

$$\Pi_1 = (\{a, b\}, [\ ]_1, a, \{r_1 : a \to aa,\ r_2 : a \to b\}, 1),$$
$$\Pi_2 = (\{a, b\}, [\ ]_1, a, \{a, b\}, \{r_1 : (a, out; aa, in),\ r_2 : (a, out; b, in)\}, 1).$$

When only label restricted transitions are allowed, the two rules of each system cannot be used at the same time, hence we obtain $N_{lr}(\Pi_1) = N_{lr}(\Pi_2) = \{2^n \mid n \geq 0\}$ (we have added the subscript lr in order to indicate that only label restricted computations are allowed). This set of numbers cannot be generated by non-cooperative transition P systems, neither by symport-antiport P systems of this complexity (one membrane, two rules) with non-restricted computations.

A more general case is the one when a pair (Π, C) is considered, where Π is a P system of any type, with the rules labeled by elements of an alphabet H and $C \subseteq H^*$ is a given language. This language is used in order to restrict the computations in Π: only label restricted computations are allowed whose control words are in C. (This corresponds to controlled context-free grammars in regulated rewriting.)

The study of controlled P systems remains to be done (combining classes of P systems with types of control languages, as already done for Chomsky controlled

grammars). It is expected that a control language provides a powerful way to "program" the work of a P system.

8 Final Remarks

Many research topics were mentioned in the previous sections, many others remain to be explored. For instance, we have said nothing about tissue-like P systems – is anything interesting in this case from the language computing point of view? How this case compares with the four types of P systems considered above?

Another direction of investigation concerns sets of infinite sequences (also called ω-languages). Some results were reported in [15] for symport-antiport P systems, and in [29] and [14] for SN P systems.

A related issue was considered in [35]: handling languages over infinite alphabets.

Besides the previous ways to associate a language with a P system, also are other ideas were preliminarily explored. One of them is to encode a string in the membrane structure itself, and then handling the membrane structure means processing the string; see [5] for some details.

For all families of languages which are not equal to RE it makes sense to consider the classic problems investigated in formal language theory: closure properties, decidability, representation theorems, semilinearity, and so on. Also, the membership complexity is of interest (an issue considered already in [2]). In view of possible applications in modeling aspects related to natural languages, it would be of interest to find ways to generate *mildly context-sensitive languages* (semilinear, parsable in polynomial time, powerful enough to cover some non-context-free constructions in natural languages).

A related research direction concerns the translation of languages. Some attempts were reported in [11] and [25].

We can conclude with the observation that many things were done in membrane computing in handling languages by means of P systems with symbol objects, but a lot of work still remains to be carried out

Acknowledgements. Work supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. A. Alhazov, E. Boian, S. Cojocaru, Yu. Rogozhin: Modelling inflections in Romanian language by P systems with string replication. *Pre-proc. Tenth Workshop on Membrane Computing, WMC10*, Curtea de Argeş, August 2009 (M.J. Pérez-Jiménez, A. Riscos-Núñez, eds.), 116–128.
2. A. Alhazov, C. Ciubotaru, S. Ivanov, Yu. Rogozhin: Membrane systems languages are polynomial-time parsable. *Computer Science Journal of Moldova*, 18, 2 (2010), 139–148.

3. A. Alhazov, C. Ciubotaru, S. Ivanov, Y. Rogozhin: The family of languages generated by non-cooperative membrane systems. *Membrane Computing. 11th International Conference, CMC11, Jena, Germany, August 24-27, 2010. Revised, Selected, and Invited Papers* (M. Gheorghe et al., eds.), LNCS 6501, Springer, Berlin, 2010, 65–79.
4. A. Alhazov, C. Ciubotaru, Yu. Rogozhin, S. Ivanov: The membrane systems language class. *Proc. Eighth Brainstorming Week on Membrane Computing, Sevilla, 2010*, 23–35, and *Proc. LA Symposium*, RIMS Kôkyûroku Series 1691, Kyoto University, 2010, 44–50.
5. F. Bernardini, M. Gheorghe: Language generating by means of P systems with active membranes. *Proc. Brainstorming Week on Membrane Computing*, Technical Report, 26/03, Rovira i Virgili University, Tarragona, 2003, 46–60.
6. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75, 1-4 (2007), 141–162.
7. H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by spiking neural P systems. *Proc. Fourth Brainstorming Week on Membrane Computing*, Sevilla, 2006.
8. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In *Proc. Fourth Brainstorming Week on Membrane Computing*, Sevilla, 2006, RGNC Report 02/2006, 241–265.
9. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Handling languages with spiking neural P systems with extended rules. *Romanian J. Information Sci. and Technology*, 9, 3 (2006), 151–162.
10. G. Ciobanu, Gh. Păun, Gh. Ştefănescu: Sevilla carpets associated with P systems. *Proc. Brainstorming Week on Membrane Computing* (M. Cavaliere et al., eds.), Tarragona Univ., TR 26/03, 2003, 135–140.
11. G. Ciobanu, Gh. Păun, Gh. Ştefănescu: P transducers. *New Generation Computing*, 24, 1 (2006), 1–28.
12. E. Csuhaj-Varjú, Gy. Vaszil: P automata or purely communicating accepting P systems. *Membrane Computing, International Workshop, WMC-CdeA, Curtea de Argeş, Romania, August 19-23, 2002, Revised Papers* (Gh. Păun et al., eds.), LNCS 2597, Springer, 2003, 219–233.
13. R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez: On the power of P and dP automata. *Ann. Univ. Buc. Mathem.-Informatics Series*, 63 (2009), 5–22.
14. R. Freund, M. Oswald. Regular omega-languages defined by finite extended spiking neural P systems. *Fundamenta Informaticae*, 83 (2008), 65-73.
15. R. Freund, M. Oswald, L. Staiger. Omega-P automata with communication rules. *Pre-proc. Workshop on Membrane Computing, Tarragona, July 2003* (A. Alhazov et al., eds.), 252–265.
16. O.H. Ibarra, Gh. Păun: Characterizations of context-sensitive languages and other language classes in terms of symport/antiport P systems. *Theoretical Computer Sci.*, 358, 1 (2006), 88–103.
17. M. Ionescu: *Membrane Computing. Traces, Neural Inspired Models, Controls.* PhD Thesis, URV Tarragona, 2008.
18. M. Ionescu, C. Martin-Vide, Gh. Păun: P systems with symport/antiport rules: The traces of objects. *Grammars*, 5 (2002), 65–79.
19. M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez, T. Yokomori: Spiking neural dP systems. *Fundamenta Informaticae*, 11, 4 (2011), 423–436.

20. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71 (2006), 279–308.
21. M. Oswald: *P Automata*, PhD Thesis, TU Viena, 2003.
22. Gh. Păun: Computing with membranes. *J. Comput. Syst. Sci.*, 61 (2000), 108–143 (see also TUCS Report 208, November 1998 (`www.tucs.fi`).
23. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
24. Gh. Păun: Languages in membrane computing. Some details for spiking neural P systems. *Proc. 10th DLT Conf., Santa Barbara, USA, 2006*, LNCS 4036, Springer, Berlin, 2006, 20–35.
25. Gh. Păun: Spiking neural P systems used as acceptors and transducers. *Proc. CIAA 2007, 12th Conf.*, Prague, July 2007, LNCS 4783 (J. Holub, J. Zdarek, eds.), Springer, Berlin, 2007, 1–4.
26. Gh. Păun, M.J. Pérez-Jiménez: Solving problems in a distributed way in membrane computing: dP systems, *Int. J. of Computers, Communication and Control*, 5, 2 (2010), 238–252.
27. Gh. Păun, M.J. Pérez-Jiménez: P and dP automata: A survey. *Rainbow of Computer Science* (C.S. Calude, G. Rozenberg, A. Salomaa, eds.), LNCS 6570, Springer, Berlin, 2011, 102–115.
28. Gh. Păun, M.J. Pérez-Jiménez: An infinite hierarchy of languages defined by dP systems. *Theoretical Computer Sci.*, 431 (2012), 4–12.
29. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002.
30. Gh. Păun, G. Rozenberg, A. Salomaa: Membrane computing with an external output. *Fundamenta Informaticae*, 41, 3 (2000), 313–340
31. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
32. M.J. Pérez–Jiménez: A computational complexity theory in membrane computing. *Membrane Computing, Tenth International Workshop, WMC 2009, Curtea de Argeş Romania, August 2009, Selected and Invited Papers* (Gh. Păun et al., eds.), LNCS 5937, Springer, Berlin, 2009, 125–148.
33. A. Ramanujan, K. Krithivasan: Control words of spiking neural P systems. Paper in preparation, 2012.
34. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. 3 volumes, Springer, Berlin, 1998.
35. G. Vaszil: On a class of P automata as a machine model for languages over infinite alphabets. *Proc. Third Brainstorming Week on Membrane Computing, Sevilla, 2005* (M.A. Gutiérez-Naranjo et al., eds.), 317–325.
36. G. Vaszil: A class of P automata for characterizing context-free languages. *Proc. Fourth Brainstorming Week on Membrane Computing, Sevilla, 2006*, vol. II (C. Graciani et al., eds.), 267–276.
37. G. Vaszil: Variants of distributed P automata and the efficient parallelizability of langauges. *Membrane Computing. 12th Intern. Conf., CMC 2011, Fontainebleau, France, August 2011, Revised Selected Papers* (M. Gheorghe et al., eds.), LNCS 7184, Springer, Berlin, 2012, 51–61.
38. The P Systems Website: `http://ppage.psystems.eu`.

Image Thresholding with Cell-like P Systems

Hong Peng[1,3], Jie Shao[1], Bing Li[1], Jun Wang[2], Mario J. Pérez-Jiménez[3], Yang Jiang[1], Yufan Yang[1]

[1] School of Mathematics and Computer Engineering, Xihua University, Chengdu, Sichuan, 610039, China
[2] School of Electrical and Information Engineering, Xihua University, Chengdu, Sichuan, 610039, China
[3] Research Group of Natural Computing, Department of Computer Science and Artificial Intelligence, University of Seville, Sevilla, 41012, Spain
ph.xhu@hotmail.com

Summary. P systems are a new class of distributed parallel computing models. In this paper, a novel three-level thresholding approach for image segmentation based on cell-like P systems is proposed in order to improve the computational efficiency of multi-level thresholding. A cell-like P system with a specially designed membrane structure is developed and an improved evolution mechanism is integrated into the cell-like P system. Due to parallel computing ability and particular mechanism of the cell-like P system, the presented thresholding approach can effectively search the optimal thresholds for three-level thresholding based on total fuzzy entropy. Experimental results of both qualitative and quantitative comparisons for the proposed approach and GA-based and PSO-based approaches illustrate the applicability and effectiveness.

Key words: Image segmentation, Thresholding approach, Membrane computing, Cell-like P systems, Total fuzzy entropy

1 Introduction

Membrane computing, as a new branch of natural computing, was proposed by Păun [1] in 2000. Membrane computing is a novel class of distributed parallel computing models, which is inspired by the structure and functioning of living cells, as well as from the way the cells are organized in tissues or higher order structure [2]. The computing models are commonly called P systems. Since then, a large number of P systems and their variants have been proposed [3, 4, 5, 6, 7, 8, 9, 10]. The main ingredients of a P system are (i) *the membrane structure*, delimiting compartments where (ii) *multisets of objects* evolve according to (iii) *(reaction) rules* of a bio-chemical inspiration. According to their structures, these models can be divided into three categories: cell-like P systems, tissue-like P systems and

neural-like P systems. The investigation of P systems mainly focuses on building a variety of computing models for different classes of problems, such as computing power, computational efficiency, and so on. The application research of P systems, especially applying P systems to solve real-world problems, has been concerned about in recent years. Among them, membrane algorithms are a class of representative models, which have been successfully used to deal with optimization problems [11, 12], control problems [13] and signal processing [14]. This paper focuses on application of P systems to image segmentation problem.

Image segmentation is a process of grouping an image into units that are homogeneous with respect to one or more characteristics. It is an important task in image analysis. Thresholding is widely used as a popular technique in image segmentation. The goal of thresholding is to separate objects from background image or discriminate objects from objects that have distinct gray levels. Over these years, A large number of thresholding techniques have been addressed [15, 16, 17]. Bi-level thresholding, which is firstly discussed, segments an image into two different regions. The pixels with gray values greater than a certain threshold are classified as object pixels, and the others with gray values lesser than the threshold are classified as background pixels. Otsu's approach [18] and Kapur's approach [19], which find the optimal thresholds by maximizing the between-class variance of gray levels and the entropy of the histogram respectively, are simple and effective in bi-level thresholding. However, the gray level histograms of most of the images in the real world is multimodal. Therefore, multi-level thresholding has been received many attentions in recent years. Multi-level thresholding determines more than one threshold for an image and segments the image into several distinct regions, which corresponds to one background and several object. The Otsu' and Kapur's approaches can be extendable to multi-level thresholding but inefficient in determining the optimal thresholds due to the exponential growth in computation time. To improve the efficiency, some approaches have been proposed to reduce the computational complexity of determining the multi-level thresholds, such as the recursive algorithm [20]. But they still suffer from long processing time when the number of thresholds increases. The fuzzy entropy has been introduced into image segmentation in recent years [21, 22, 23, 24]. Cheng et al. [21] proposed a thresholding approach, where the fuzzy relation and the maximum fuzzy entropy were used to perform fuzzy partition on a two-dimensional histogram. In [22], Shelokar et al. found the optimal threshold by minimizing the sum of the fuzzy entropies. Zhao et al. [23] presented a three-level thresholding approach based on fuzzy entropy. In [24], Liu et al. presented a fuzzy classification entropy to deal with multi-level thresholding. However, these approaches still suffer from the same problem mentioned above. In order to overcome this problem, some intelligent computing approaches have been applied to solve multi-level thresholding problems, such as genetic algorithm (GA), particle swarm optimization (PSO) and ant colony optimization (ACO). Yin et al. [25] presented a GA-based thresholding approach, where the objective function was similar to Otsu's or Kapur's function. In [26], Cheng et al. defined an approach to fuzzy entropy and employed the GA to

find the optimal combination of the fuzzy parameters. Tao et al. [27] presented a three-level thresholding approach that uses the GA to find the optimal thresholds by maximizing the fuzzy entropy. In [28], Hammouche et al. proposed a multi-level thresholding approach, which allows the determination of the appropriate number of threshols as well as the adequate threshold values. However, GA has some drawbacks such as slow convergence rate, premature convergence to local minima. Thus, the PSO has been applied to multi-level thresholding [29, 30, 31]. In addition, Tao et al. [32] used the ACO to obtain the optimal parameters of the presented entropy-based object segmentation approach. Currently, adapting P systems to solve image segmentation problems has been addressed [33, 34]. Díaz-Pernil et al. [33] combined the membrane structure and symport-antiport communication rules of tissue-like P systems to deal with homology groups of binary 2D image. Wang et al. [34] presented a bi-level image thresholding approach.

In this paper, we propose a novel three-level thresholding approach based on cell-like P systems for image segmentation. Our main motivation is to improve and enhance the efficiency of multi-level thresholding approach based on the fuzzy entropy criterion by applying the parallel computing ability as well as specially designed structure and mechanisms of cell-like P systems. The proposed three-level thresholding approach is evaluated on several standard images and compared with the GA-based and PSO-based approaches.

The rest of this paper is organized as follows. Section 2 briefly describes the maximum fuzzy entropy principle. The proposed three-level thresholding approach based on cell-like P systems is presented in Section 3. Experimental results are provided in Section 4. Finally, Section 5 draws the conclusions.

2 Maximum Fuzzy Entropy Principle

In this section, we briefly review maximum fuzzy entropy principle and give fuzzy membership functions used in this paper.

Let $D = \{(i, j) \mid i = 0, 1, \ldots, M-1; j = 0, 1, \ldots, N-1\}$, $G = \{0, 1, \ldots, l-1\}$, where M, N and l are three positive integers. Let $I(x, y)$ be the gray level of an image I at the pixel (x, y). Denote

$$D_k = \{(x, y) \mid I(x, y) = k,\ (x, y) \in D\},\ k = 0, 1, 2, \ldots, l-1 \tag{1}$$

$$h_k = \frac{n_k}{M \times N} \tag{2}$$

where n_k is the number of pixels in D_k. So, $0 \leq h_k \leq 1$, $\sum_{n=0}^{l-1} h_k = 1$. Let $H = \{h_0, h_1, \ldots, h_{l-1}\}$ be the gray histogram of the image I. $\{D_o, D_1, \ldots, D_{l-1}\}$ forms a probability partition of D and its probabilistic distribution is $p_k = P(D_k) = h_k$ $(k = 0, 1, \ldots, l-1)$.

In this paper, we will deal with three-level image thresholding, which has two thresholds, t_1 and t_2. The two thresholds will segment the image I into three gray levels, low gray level, middle gray level and high gray level, and the corresponding

domains are denoted by D_l, D_m and D_h respectively. Thus, $D = D_l \bigcup D_m \bigcup D_h$, $D_l \bigcap D_m = D_l \bigcap D_h = D_m \bigcap D_h = \phi$. Let p_l, p_m and p_h be the probabilistic distributions of D_l, D_m and D_h respectively, i.e., $p_l = P(D_l)$, $p_m = P(D_m)$, $p_h = P(D_h)$. However, these probabilistic distributions are unknown.

For $k = 0, 1, \ldots, 255$, denote

$$\begin{aligned} D_{kl} &= \{(x,y) \mid I(x,y) \le t_1,\ (x,y) \in D_k\} \\ D_{km} &= \{(x,y) \mid t_1 < I(x,y) \le t_2,\ (x,y) \in D_k\} \\ D_{kh} &= \{(x,y) \mid I(x,y) > t_2,\ (x,y) \in D_k\} \end{aligned} \tag{3}$$

Then, we have

$$\begin{aligned} p_{kl} &= P(D_{kl}) = p_k \times p_{l|k} \\ p_{km} &= P(D_{km}) = p_k \times p_{m|k} \\ p_{kh} &= P(D_{kh}) = p_k \times p_{h|k} \end{aligned} \tag{4}$$

with a constraint that $p_{l|k} + p_{m|k} + p_{h|k} = 1$ $(k = 0, 1, \ldots, 255)$. Thus, $p_l = \sum_{k=0}^{255} p_k \times p_{l|k}$, $p_m = \sum_{k=0}^{255} p_k \times p_{m|k}$ and $p_h = \sum_{k=0}^{255} p_k \times p_{h|k}$.

Let $\mu_l(k)$, $\mu_m(k)$ and $\mu_h(k)$ denote the membership grades of a pixel belonging to D_l, D_m and D_h respectively. Then

$$p_l = \sum_{k=0}^{255} p_k \times \mu_l(k),\ p_m = \sum_{k=0}^{255} p_k \times \mu_m(k),\ p_h = \sum_{k=0}^{255} p_k \times \mu_h(k) \tag{5}$$

In this paper, we employ the following three functions to approximate the membership functions $\mu_l(k)$, $\mu_m(k)$ and $\mu_h(k)$, respectively

$$\mu_l(k) = \begin{cases} 1, & k \le a \\ 1 - \frac{(k-a)^2}{(c-a)\times(b-a)}, & a < k \le b \\ \frac{(k-c)^2}{(c-a)\times(c-b)}, & b < k \le c \\ 0, & k > c \end{cases} \tag{6}$$

$$\mu_m(k) = \begin{cases} 0, & k \le a \\ \frac{(k-a)^2}{(c-a)\times(b-a)}, & a < k \le b \\ 1 - \frac{(k-c)^2}{(c-a)\times(c-b)}, & b < k < c \\ 1, & k = c \\ 1 - \frac{(k-c)^2}{(e-c)\times(d-c)}, & c < k \le d \\ \frac{(k-e)^2}{(e-c)\times(e-d)}, & d < k \le e \\ 0, & k > e \end{cases} \tag{7}$$

$$\mu_h(k) = \begin{cases} 0, & k \le c \\ \frac{(k-c)^2}{(e-c)\times(d-c)}, & c < k \le d \\ 1 - \frac{(k-e)^2}{(e-c)\times(e-d)}, & d < k \le e \\ 1, & k > e \end{cases} \tag{8}$$

where $0 < a \leq b \leq c \leq d \leq e < 255$.

The fuzzy entropies of above three classes are given as follows:

$$\begin{aligned} H_l &= -\sum_{k=0}^{255} \frac{p_k \times \mu_l(k)}{p_l} \times \ln\left(\frac{p_k \times \mu_l(k)}{p_l}\right) \\ H_m &= -\sum_{k=0}^{255} \frac{p_k \times \mu_m(k)}{p_m} \times \ln\left(\frac{p_k \times \mu_m(k)}{p_m}\right) \\ H_h &= -\sum_{k=0}^{255} \frac{p_k \times \mu_h(k)}{p_h} \times \ln\left(\frac{p_k \times \mu_h(k)}{p_h}\right) \end{aligned} \tag{9}$$

Then the total fuzzy entropy is computed by

$$H(a, b, c, d, e) = H_l + H_m + H_h \tag{10}$$

From the Eq.(10), we can see that H is the function of five parameters a, b, c, d, e in fact. The optimal image thresholding is to find the most appropriate combination of these parameters so that the total fuzzy entropy $H(a, b, c, d, e)$ achieves the maximum value. Then the most appropriate combination of these parameters, by which the image I is segmented into three classes, can satisfy the following relation:

$$\mu_l(t_1) = \mu_m(t_1) = 0.5, \ \mu_m(t_2) = \mu_h(t_2) = 0.5 \tag{11}$$

Note that threshold t_1 is the intersection point of curves $\mu_l(k)$ and $\mu_m(k)$, while threshold t_2 is the intersection point of curves $\mu_m(k)$ and $\mu_h(k)$. Therefore, according to Eqs.(6)-(8), the two thresholds can be determined as follows:

$$t_1 = \begin{cases} a + \sqrt{(c-a) \times (b-a)/2}, & (a+c)/2 \leq b \leq c \\ c - \sqrt{(c-a) \times (c-b)/2}, & a \leq b \leq (a+c)/2 \end{cases} \tag{12}$$

$$t_2 = \begin{cases} c + \sqrt{(e-c) \times (d-c)/2}, & (c+e)/2 \leq d \leq e \\ e - \sqrt{(e-c) \times (e-d)/2}, & c \leq d \leq (c+e)/2 \end{cases} \tag{13}$$

3 The Proposed Image Thresholding Approach

The proposed thresholding approach is based on a cell-like P system. In order to effectively deal with three-level thresholding problem under P systems, we design a special membrane structure with three layers, which consists of $(2m + 1)$ membranes, shown in Fig. 1. These membranes are labeled by $1, 2, \ldots, m, m + 1, m + 2, \ldots, 2m, 2m + 1$, respectively. The m membranes labeled by $1, 2, \ldots, m$, which are called evolution membranes, will cooperatively evolve the objects in the system to find the optimal segmentation thresholds. Each evolution membrane has one child membrane, called local store membrane, whose role is to store the best object

found as far in the evolution membrane. In Fig.1, membranes $m+1, m+2, \ldots, 2m$ are the local store membranes of evolution membranes $1, 2, \ldots, m$, respectively. In each computing step, if a evolution membrane find its new best object by evolution rule it will transmit the new best object into the corresponding local store membrane and skin membrane $(2m+1)$. The skin membrane is called global store membrane in this paper, whose role is to store best object found as far in entire system. In the beginning of computing step, each evolution membrane will receive local best object from the corresponding local store membrane as well as global best object from global store membrane (skin membrane). In Fig.1, the arrows with different directions indicate the transitive relations of objects. As usual in P systems, these evolution membranes as parallel computing units work in a maximally parallel way (a universal clock is considered here).

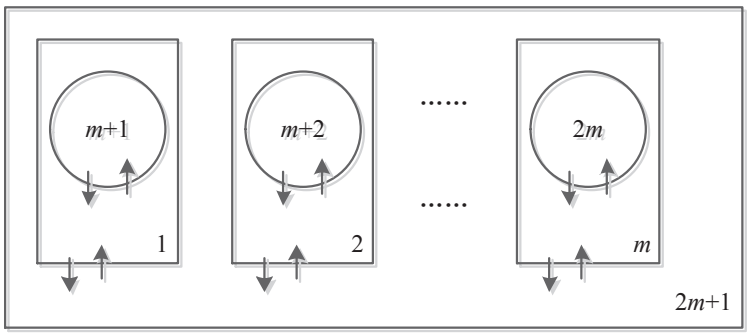


Fig. 1. Membrane structure of the used cell-like P system.

As we known, every membrane contains a certain number of objects. For simplicity, we assume that every evolution membrane contains same number of objects, and the number is denoted by n. However, local store membranes and global store membrane contain only one object respectively. In this work, each object is a five-dimensional vector $X = (x_1, x_2, x_3, x_4, x_5)$, where x_1, x_2, x_3, x_4 and x_5 correspond to five segmentation parameters, a, b, c, d, and e respectively. Therefore, each object in fact expresses a candidate of the optimal segmentation thresholds to be found. In the cell-like P system, the total fuzzy entropy (i.e., Eq. (10)) will be regarded as fitness function of object in the system to evaluate the quality of each object, i.e., $Fitness = H(a, b, c, d, e)$. Initially, we randomly generate n objects for each evolution membrane and fill its best object into the corresponding local store membrane, and then fill the best object of entire system into global store membrane. Note that if a randomly generate object do not hold the increasing order $0 < a \leq b \leq c \leq d \leq e < 255$, we re-compute its components as follows:

$$\begin{cases} c' = c \\ b' = c' \times (b/255) \\ a' = b' \times (a/255) \\ d' = c' + (255 - c') \times (d/255) \\ e' = d' + (255 - d') \times (e/255) \end{cases} \tag{14}$$

In this work, multiple evolution membranes are designed to collaboratively evolve objects in the system, thus this will accelerate the exploitation of optimal segmentation parameters. The mutation operation and crossover operation of differential evolutionary (DE) algorithm are used as evolution rules of evolution membranes, however, we use a modified mutation operation according to special structure of the cell-like P system, which can be viewed as a variant of the rule "DE/current-to-best/1" in the DE . During one computing step, each evolution membrane will use the modified mutation operation to generate a mutation object for its every current object, X_i,

$$Y_i = X_i + F \cdot (X_{lbest} - X_i) + F(X_{gbest} - X_i) + F(X_{r_1} - X_{r_2}), \tag{15}$$

where X_{lbest} is the best object form the corresponding local archive membrane, X_{gbest} is the best object from global archive membrane, and X_{r_1}, X_{r_2} are two randomly selected objects from current objects. The scaling factor F is a positive control parameter for scaling the difference vector. The improved mutation rule is based on the point: two best objects, which are from different sources (local and global store membranes), will guide the evolution of objects and speed up the convergence, and can also improve the diversity of objects in the system.

After the mutation operation, crossover operation is applied to each pair of the current object and its corresponding mutant object to generate a trial object Z_i, and the crossover operation is defined as follows:

$$Z_i = \begin{cases} Y_i, \text{ if } \text{rand}_i \leq \ CR \text{ or } j = \text{rand}_j \\ X_i, \text{ otherwise} \end{cases} \tag{16}$$

where the crossover rate CR is a user-specified constant within the range $[0, 1]$, which controls the fraction of parameter values copied from the mutant object, and rand_j is a randomly chosen integer in the range $[1, 5]$.

The maximum execution step number is employed as halt condition in the proposed three-level thresholding approach based on cell-like P systems. When the system halts, the object in the skin membrane is regarded as the output of entire system.

The proposed three-level thresholding approach based on cell-like P systems is summarized as follows.

Three-level Thresholding Approach Based on Cell-like P Systems

```
program Three-level-thresholding
   Input:
```

```
  Number of evolution membranes m;
  Number of objects in each evolution membrane n;
  Maximum execution step number Smax;
  Scaling factor F;
  Crossover rate CR;
Output:
  Optimal thresholds (a,b,c,d,e);
begin
  Step 1: /* Initialization */
    for k=1 to m
       for j=1 to n
         /* Generate initial objects for each evolution
            membrane */
         X(k,j) = rand(5,255);
         /* Calculate the fitness value of the object
            according to Eq.(10) */
         Fit(k,j) = FitnessCalculation(X(k,j));
       end for
    end for
    Set computing step s = 0;
  Step 2:  /* Object evolution in membranes */
    Receive best object from global store membrane;
    for each evolution membrane k in parallel do
       Receive best object from its local store membrane;
       for j=1 to n
         Evolve the object X(k,j) in the evolution membrane k
            according to Eqs.(15)-(16);
         Fit(k,j) = FitnessCalculation(X(k,j));
       end for
       /* Update best object into its local store membrane and
          global store membrane */
       Update Xlbest(k) and Xgbest(k);
    end for
  Step 3: /* Halt condition judgment */
    If s > Smax is satisfied then
       Export object in skin membrane as (a,b,c,d,e);
       HALT;
    else
       s = s + 1;
       goto Step 2;
    end if
end.
```

4 Experimental results

The applicability and efficiency of the proposed image thresholding approach in image segmentation has be evaluated on three standard test images. These well-known images are Hunter, Lena and Peppers respectively, shown in Fig. 2(a)-(c). The test images are with size 512×512. Fig. 2(d)-(f) show the histograms of the three test images. In experiments, parameters of the proposed image thresholding approach based on cell-like P systems are given as follows:

(i) The used cell-like P system includes three evolution membranes ($m = 3$), where the number of objects contained in each evolution membrane is $n = 50$, and the maximum execution step number is $S_{max} = 100$;
(ii) In the used mutation rule (15), the scaling factor is set to be $F = 0.35$. In the used crossover rule (16), the crossover rate is set to be $CR = 0.2$.

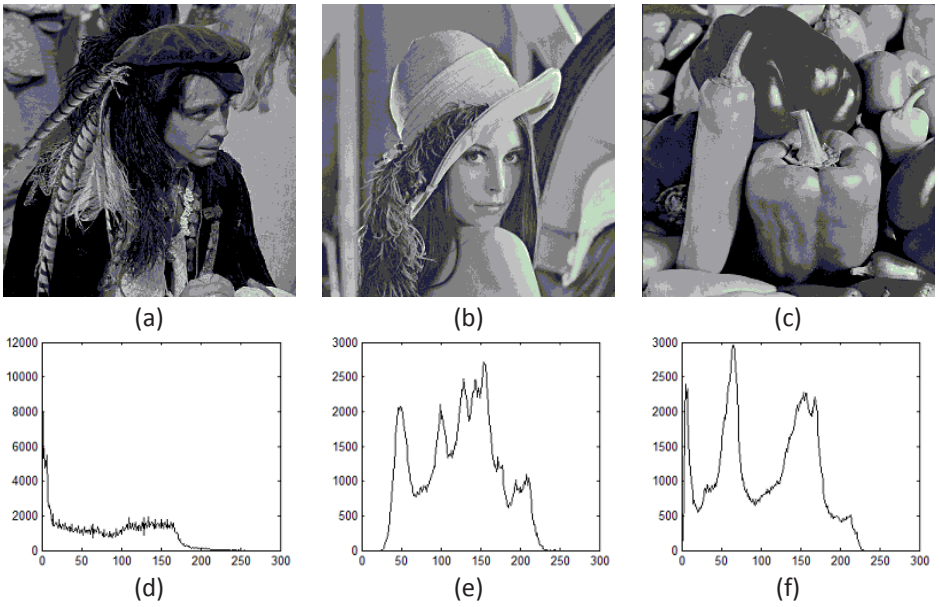


Fig. 2. Three test images ((a) Hunter; (b) Lena; (c) Peppers) and their histograms ((d) Hunter; (e) Lena; (f) Peppers).

In order to illustrate segmentation performance of the proposed three-level thresholding approach, its segmentation results are compared with the results obtained by PSO-based and GA-based approaches respectively. For the PSO-based approach, basic position-velocity model is employed and its parameters are set: population size $NP = 30$, maximum generation number $G_{max} = 100$,

$c_1 = c_2 = 1.0$, and w linearly varies from 0.9 to 0.4. For the GA-based approach, its parameters are given: population size $NP = 30$, crossover probability $P_c = 0.6$, mutation rate $P_m = 0.01$ and maximum generation number $G_{max} = 100$.

Table 1 lists their optimal segmentation thresholds. Fig. 3 gives optimal three-level segmentation results on above three test images for the proposed three-level thresholding approach based on cell-like P systems (in short, P systems), PSO-based approach (in short, PSO), GA-based approach (in short, GA), respectively. From the Fig. 3, we can see that results of the proposed three-level thresholding approach based on cell-like P systems is slightly better than that of PSO-based approach but evidently outperforms that of GA-based approach. This illustrates the applicability of the proposed approach for three-level thesholding.

Table 1. The optimal thresholds obtained by different methods.

Approaches	Lunter	Lena	Peppers
P systems	86, 179	98, 165	76, 152
PSO	82, 183	99, 166	80, 145
GA	73, 179	103, 168	85, 151

In order to investigate the efficiency, all approaches are compared based on the average CPU time (in seconds) taken to converge the solution. Comparison results of all methods given in Table 2. Form Table 2, it is clear that the proposed three-level thresholding approach based on cell-like P systems has fast convergence compared with PSO-based and GA-based approaches. The results demonstrate that the proposed three-level thresholding approach based on cell-like P systems is more efficient and effective that other approaches for three-level thresholding.

Table 2. Comparison of CPU time (in seconds) for different methods.

Approaches	Lunter	Lena	Peppers
P systems	7.975	7.641	7.012
PSO	9.521	9.136	9.849
GA	11.973	11.654	12.117

5 Conclusion

In this paper, we have presented a fast three-level thresholding approach based on cell-like P systems, which employed the total fuzzy entropy as the evaluation criterion. In order to effectively exploit the optimal segmentation thresholds, a special membrane structure with three layers was designed, which allows multiple membranes to co-evolve the objects of the system, and an improved evolution

Fig. 3. Three-level thresholding images obtained by different methods. (a)-(c) P systems; (d)-(f) PSO; (g)-(i) GA.

operation of DE was used as evolution rules of these membranes. With the special membrane structure and mechanism of the cell-like P system, two best objects were used to guide the evolution of the objects: one was best object from the corresponding local store membrane and another was from global store membrane. This mechanism not only effectively accelerates the speed of convergence but also enhances the diversity of objects in the system. The proposed thresholding approach based on cell-like P systems has been tested on several standard images and were compared with GA-based and PSO-based approaches. The experimental results showed the proposed thresholding approach outperforms the other approaches in terms of the applicability and computation efficiency. Further works are to be car-

ried out to feasibility of the proposed thresholding approach for various types of image processing applications.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (Grant No. 61170030), Foundation of Sichuan Provincial Key Discipline of Computer Software and Theory (No. SZD0802-09-1), Research Fund of Sichuan Key Laboratory of Intelligent Network Information Processing (No. SGXZD1002-10), and the Importance Project Foundation of Xihua University (No. Z1122632), China.

References

1. Păun, Gh.: Computing with Membranes. Journal of Computer System Sciences 61(1), 108–143 (2000)
2. Păun, Gh., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrance Computing. Oxford Unversity Press, New York (2010)
3. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking Neural P Systems. Fundameta Informaticae 71(2-3), 279–308 (2006)
4. Wang, J., Zhou, L., Peng, H., Zhang, G.X.: An Extended Spiking Neural P System for Fuzzy Knowledge Representation. International Journal of Innovative Computing, Information and Control 7(7A), 3709–3724 (2011)
5. Păun, Gh., Pérez-Jiménez, M.J.: Membrane Computing: Brief Introduction, Recent Results and Applications. BioSystem 85, 11–22 (2006)
6. Freund, R., Păun, Gh., Pérez-Jiménez, M.J.: Tissue-like P Systems with Channel-states. Theoretical Computer Science 330, 101–116 (2005)
7. Wang, J., Zhou, L., Peng, H., Zhang, G.X.: An Extended Spiking Neural P System for Fuzzy Knowledge Representation. International Journal of Innovative Computing, Information and Control 7(7A), 3709–3724 (2011)
8. Wang, H., Peng, H., Shao, J.: A Thresholding Method Based on P Systems for Image Segmentation. ICIC Express Letters, 6(1), 221–227 (2012)
9. Wang, T., Wang, J., Peng, H., Tu, M.: Optimization of PID Controller Parameters Based on PSOPS Algorithm. ICIC Express Letters, 6(1), 273–280 (2012)
10. Peng, H., Wang, J., Perez-Jimenez, M.J., Wang, H., Shao, J., Wang, T.: Fuzzy Reasoning Spiking Neural P System for Fault Diagnosis. Information Sciences, 2012. (Accepted)
11. Nishida, T.Y.: An Application of P-system: A New Algorithm for NP-complete Optimization Problems. In: Proc. 8th World Multi-Conference on Systemics, Cybernetics and Informatics, 109–112 (2004)
12. Zaharie, D., Ciobanu, G.: Distributed Evolutionary Algorithms Inspired by Membranes in Solving Continuous Optimization Problems. Lecture Notes in Computer Science, vol. 4361, 536–553 (2006)
13. Huang, L., Suh, I.H., Abraham, A.: Dynamic Multi-objective Optimization Based on Membrane Computing for Control of Time-varying Unstable Plants. Information Sciences 181, 2370–2391 (2011)

14. Zhang, G.-X., Liu, C.-X., Rong, H.-N.: Analyzing Radar Emitter Signals with Membrane Algorithms. Mathematical and Computer Modelling 52, 1997–2010 (2010)
15. Sahoo, P.K., Soltani, S., Wong, A.K.C., Chen, Y.C.: A survey of Thresholding Techniques. Comput. Vis. Graph. Image Process. 41(2), 233–260 (1988)
16. Pikaz, A., Averbuch, A.: Digital Image Thresholding Based on Topological Stable State. Pattern Recognit. 29(5), 829–843 (1996)
17. Huang, L.K., Wang, M.J.: Image Thresholding by Minimizing The Measure of Fuzziness. Pattern Recognition 28, 41–51 (1995)
18. Otsu, N.: A Threshold Selection Method from Gray Level Histograms. IEEE Transactions on Systems, Man and Cybernetics SMC 9(1), 62–66 (1979)
19. Kapur, J.N., Sahoo, P.K., Wong, A.K.C.: A New Method for Gray-level Picture Thresholding Using The Entropy of The Histogram. Computer Vision, Graphics and Image Processing 29(3), 273–285 (1985)
20. Liao, P.S., Chen, T.S., Chung, P.C.: A Fast Algorithm for Multilevel Thresholding. Journal of Information Sciences and Engineering 17(5), 713–727 (2001)
21. Cheng, H.D., Chen, Y.H., Jiang, X.H.: Thresholding Using Two-dimensional Histogram and Fuzzy Entropy Principle. IEEE Trans. on Image Processing 9(4), 732-735 (2000)
22. Shelokar, P.S., Jayaraman, V.K, Kulkarni, B.D.: An Ant Colonyapproach for Clustering. Anal. Chim. Acta 59, 187–195 (2004)
23. Zhao, M.S., Fu, A.M.N., Yan, H.: A Technique of Threelevel Thresholding Based on Probability Partition and Fuzzy 3-partition. IEEE Trans. on Fuzzy Systems 9(3), 469–479 (2001)
24. Liu, D., Jiang, Z.H., Feng, H.Q.: A Novel Fuzzy Classification Entropy Approach to Image Thresholding. Pattern Recognition Letters 27, 1968–1975 (2006)
25. Yin, P.Y.: A Fast Scheme for Optimal Thresholding Using Genetic Algorithms. Signal Processing 72, 85–95 (1999)
26. Cheng, H.D., Chen, Y.H., Sun, Y.: A Novel Fuzzy Entropy Approachto Image Enhancement and Thresholding. Signal Processing 75, 277–301 (1999)
27. Tao, W.B., Tian, J.W., Liu, J.: Image Segmentation by Three-level Thresholding Based on Maximum Fuzzy Entropy and Genetic Algorithm. Pattern Recognit. Lett. 24, 3069–3078 (2003)
28. Hammouche, K., Diaf, M., Siarry, P.: A Multilevel Automatic Thresholding Method Based on A Genetic Algorithm for A Fast Image Segmentation. Computer Vision and Image Understanding 109, 163–175 (2008)
29. Zahara, E., Fan, S.-K.S., Tsai, D.M.: Optimal Multi-thresholding Using A Hybrid Optimization Approach. Pattern Recognit. Lett. 26, 1085–1095 (2005)
30. Maitra, M., Chatterjee, A.: A Hybrid Cooperative-comprehensive Learning Based PSO Algorithm for Image Segmentation Using Multilevel Thresholding. Expert Systems with Applications 34, 1341–1350 (2008)
31. Gao, H., Xu, W.B., Sun, J., Tang, Y.L.: Multilevel Thresholding for Image Segmentation Through An Improved Quantum-behaved Particle Swarm Algorithm. IEEE Trans. On Instrumentation and Measurement 59(4), 934–946 (2010)
32. Tao, W.B., Jin, H., Liu, L.M.: Object Segmentation Using Ant Colony Optimization Algorithm and Fuzzy Entropy. Pattern Recognit. Lett. 28(7), 788–796 (2008)
33. Díaz-Pernil, D., Gutiérrez-Naranjo M.A., Real P., Sánchez-Canales V.: A Cellular Way to Obtain Homology Groups in Binary 2D Images. Eighth Brainstorming Week on Membrane Computing, 89–100 (2010)
34. Wang, H., Peng, H., Shao, J.: A Thresholding Method Based on P Systems for Image Segmentation. ICIC Express Letters 6(1), 221–227 (2012)

The Role of the Environment in Tissue P Systems with Cell Division

Mario J. Pérez-Jiménez[1], Agustín Riscos-Núñez[1], Miquel Rius-Font[2], Francisco J. Romero-Campero[1]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: marper@us.es, ariscosn@us.es, fran@us.es

[2] Department of Applied Mathematics IV
Universitat Politécnica de Catalunya, Spain
E-mail: mrius@ma4.upc.edu

Summary. Classical tissue P systems with cell division have a special alphabet whose elements appear at the initial configuration of the system in an arbitrary large number of copies. These objects are shared in a distinguished place of the system, called the environment. Besides, the ability of these computing devices to have infinite copies of some objects has been widely exploited in the design of efficient solutions to computationally hard problems.

This paper deals with computational aspects of tissue P systems with cell division where there is not an environment having the property mentioned above. Specifically, we establish the relationships between the polynomial complexity class associated with tissue P systems with cell division and with or without environment. As a consequence, we prove that it is not necessary to have infinite copies of some objects at the initial configuration in order to solve **NP**–complete problems in an efficient way.

Key words: Membrane Computing, Tissue P Systems, Cell Division, Environment of a tissue, Computational Complexity.

1 Preliminaries

An *alphabet*, Γ, is a non–empty set whose elements are called *symbols*. An ordered finite sequence of symbols is a *string* or *word*. If u and v are strings over Γ, then so is their *concatenation* uv, obtained by juxtaposition, that is, writing u and v one after the other. The number of symbols in a string u is the *length* of the string and it is denoted by $|u|$. As usual, the empty string (with length 0) will be denoted by λ. The set of all strings over an alphabet Γ is denoted by Γ^*. In algebraic terms, Γ^*

is the free monoid generated by Γ under the operation of concatenation. Subsets, finite or infinite, of Γ^* are referred to as *languages* over Γ.

The set of symbols occurring in a string $u \in \Gamma^*$ is denoted by $alph(u)$.

The *Parikh vector* associated with a string $u \in \Gamma^*$ with respect to the alphabet $\Sigma = \{a_1, \ldots, a_r\} \subseteq \Gamma$ is $\Psi_\Sigma(u) = (|u|_{a_1}, \ldots, |u|_{a_r})$, where $|u|_{a_i}$ denotes the number of ocurrences of symbol a_i in string u. This is called the *Parikh mapping* associated with Σ. Notice that, in this definition, the ordering of the symbols from Σ is relevant. If $\Sigma_1 = \{a_{i_1}, \ldots, a_{i_r}\} \subseteq \Gamma$, then we define $\Psi_{\Sigma_1}(u) = (|u|_{a_{i_1}}, \ldots, |u|_{a_{i_r}})$, for each $u \in \Gamma^*$.

A *multiset* m over a set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in A \mid f(x) > 0\}$. A multiset is empty (resp. finite) if its support is the empty set (resp. a finite set). If $m = (A, f)$ is a finite multiset over A and $supp(m) = \{a_1, \ldots, a_k\}$, then it will be denoted as $m = \{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}$. That is, superscripts indicate the multiplicity of each element, and if $f(x) = 0$ for $x \in A$, then element x is omitted. A finite multiset $m = \{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}$ can also be represented by the string $a_1^{f(a_1)} \ldots a_k^{f(a_k)}$ over the alphabet $\{a_1, \ldots, a_k\}$. Nevertheless, all permutations of this string identify the same multiset m precisely. Throughout this paper, we speak about "the finite multiset m" where m is a string, meaning "the finite multiset represented by the string m". If $m_1 = (A, f_1)$, $m_2 = (A, f_2)$ are multisets over A, then we define the union of m_1 and m_2 as $m_1 + m_2 = (A, g)$, where $g = f_1 + f_2$, that is, $g(a) = f_1(a) + f_2(a)$, for each $a \in A$.

For any sets A and B the *relative complement* $A \setminus B$ of B in A is defined as follows: $A \setminus B = \{x \in A \mid \ x \notin B\}$.

Finally, for any set A we denote $|A|$ the cardinal (number of elements) of A, as usual.

In what follows, we assume the reader is already familiar with the basic notions and terminology of P systems. For details, see [4].

2 Tissue P Systems with communication rules

Definition 2.1 *A tissue P system with communication rules of degree $q \geq 1$ is a tuple $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$, where:*

1. *Γ is a finite* alphabet *whose elements are called* objects*;*
2. *$\mathcal{E} \subseteq \Gamma$;*
3. *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are strings over Γ, representing finite multisets of objects;*
4. *$\mathcal{R}$ is a finite set of communication rules of the form $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$, $|u| + |v| > 0$;*
5. *$i_{out} \in \{0, 1, 2, \ldots, q\}$.*

A tissue P system <u>*without environment*</u> is a tissue P system such that $\mathcal{E} = \emptyset$. In this case, alphabet $\mathcal{E}$ can be removed from the tuple.

A *tissue P system with communication rules* $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$, of degree $q \geq 1$ can be viewed as a set of q cells, labelled by $1, \ldots, q$, with an environment labelled by 0 such that: (a) $\mathcal{M}_1, \ldots, \mathcal{M}_q$ represent the finite multisets of objects initially placed in the q cells of the system; (b) $\mathcal{E}$ is the set of objects initially located in the environment of the system, all of them available in an arbitrary number of copies; and (c) $i_{out} \in \{0, 1, 2, \ldots, q\}$ represents a distinguished cell or the environment which will encode the output of the system. We use the term *region* i $(0 \leq i \leq q)$ to refer cell i in the case $1 \leq i \leq q$ and to refer the environment in the case $i = 0$.

When applying a rule $(i, u/v, j)$, the objects of the multiset represented by u are sent from region i to region j and, simultaneously, the objects of multiset v are sent from region j to region i. The length of the communication rule $(i, u/v, j)$ is defined as $|u| + |v|$.

A communication rule $(i, u/v, j)$ is called a *symport rule* if $u = \lambda$ or $v = \lambda$. A symport rule $(i, u/\lambda, j)$, with $i \neq 0, j \neq 0$, provides a virtual arc from cell i to cell j. A communication rule $(i, u/v, j)$ is called an *antiport rule* if $u \neq \lambda$ and $v \neq \lambda$. An antiport rule $(i, u/v, j)$, with $i \neq 0, j \neq 0$, provides two arcs: one from cell i to cell j and another one from cell j to cell i. Thus, every tissue P system has an underlying directed graph whose nodes are the cells of the system and the arcs are obtained from communication rules. In this context, the environment can be considered as a virtual node of the graph such that its connections are defined by communication rules of the form $(i, u/v, j)$, with $i = 0$ or $j = 0$.

The rules of a system like the one above are used in a non-deterministic maximally parallel manner as it is customary in membrane computing. At each step, all cells which can evolve must evolve in a maximally parallel way (at each step we apply a multiset of rules which is maximal, no further applicable rule can be added).

An *instantaneous description* or a *configuration* at any instant of a tissue P system with communication rules is described by all multisets of objects over Γ associated with all the cells present in the system, and the multiset of objects over $\Gamma - \mathcal{E}$ associated with the environment at that moment. Bearing in mind that the objects from $\mathcal{E}$ have infinite copies in the environment, they are not properly changed along the computation. The *initial configuration* is $(\mathcal{M}_1, \cdots, \mathcal{M}_q; \emptyset)$. A configuration is a *halting configuration* if no rule of the system is applicable to it.

Let us fix a tissue P system with communication rules Π. We say that configuration $\mathcal{C}_1$ yields configuration $\mathcal{C}_2$ in one *transition step*, denoted $\mathcal{C}_1 \Rightarrow_\Pi \mathcal{C}_2$, if we can pass from $\mathcal{C}_1$ to $\mathcal{C}_2$ by applying the rules from $\mathcal{R}$ following the previous remarks. A *computation* of Π is a (finite or infinite) sequence of configurations such that:

1. the first term of the sequence is the initial configuration of the system;
2. each non-initial configuration of the sequence is obtained from the previous configuration by applying the rules of the system in a maximally parallel manner with the restrictions previously mentioned; and

3. if the sequence is finite (called *halting computation*), then the last term of the sequence is a halting configuration.

All computations start from an initial configuration and proceed as stated above; only halting computations give a result, which is encoded by the objects present in the output region i_{out} in the halting configuration.

We denote by $\mathbf{Comp}(\Pi)$ the set of computations of the tissue P system Π. If $\mathcal{C} = \{\mathcal{C}_i\}_{i<r+1}$ of Π ($r \in \mathbf{N}$) is a halting computation, then the *length of* $\mathcal{C}$ is r, that is, the number of non-initial configurations which appear in the finite sequence $\mathcal{C}$. We denote it by $|\mathcal{C}|$. We also denote by $\mathcal{C}_i(j)$ the contents of cell j at configuration $\mathcal{C}_i$.

3 Tissue P Systems with Cell Division

Cell division is an elegant process that enables organisms to grow and reproduce. Mitosis is a process of cell division which results in the production of two daughter cells from a single parent cell. Daughter cells are identical to one another and to the original parent cell. Through a sequence of steps, the replicated genetic material in a parent cell is equally distributed to two daughter cells. While there are some subtle differences, mitosis is remarkably similar across organisms.

Before a dividing cell enters mitosis, it undergoes a period of growth where the cell replicates its genetic material and organelles. Replication is one of the most important functions of a cell. DNA replication is a simple and precise process that creates two complete strands of DNA (one for each daughter cell) where only one existed before (from the parent cell).

Let us recall that the model of *tissue P systems with cell division* is based on the cell-like model of P systems with membranes division [3]. In these models, the cells are not polarized; the cells obtained by division have the same labels as the original cell, and if a cell is divided, its interaction with other cells or with the environment is locked during the division process. In some sense, this means that while a cell is dividing it closes its communication channels.

Definition 3.1 *A tissue P system with cell division of degree* $q \geq 1$ *is a tuple* $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$*, where:*

1. Γ *is a finite* alphabet *whose elements are called* objects*;*
2. $\mathcal{E} \subseteq \Gamma$*;*
3. $\mathcal{M}_1, \ldots, \mathcal{M}_q$ *are strings over* Γ*, representing finite multisets of objects;*
4. $\mathcal{R}$ *is a finite set of rules of the following forms:*
   (a) Communication rules*:* $(i, u/v, j)$*, for* $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$*,* $u, v \in \Gamma^*$*,* $|u| + |v| > 0$*;*
   (b) Division rules*:* $[a]_i \rightarrow [b]_i[c]_i$*, where* $i \in \{1, 2, \ldots, q\}$*,* $i \neq i_{out}$ *and* $a, b, c \in \Gamma$*;*
5. $i_{out} \in \{0, 1, 2, \ldots, q\}$.

A *tissue P system with cell division* is a tissue P system with communication rules where also division rules are allowed. When applying a division rule $[a]_i \to [b]_i[c]_i$, under the influence of object a, the cell with label i is divided into two cells with the same label; in the first copy, object a is replaced by object b, in the second one, object a is replaced by object c; all the other objects residing in cell i are replicated and copies of them are placed in the two new cells. The output cell i_{out} cannot be divided.

The rules of a tissue P system with cell division are applied in a non-deterministic maximally parallel manner as it is customary in membrane computing. At each step, all cells which can evolve must evolve in a maximally parallel way (at each step we apply a multiset of rules which is maximal, no further applicable rule can be added), with the following important remark: if a cell divides, then the division rule is the only one which is applied for that cell at that step; the objects inside that cell do not evolve by means of communication rules. In other words, before division a cell interrupts all its communication channels with the other cells and with the environment. The new cells resulting from division will interact with other cells or with the environment only at the next step – providing that they do not divide once again. The label of a cell precisely identifies the rules which can be applied to it.

4 Recognizer Tissue P Systems

Let us recall that a *decision problem* is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X. Many abstract problems are not decision problems. For example, in *combinatorial optimization problems* some value must be optimized (minimized or maximized). In order to deal with such problems, they can be transformed into roughly equivalent decision problems by supplying a target/threshold value for the quantity to be optimized, and then asking whether this value can be attained.

A natural correspondence between decision problems and languages can be established as follows. Given a decision problem $X = (I_X, \theta_X)$, its associated language is $L_X = \{w \in I_X : \theta_X(w) = 1\}$. Conversely, given a language L, over an alphabet Γ, its associated decision problem is $X_L = (I_{X_L}, \theta_{X_L})$, where $I_{X_L} = \Gamma^*$, and $\theta_{X_L} = \{(x, 1) : x \in L\} \cup \{(x, 0) : x \notin L\}$. The solvability of decision problems is defined through the recognition of the languages associated with them.

In order to study the computing efficiency, the notions from classical *computational complexity theory* are adapted for membrane computing, and a special class of cell-like P systems is introduced in [7]: *recognizer P systems* (called *accepting P systems* in a previous paper [6]). For tissue P systems, with the same idea as recognizer cell-like P systems, *recognizer tissue P systems* is introduced in [5].

Definition 4.1 *A* recognizer tissue P system *with cell division of degree* $q \geq 1$ *is a tuple* $\Pi = (\Gamma, \Sigma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$*, where:*

- $(\Gamma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$ *is a tissue P system with cell division of degree* $q \geq 1$*, as defined in the previous section.*
- *The working alphabet* Γ *has two distinguished objects* `yes` *and* `no`*, at least one copy of them present in some initial multisets* $\mathcal{M}_1, \ldots, \mathcal{M}_q$*, but none of them is present in* $\mathcal{E}$*.*
- Σ *is an (input) alphabet strictly contained in* Γ *such that* $\mathcal{E} \cap \Sigma = \emptyset$*.*
- $\mathcal{M}_1, \ldots, \mathcal{M}_q$ *are strings over* $\Gamma \setminus \Sigma$*.*
- $i_{in} \in \{1, \ldots, q\}$ *is the input cell.*
- *The output region* i_{out} *is the environment. In the case of tissue without environment,* i_{out} *is a distinguished cell, that is* $i_{out} \in \{1, \ldots, q\}$*.*
- *All computations halt.*
- *If* $\mathcal{C}$ *is a computation of* Π*, then either object* `yes` *or object* `no` *(but not both) must have been released into the environment, and only at the last step of the computation.*

For each multiset m over Σ, the *computation of the system* Π *with input* m starts from the configuration of the form $(\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_{i_{in}} + m, \ldots, \mathcal{M}_q; \emptyset)$, that is, the input multiset m has been added to the contents of the input cell i_{in}, and we denote it by $\Pi + m$. Therefore, we have an initial configuration associated with each input multiset m (over the input alphabet Σ) in this kind of systems.

Given a recognizer tissue P system with cell division, and a halting computation $\mathcal{C} = \{\mathcal{C}_i\}_{i<r+1}$ of Π ($r \in \mathbf{N}$), we define the result of $\mathcal{C}$ as follows:

$$Output(\mathcal{C}) = \begin{cases} \texttt{yes}, & \text{if } \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{r,i_{out}}) = (1,0) \wedge \\ & \quad \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{i,i_{out}}) = (0,0) \text{ for } i = 0, \ldots, r-1 \\ \texttt{no}, & \text{if } \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{r,i_{out}}) = (0,1) \wedge \\ & \quad \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{i,i_{out}}) = (0,0) \text{ for } i = 0, \ldots, r-1 \end{cases}$$

where Ψ is the Parikh mapping, and $M_{i,i_{out}}$ is the multiset over $\Gamma \setminus \mathcal{E}$ associated with the output region at the configuration $\mathcal{C}_i$, in particular, $M_{r,i_{out}}$ is the multiset over $\Gamma \setminus \mathcal{E}$ associated with the output region at the halting configuration $\mathcal{C}_r$.

We say that a computation $\mathcal{C}$ is an *accepting computation* (respectively, *rejecting computation*) if $Output(\mathcal{C}) =$ `yes` (respectively, $Output(\mathcal{C}) =$ `no`), that is, if object `yes` (respectively, object `no`) appears in the output region associated with the corresponding halting configuration of $\mathcal{C}$, and neither object `yes` nor `no` appears in the output region associated with any non–halting configuration of $\mathcal{C}$.

Let us notice that if a recognizer tissue P system

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

has a rule of the type $(i, \lambda/u, 0)$ then $alph(u) \cap (\Gamma \setminus \mathcal{E}) \neq \emptyset$, because on the contrary all computations of Π would be non halting.

For each natural number $k \geq 1$, we denote by $\mathbf{TDC}(k)$ the class of recognizer tissue P systems with cell division and with communication rules of length at most k. In the case of tissue P systems without environment, we denote by $\widehat{\mathbf{TDC}}(k)$ the class of recognizer tissue P systems with cell division and with communication rules of length at most k.

5 Polynomial Complexity Classes of Tissue P systems

Next, we define what solving a decision problem in the framework of tissue P systems in a uniform and efficient way means. Bearing in mind that they provide devices with a finite description, a numerable family of tissue P systems will be necessary in order to solve a decision problem.

Definition 5.1 *We say that a decision problem $X = (I_X, \theta_X)$ is solvable in a uniform way and polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of recognizer tissue P systems (with symport/antiport rules, with cell division or with cell separation) if the following holds:*

- *The family $\mathbf{\Pi}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$.*
- *There exists a pair (cod, s) of polynomial-time computable functions over I_X such that:*
  - *for each instance $u \in I_X$, $s(u)$ is a natural number, and $cod(u)$ is an input multiset of the system $\Pi(s(u))$;*
  - *for each $n \in \mathbb{N}$, $s^{-1}(n)$ is a finite set;*
  - *the family $\mathbf{\Pi}$ is* polynomially bounded *with regard to (X, cod, s), that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(s(u))$ with input $cod(u)$ is halting and it performs at most $p(|u|)$ steps;*
  - *the family $\mathbf{\Pi}$ is* sound *with regard to (X, cod, s), that is, for each $u \in I_X$, if there exists an accepting computation of $\Pi(s(u))$ with input $cod(u)$, then $\theta_X(u) = 1$;*
  - *the family $\mathbf{\Pi}$ is* complete *with regard to (X, cod, s), that is, for each $u \in I_X$, if $\theta_X(u) = 1$, then every computation of $\Pi(s(u))$ with input $cod(u)$ is an accepting one.*

From the soundness and completeness conditions above we deduce that every P system $\Pi(n)$ is *confluent*, in the following sense: every computation of a system with the *same* input multiset must always give the *same* answer.

Let $\mathbf{R}$ be a class of recognizer tissue P systems. We denote by $\mathbf{PMC_R}$ the set of all decision problems which can be solved in a uniform way and polynomial time by means of families of systems from $\mathbf{R}$. The class $\mathbf{PMC_R}$ is closed under complement and polynomial–time reductions [6].

Next, we prove a technical result concerning recognizer tissue P systems.

Lemma 5.2 *Let $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ a family of recognizer tissue P systems solving a decision problem $X = (I_X, \theta_X)$ in polynomial time according to the previous definition. Let (cod, s) a polynomial encoding associated with that solution. Let $r(n)$ be a polynomial function such that for each $u \in I_X$ every computation of $\Pi(s(u)) + cod(u)$ is halting and it performs at most $r(|u|)$ steps. Then, there exists a polynomial function $p(n)$ such that for each instance $u \in I_X$, $2^{p(|u|)}$ is an*

upper bound of the number of objects from $\mathcal{E}$ which are moved from the environment to all cells of the system $\Pi(s(u))+cod(u)$ by communication rules along any computation.

Proof: Let $u \in I_X$ be an instance of X and

$$\Pi(s(u)) + cod(u) = (\Gamma, \Sigma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$$

Let $k \in \mathbb{N}$ be such that $\Pi(s(u)) + cod(u) \in \mathbf{TDC}(k)$. Let $M = |\mathcal{M}_1 + \cdots + \mathcal{M}_q|$. Then, any computation of $\Pi(s(u)) + cod(u)$ performs, at most, $r(|u|)$ transition steps. Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_m)$, $0 \leq m \leq r(|u|)$, be a computation of Π. For each t, $0 \leq t \leq m$ and i, $1 \leq i \leq q$, we denote by $\mathcal{C}_t(i)$ the multiset of objects over Γ in cell i at time t. We also denote $\mathcal{C}_t(0)$ the multiset of objects over $\Gamma \setminus \mathcal{E}$ in the environment at time t.

Let us suppose that we apply only communication rules at m consecutive transition steps. At this situation, for each t $(0 \leq t \leq m)$ we compute an upper bound of $|\mathcal{C}_t(0)+\mathcal{C}_t(1)+\ldots+\mathcal{C}_t(q)|$. Then, for each i, j $(0 \leq i, j \leq q, i \neq j)$ we denote by $A_t(i,j)$ the multiset of objects being moved from region j to region i by applying rules of the type $(i, u/v, j)$ at time t.

Let us construct α_t, $0 \leq t \leq m$, an upper bound of the number of objects which appear in the whole system (taking all cells into account) at time t. That is,

$$\alpha_t \geq |\mathcal{C}_t(0) + \mathcal{C}_t(1) + \ldots + \mathcal{C}_t(q)|$$

The construction is made by induction on t. For $t = 0$ we consider $\alpha_0 = M$. Let t be such that $0 \leq t < m$ and for each t' $(0 \leq t' \leq t)$ let us assume that we have constructed $\alpha_{t'}$ such that

$$\alpha_{t'} \geq \sum_{i=0}^{q} |\mathcal{C}_{t'}(i)|$$

The number of objects moved into cell i $(1 \leq i \leq q)$ at instant $t+1$ is

$$A_t(i,0) + \sum_{j=1, j\neq i}^{q} A_t(i,j)$$

The number of objects sent to the environment at instant $t+1$ is $\sum_{j=1}^{q} A_t(0,j)$.

Notice that objects coming to region i from some other cell j were already present in the previous configuration. Besides, in order to trigger a communication rule bringing objects from the environment into region i, at least one object in region i is required, or else one symbol from $\Gamma \setminus \mathcal{E}$ in the environment. Finally, recall that the length of communication rules is bounded by k.

From these considerations, we deduce:

$$\sum_{i=1}^{q} \sum_{j=1, j\neq i}^{q} |A_t(i,j)| \leq \alpha_t \quad \text{and} \quad \sum_{i=1}^{q} |A_t(i,0)| \leq \alpha_t \cdot k$$

Besides,

$$\sum_{j=1}^{q} |A_t(0,j)| \leq \alpha_t \cdot k$$

Then, we can consider $\alpha_{t+1} = \alpha_t + \alpha_t \cdot k + \alpha_t \cdot k = \alpha_t \cdot (1+2k)$. Thus, for each t $(0 \leq t \leq m)$ we define $\alpha_t = M \cdot (1+2k)^t$. Hence, if we applied in a consecutive way the maximum possible number of communication rules (without applying any division rules) to the system $\Pi(s(u)) + cod(u)$, in any instant of any computation of the system, $M \cdot (1+2k)^{r(|u|)}$ is an upper bound of the number of objects in the whole system.

Now, let us consider the effects of applying in a consecutive way the maximum possible number of division rules (without applying any communication rules) to the system $\Pi(s(u)) + cod(u)$ when the initial configuration has $M \cdot (1+2k)^{r(|u|)}$ objects. After that, an upper bound of the number of objects in the whole system by any computation is $M \cdot (1+2k)^{r(|u|)} \cdot 2^{r(|u|)} \cdot r(|u|)$. Hence, for each instance $u \in I_X$ the number of objects from $\mathcal{E}$ which are moved from the environment to the whole cells of the system $\Pi(s(u))+cod(u)$ is, at most, $M \cdot (1+2k)^{r(|u|)} \cdot 2^{r(|u|)} \cdot r(|u|)$.

Then, we consider a polynomial function $p(n)$ such that

$$p(|u|) \geq \log(M) + r(|u|) \cdot \log(1+2k) + r(|u|) + \log(r(|u|))$$

for each instance $u \in I_X$. The polynomial function $p(n)$ fulfills the property required at the Lemma.

□

6 Simulating tissue P systems with cell division by means of tissue P systems with cell division and without environment

The goal of this section is to show that any tissue P system with cell division can be simulated by a tissue P system with cell division and without environment in an efficient way.

First of all, we define the meaning of efficient simulations in the framework of recognizer tissue P systems.

Definition 6.1 *Let Π and Π' be recognizer tissue P systems. We say that Π'* simulates *Π* in an efficient way *if the following holds:*

1. *Π' can be constructed from Π by a deterministic Turing machine working in polynomial time.*
2. *There exists an injective function, f, from the set $\mathbf{Comp}(\Pi)$ of computations of Π onto the set $\mathbf{Comp}(\Pi')$ of computations of Π' such that:*
   - ⋆ *There exists a deterministic Turing machine that constructs computation $f(\mathcal{C})$ from computation $\mathcal{C}$ in polynomial time.*
   - ⋆ *A computation $\mathcal{C} \in \mathbf{Comp}(\Pi)$ is an accepting computation if and only if $f(\mathcal{C}) \in \mathbf{Comp}(\Pi')$ is an accepting one.*

- ⋆ *There exists a polynomial function $p(n)$ such that for each $\mathcal{C} \in \mathbf{Comp}(\Pi)$ we have $|f(\mathcal{C})| \leq p(|\mathcal{C}|)$.*

Now, for every family of recognizer tissue P system with cell division solving a decision problem, we design a family of recognizer tissue P systems with cell division and *without environment* efficiently simulating it, according to Definition 6.1.

In what follows throghout this Section, let $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ a family of recognizer tissue P systems solving a decision problem $X = (I_X, \theta_X)$ in polynomial time according to Definition 5.1, and let $p(n)$ be a polynomial function such that for each instance $u \in I_X$, $2^{p(|u|)}$ is an upper bound of the number of objects from $\mathcal{E}$ which are moved from the environment to all cells of the system by any computation of $\Pi(s(u)) + cod(u)$.

Definition 6.2 *For each $n \in \mathbb{N}$, let $\Pi(n) = (\Gamma, \Sigma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$ an element of the previous family of degree q and for the sake of simplicity we denote p instead of $p(n)$. Let us consider the recognizer tissue P system of degree $q_1 = 1 + q \cdot (p+2) + |\mathcal{E}|$ with cell division and without environment*

$$\mathbf{S}(\Pi(n)) = (\Gamma', \Sigma', \mathcal{M}'_0, \mathcal{M}'_1, \ldots, \mathcal{M}'_{q_1}, \mathcal{R}', i'_{in}, i'_{out})$$

defined as follows:

- $\Gamma' = \Gamma \cup \{\alpha_i : 0 \leq i \leq p-1\}$.
- $\Sigma' = \Sigma$.
- *Each cell $i \in \{1, \ldots, q\}$ of Π provides a cell of $\mathbf{S}(\Pi(n))$ with the same label. In addition, $\mathbf{S}(\Pi(n))$ has:*
  - *$p+1$ new cells, labelled by $(i,0), (i,1), \ldots, (i,p)$, respectively, for each $i \in \{1, \ldots, q\}$.*
  - *A distinguished cell labelled by 0.*
  - *A new cell, labelled by l_b, for each $b \in \mathcal{E}$.*
- *Initial multisets: $\mathcal{M}'_{l_b} = \{\alpha_0\}$, for each $b \in \mathcal{E}$, and*

$$\left.\begin{array}{rcl} \mathcal{M}'_{(i,0)} & = & \mathcal{M}_i \\ \mathcal{M}'_{(i,1)} & = & \emptyset \\ \ldots\ldots & & \ldots \\ \mathcal{M}'_{(i,p)} & = & \emptyset \\ \mathcal{M}'_i & = & \emptyset \end{array}\right\} (1 \leq i \leq q)$$

- *Set of rules:*

$$\begin{array}{l} \mathcal{R}' = \mathcal{R} \cup \{[\alpha_j]_{l_b} \to [\alpha_{j+1}]_{l_b}\ [\alpha_{j+1}]_{l_b} : b \in \mathcal{E} \wedge 0 \leq j \leq p-2\} \\ \qquad \cup \{[\alpha_{p-1}]_{l_b} \to [b]_{l_b}\ [b]_{l_b} : b \in \mathcal{E}\} \\ \qquad \cup \{(l_b,\ b/\lambda\ , 0) : b \in \mathcal{E}\} \\ \qquad \cup \{\big((i,j),\ a/\lambda\ , (i,j+1)\big) : a \in \Gamma \wedge 1 \leq i \leq q \wedge 0 \leq j \leq p-1\} \\ \qquad \cup \{\big((i,p),\ a/\lambda\ , i\big) : a \in \Gamma \wedge 1 \leq i \leq q\} \end{array}$$

- $i'_{in} = (i_{in}, 0)$, *and* $i'_{out} = 0$.

Let us notice that $\mathbf{S}(\Pi(n))$ can be considered as an **extension** of $\Pi(n)$ **without environment**, in the following sense:

⋆ $\Gamma \subseteq \Gamma', \Sigma \subseteq \Sigma'$ and $\mathcal{E} = \emptyset$.
⋆ Each cell in Π is also a cell in $\mathbf{S}(\Pi(n))$.
⋆ There is a distinguished cell in $\mathbf{S}(\Pi(n))$ labelled by 0 which plays the role of environment of $\Pi(n)$.
⋆ $\mathcal{R} \subseteq \mathcal{R}'$, and now 0 is the label of a "normal cell" in $\mathbf{S}(\Pi(n))$.

Next, we analyze the structure of the computations of system $\mathbf{S}(\Pi(n))$ and we compare them with the computations of $\Pi(n)$.

Lemma 6.3 *Let* $\mathcal{C}' = (\mathcal{C}'_0, \mathcal{C}'_1, \dots)$ *be a computation of* $\mathbf{S}(\Pi(n))$. *For each* t $(1 \leq t \leq p)$ *the following holds:*

- $\mathcal{C}'_t(i) = \emptyset$, *for* $0 \leq i \leq q$.
- *For each* $1 \leq i \leq q$, *and* $0 \leq j \leq p$ *we have:*

$$\mathcal{C}'_t(i,j) = \begin{cases} \mathcal{M}_i, \text{ if} & j = t \\ \emptyset, \quad \text{if} & j \neq t \end{cases}$$

- *For each* $b \in \mathcal{E}$, *there exist* 2^t *cells labelled by* l_b *whose content is:*

$$\mathcal{C}'_t(l_b) = \begin{cases} \alpha_t, \text{ if} & 1 \leq t \leq p-1 \\ b, \quad \text{if} & t = p \end{cases}$$

Proof: By induction on t.

Let us start with the basic case $t = 1$. The initial configuration of system $\mathbf{S}(\Pi(n))$ is the following:

- $\mathcal{C}'_0(i) = \emptyset$, for $0 \leq i \leq q$.
- For each $1 \leq i \leq q$ we have $\mathcal{C}'_0(i,0) = \mathcal{M}_i$, and $\mathcal{C}'_0(i,j) = \emptyset$, for $1 \leq j \leq p$.
- For each $b \in \mathcal{E}$, there exists only one cell labelled by l_b whose contents is $\{\alpha_0\}$.

At configuration $\mathcal{C}'_0$, only the following rules are applicable:

- $[\alpha_0]_{l_b} \to [\alpha_1]_{l_b}\ [\alpha_1]_{l_b}$, for each $b \in \mathcal{E}$.
- $\big((i,0),\ a/\lambda\ , (i,1)\big)$, for each $a \in \mathit{supp}(\mathcal{M}_i)$.

Thus,

- For each i $(1 \leq i \leq q)$ we have:

$$\begin{cases} \mathcal{C}'_1(i) & = \emptyset \\ \mathcal{C}'_1(0) & = \emptyset \\ \mathcal{C}'_1(i,0) & = \emptyset \\ \mathcal{C}'_1(i,1) & = \mathcal{M}_i \\ \mathcal{C}'_1(i,j) & = \emptyset, \text{ for } 2 \leq j \leq p \end{cases}$$

- For each $b \in \mathcal{E}$, there are 2 cells labelled by l_b whose content is $\{\alpha_1\}$.

Hence, the result holds for $t = 1$.

By induction hypothesis, let t be such that $1 \leq t < p$, and let us suppose the result holds for t, that is,

- $\mathcal{C}'_t(i) = \emptyset$, for $0 \leq i \leq q$.
- For each $1 \leq i \leq q$, and $0 \leq j \leq p$ we have:

$$\mathcal{C}'_t(i,j) = \begin{cases} \mathcal{M}_i, \text{ if } & j = t \\ \emptyset, \text{ if } & j \neq t \end{cases}$$

- For each $b \in \mathcal{E}$, there exist 2^t cells labelled by l_b whose contents is $\mathcal{C}'_t(l_b) = \{\alpha_t\}$ (because $t \leq p-1$).

Then, at configuration $\mathcal{C}'_t$ only the following rules are applicable:

(1) If $t \leq p-2$, the rules $[\alpha_t]_{l_b} \rightarrow [\alpha_{t+1}]_{l_b}\ [\alpha_{t+1}]_{l_b}$, for each $b \in \mathcal{E}$.
(2) If $t = p-1$, the rules $[\alpha_{p-1}]_{l_b} \rightarrow [b]_{l_b}\ [b]_{l_b}$, for each $b \in \mathcal{E}$.
(3) $\big((i,t),\ a/\lambda\ , (i,t+1)\big)$, for each $a \in \Gamma$.

From the application of rules of types (1) or (2) at configuration $\mathcal{C}'_t$ we deduce that there are 2^{t+1} cells labelled by l_b in $\mathcal{C}'_{t+1}$, for each $b \in \mathcal{E}$, whose content is $\{\alpha_{t+1}\}$, if $t \leq p-2$, or $\{b\}$, if $t = p-1$.

¿From the application of rules of type (3) at configuration $\mathcal{C}'_t$, we deduce that

$$\mathcal{C}'_{t+1}(i,j) = \begin{cases} \mathcal{M}_i, \text{ if } & j = t+1 \\ \emptyset, \text{ if } & 0 \leq j \leq p \ \wedge\ j \neq t+1 \end{cases}$$

Bearing in mind that no other rule of system $\mathbf{S}(\Pi(n))$ is applicable, we deduce that $\mathcal{C}'_{t+1}(i) = \emptyset$, for $0 \leq i \leq q$.

This completes the proof of this Lemma.

□

Lemma 6.4 *Let $\mathcal{C}' = (\mathcal{C}'_0, \mathcal{C}'_1, \dots)$ be a computation of the tissue P system $\mathbf{S}(\Pi(n))$. Configuration $\mathcal{C}'_{p+1}$ is the following:*

(1) $\mathcal{C}'_{p+1}(0) = b_1^{2^p} \dots b_m^{2^p}$, *where* $\mathcal{E} = \{b_1, \dots, b_m\}$.
(2) $\mathcal{C}'_{p+1}(i) = \mathcal{M}_i = C_0(i)$, *for* $1 \leq i \leq q$.
(3) $\mathcal{C}'_{p+1}(i,j) = \emptyset$, *for* $1 \leq i \leq q$, $0 \leq j \leq p$.
(4) *There exist* 2^p *cells labelled by* l_b *whose content is empty, for* $b \in \mathcal{E}$.

Proof: From Lemma 6.3, the configuration $\mathcal{C}'_p$ is the following:

- $\mathcal{C}'_p(i) = \emptyset$, for $0 \leq i \leq q$.
- For each i $(1 \leq i \leq q)$ we have

$$\mathcal{C}'_p(i,j) = \begin{cases} \mathcal{M}_i, \text{ if } & j = p \\ \emptyset, \text{ if } & j \neq p \end{cases}$$

- For each $b \in \mathcal{E}$, there exist 2^p cells labelled by l_b whose content is $\{b\}$.

At configuration $\mathcal{C}'_p$ only the following rules are applicables:

- $\big((i,p),\ a/\lambda\ ,i\big)$, for each $a \in \Gamma \cap supp(\mathcal{M}_i)$.
- $\big(l_b,\ b/\lambda\ ,0\big)$, for each $b \in \mathcal{E}$.

Thus,

- $\mathcal{C}'_{p+1}(0) = b_1^{2^p} \dots b_m^{2^p}$, where $\mathcal{E} = \{b_1, \dots, b_m\}$.
- $\mathcal{C}'_{p+1}(i) = \mathcal{M}_i = C_0(i)$, for $1 \leq i \leq q$.
- $\mathcal{C}'_{p+1}(i,j) = \emptyset$, for $1 \leq i \leq q$ and $0 \leq j \leq p$.
- There exist 2^p cells labelled by l_b whose content is empty, for each $b \in \mathcal{E}$.

□

Definition 6.5 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_r)$ be a halting computation of $\Pi(n)$. Then we define the computation $\mathbf{S}(\mathcal{C}) = (\mathcal{C}'_0, \mathcal{C}'_1, \dots, \mathcal{C}'_p, \mathcal{C}'_{p+1}, \dots, \mathcal{C}'_{p+1+r})$ of $\mathbf{S}(\Pi(n))$ as follows:*

(1) *The initial configuration is:*
$$\begin{cases} \mathcal{C}'_0(i) = \emptyset, \text{ for } 0 \leq i \leq q \\ \mathcal{C}'_0(i,0) = C_0(i), \text{ for } 1 \leq i \leq q \\ \mathcal{C}'_0(i,j) = \emptyset, \text{ for } 1 \leq i \leq q \text{ and } 1 \leq j \leq p \\ \mathcal{C}'_0(l_b) = \alpha_0, \text{ for each } b \in \mathcal{E} \end{cases}$$
(2) *The configuration $\mathcal{C}'_t$, for $1 \leq t \leq p$, is described by Lemma 6.3.*
(3) *The configuration $\mathcal{C}'_{p+1}$ is described by Lemma 6.4.*
(4) *The configuration $\mathcal{C}'_{p+1+s}$, for $0 \leq s \leq r$, coincides with the configuration $\mathcal{C}_s$ of Π, that is, $\mathcal{C}_s(i) = \mathcal{C}'_{p+1+s}(i)$, for $1 \leq i \leq q$. The content of the remaining cells (excluding cell 0) at configuration $\mathcal{C}'_{p+1+s}$ is equal to the content of that cell at configuration $\mathcal{C}'_{p+1}$, that is, these cells do not evolve after step $p+1$.*

That is, every computation $\mathcal{C}$ of $\Pi(n)$ can be "reproduced" by a computation $\mathbf{S}(\mathcal{C})$ of $\mathbf{S}(\Pi(n))$ with a delay: from step $p+1$ to step $p+1+r$ the computation $\mathbf{S}(\mathcal{C})$ restricted to cells $1, \dots, q$ provides the computation $\mathcal{C}$ of $\Pi(n)$.

¿From Lemma 6.3 and Lemma 6.4 we deduce that: (a) $\mathbf{S}(\mathcal{C})$ is a computation of $\mathbf{S}(\Pi(n))$, and (b) S is an injective function from $\mathbf{Comp}(\Pi(n))$ onto $\mathbf{Comp}(S(\Pi(n)))$. Moreover, if p is a polynomial function on the size of $\Pi(n)$, then we have the following:

Proposition 6.6 *The tissue P system $\mathbf{S}(\Pi(n))$ defined in 6.2 simulates $\Pi(n)$ in an efficient way.*

Proof. In order to show that $\mathbf{S}(\Pi(n))$ can be constructed from $\Pi(n)$ by a deterministic Turing machine working in polynomial time, it is enough to note that the amount of resources needed to construct $\mathbf{S}(\Pi(n))$ from $\Pi(n)$ is polynomial in the size of the initial resources of $\Pi(n)$. Indeed,

1. The size of the alphabet of $\mathbf{S}(\Pi(n))$ is $|\Gamma'| = |\Gamma| + p$.

2. The initial number of cells of $\mathbf{S}(\Pi(n))$ is $1 + q \cdot (p + 2) + |\mathcal{E}|$.
3. The initial number of objects of $\mathbf{S}(\Pi(n))$ is the initial number of objects of $\Pi(n)$ plus $|\mathcal{E}|$.
4. The number of rules of $\mathbf{S}(\Pi(n))$ is $|\mathcal{R}'| = |\mathcal{R}| + (p + 1) \cdot |\mathcal{E}| + |\Gamma| \cdot q \cdot (p + 1)$.
5. The maximal length of a communication rule of $\mathbf{S}(\Pi(n))$ is equal to the maximal length of a communication rule of $\Pi(n)$.

¿From Lemma 6.3 and Lemma 6.4 we deduce that: (a) every computation $\mathcal{C}'$ of $\mathbf{S}(\Pi(n))$ has associated a computation $\mathcal{C}$ of $\Pi(n)$ such that $\mathbf{S}(\mathcal{C}) = \mathcal{C}'$ in a natural way, (b) the function S is injective, and (c) a computation $\mathcal{C}$ of Π is an accepting computation if and only if $\mathbf{S}(\mathcal{C})$ is an accepting computation of $\mathbf{S}(\Pi(n))$.

Finally, let us notice that if $\mathcal{C}$ is a computation of $\Pi(n)$ with length r, then $\mathbf{S}(\mathcal{C})$ is a computation of $\mathbf{S}(\Pi(n))$ with length $p + 1 + r$.

7 Computational Complexity classes of Tissue P Systems with Cell Division and without environment

In this Section, we analyze the role of the environment in the efficiency of tissue P systems with cell division. That is, we study the ability of these P systems with respect to the computational efficiency when the alphabet of the environment is an empty set.

Theorem 7.1 *For each $k \in \mathbb{N}$ we have* $\mathbf{PMC}_{\mathbf{TDC}(k+1)} = \mathbf{PMC}_{\widehat{\mathbf{TDC}}(k+1)}$.

Proof: Obviously, $\mathbf{P} \subseteq \mathbf{PMC}_{\widehat{\mathbf{TDC}}(1)} \subseteq \mathbf{PMC}_{\mathbf{TDC}(1)} = \mathbf{P}$.

Let $k \geq 1$. Since $\widehat{\mathbf{TDC}}(k+1) \subseteq \mathbf{TDC}(k+1)$ it suffices to show that $\mathbf{PMC}_{\mathbf{TDC}(k+1)} \subseteq \mathbf{PMC}_{\widehat{\mathbf{TDC}}(k+1)}$. For that, let $X \in \mathbf{PMC}_{TDC(k+1)}$. Let us show that $X \in \mathbf{PMC}_{\widehat{TDC}(k+1)}$.

Let $\{\Pi(n) : n \in N\}$ be a family of tissue P systems from $\mathbf{TDC}(k+1)$ solving X according to Definition 5.1. Let (cod, s) be a polinomial encoding associated with that solution. Let $u \in I_X$ be an instance of the problem X and $s(u) = n$. Then, that instance will be processed by the system $\Pi(s(u)) + cod(u)$. According to Lemma 5.2, let $p(n)$ be a polynomial function such that $2^{p(|u|)}$ is an upper bound of the number of objects from $\mathcal{E}$ which are moved from the environment to all cells of the system by any computation of $\Pi(s(u)) + cod(u)$, for each instance $u \in I_X$.

If $\Pi(s(u)) + cod(u) = (\Gamma, \Sigma, \mathcal{M}_1, \ldots, \mathcal{M}_{i_{in}} + cod(u), \ldots \mathcal{M}_q, \mathcal{E}, \mathcal{R}, i_{in}, i_{out})$, we consider the tissue P system without environment

$$\mathbf{S}(\Pi(s(u))) + cod(u) = (\Gamma', \Sigma', \mathcal{M}'_0, \mathcal{M}'_1, \ldots, \mathcal{M}'_{i_{in}} + cod(u), \ldots \mathcal{M}'_{q_1}, \mathcal{R}', i'_{in}, i'_{out})$$

according to Definition 6.2, where $q_1 = 1 + q \cdot (p(|u|) + 2) + |\mathcal{E}|$.

Therefore, $\mathbf{S}(\Pi(s(u))) + cod(u) \in \widehat{TDC}(k+1)$ and in the system $\mathbf{S}(\Pi(s(u))) + cod(u)$ the following holds:

- A new distinguished cell labelled by 0 has been considered, which will play the role of the environment at the system $\Pi(s(u)) + cod(u)$.
- We must guarantiee that system $\mathbf{S}(\Pi(s(u))) + cod(u)$ has initially enough objects in cell 0 to simulate the behaviour of the environment of $\Pi(n)$.
- New objects, new rules and new cells will be introduced in $\mathbf{S}(\Pi(s(u))) + cod(u)$.
- After $p(n) + 1$ step, computations of $\mathbf{S}(\Pi(s(u))) + cod(u)$ reproduce the computations of $\Pi(s(u)) + cod(u)$ exactly.

Let us suppose that $\mathcal{E} = \{b_1, \ldots, b_m\}$. In order to simulate $\Pi(s(u)) + cod(u)$ by a tissue P system without environment in an efficient way, we need to have enough objects in the cell of $\mathbf{S}(\Pi(s(u))) + cod(u)$ labelled by 0 available. That is, $2^{p(n)}$ objects in that cell are enough.

In order to start the simulation of any computation $\mathcal{C}$ of $\Pi(s(u)) + cod(u)$, it would be enough to have $2^{p(n)}$ copies of each object $b_j \in \mathcal{E}$ in the cell of $\mathbf{S}(\Pi(s(u))) + cod(u)$ labelled by 0. For this purpose

- For each $b \in \mathcal{E}$ we consider a cell in $\mathbf{S}(\Pi(s(u))) + cod(u)$ labelled by l_b which only contains object α_0 initially. We also consider the following rules:
  - $[\alpha_j]_{l_b} \rightarrow [\alpha_{j+1}]_{l_b}\ [\alpha_{j+1}]_{l_b}$, for $0 \leq j \leq p(n) - 2$.
  - $[\alpha_{p(n)-1}]_{l_b} \rightarrow [b]_{l_b}\ [b]_{l_b}$.
  - $(l_b, b/\lambda, 0)$.
- By applying the previous rules, after $p(n)$ transition steps we get $2^{p(n)}$ cells labelled by l_b, for each $b \in \mathcal{E}$ in such a way that each of them contains only object b. Finally, by applying the third rule we get $2^{p(n)}$ copies of objects b in cell 0, for each $b \in \mathcal{E}$.

Therefore, after the execution of $p(n) + 1$ transition steps in each computation of $\mathbf{S}(\Pi(s(u))) + cod(u)$ in cell 0 of the corresponding configuration, we have $2^{p(n)}$ copies of each object $b_1, \ldots, b_m \in \mathcal{E}$. This number of copies is enough to simulate any computation $\mathcal{C}$ of $\Pi(s(u)) + cod(u)$ through the system $\mathbf{S}(\Pi(s(u)) + cod(u))$.

¿From Proposition 6.6 we deduce that the family $\{\mathbf{S}(\Pi(n))|\ n \in N\}$ solves X in polynomial time according to Definition 5.1. Hence, $X \in \mathbf{PMC}_{\widehat{TDC}(k+1)}$.

□

8 Conclusions and Further Works

The efficiency of cell-like P systems for solving **NP**-complete problems has been widely studied. The space-time tradeoff method is used to efficiently solve **NP**-complete problems in the framework of *Membrane Computing*. Membrane division, membrane creation, and membrane separation are three efficient ways to obtain exponential workspace in polynomial time. Cell division were introduced [5] into tissue–like P systems, and a linear time solution for SAT problem by tissue P systems with cell division was given [5].

In the framework of tissue P systems, there is an additional advantage when cell division is used to generate exponential workspace in polynomial time: all the

other objects in the cell are duplicated except the object that activate the cell division operation.

In this paper, the computational efficiency of tissue P systems with cell division and *without environment* has been studied. We conclude that the environment of tissue P systems can be removed without a loss of efficiency.

For future work, we plan to do further research in the study of tissue P systems with cell separation. Let us recall that, in this kind of systems, the application of separation rules only duplicates the cell while the objects are not replicated. They are simply distributed according to a prefixed criterion.

Acknowledgements

The work was supported by Project TIN2009-13192 of the Ministerio de Ciencia e Innovación of Spain and Project of Excellence with *Investigador de Reconocida Valía*, from Junta de Andalucía, grant P08 – TIC 04200.

References

1. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J. and Romero-Campero, F.J. A linear solution for QSAT with Membrane Creation. *Lecture Notes in Computer Science* **3850**, (2006), 241–252.
2. Pan, L. and Ishdorj, T.-O. P systems with active membranes and separation rules. *Journal of Universal Computer Science*, **10**, 5, (2004), 630–649.
3. Păun, Gh. Attacking **NP**-complete problems. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou, C. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, 94-115.
4. Păun, Gh. *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, (2002).
5. Păun, Gh., Pérez-Jiménez, M.J. and Riscos-Núñez, A. Tissue P System with cell division. *In. J. of Computers, communications & control*, **3**, 3, (2008), 295–303.
6. Pérez-Jiménez, M.J., Romero-Jiménez, A. and Sancho-Caparrini, F. Complexity classes in models of cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265–285.
7. Pérez-Jiménez, M.J., Romero-Jiménez, A. and Sancho-Caparrini, F. A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics*, **11**, 4, (2006), 423-434.

Improving the Efficiency of Tissue P Systems with Cell Separation

Mario J. Pérez-Jiménez[1], Petr Sosík[2,3]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, 41012 Sevilla, Spain
marper@us.es

[2] Departamento de Inteligencia Artificial, Facultad de Informática,
Universidad Politécnica de Madrid, Campus de Montegancedo s/n,
Boadilla del Monte, 28660 Madrid, Spain

[3] Research Institute of the IT4Innovations Centre of Excellence,
Faculty of Philosophy and Science, Silesian University in Opava,
74601 Opava, Czech Republic
psosik@fi.upm.es

Summary. Cell fission process consists of the division of a cell into two new cells such that the contents of the initial cell is distributed between the newly created cells. This process is modelled by a new kind of *cell separation rules* in the framework of *Membrane Computing*. Specifically, in tissue-like membrane systems, cell separation rules have been considered joint with communication rules of the form symport/antiport. These models are able to create an exponential workspace, expressed in terms of the number of cells, in linear time. On the one hand, an efficient and uniform solution to the SAT problem by using cell separation and communication rules with length at most 8 has been recently given. On the other hand, only tractable problems can be efficiently solved by using cell separation and communication rules with length at most 1. Thus, in the framework of tissue P systems with cell separation, and assuming that $\mathbf{P} \neq \mathbf{NP}$, a first frontier between efficiency and non-efficiency is obtained when passing from communication rules with length 1 to communication rules with length at most 8.

In this paper we improve the previous result by showing that the SAT problem can be solved by a family of tissue P systems with cell separation in linear time, by using communication rules with length at most 3. Hence, we provide a new tractability borderline: passing from 1 to 3 amounts to passing from non–efficiency to efficiency, assuming that $\mathbf{P} \neq \mathbf{NP}$.

1 Introduction

Membrane Computing is a young branch of *Natural Computing* initiated by Gh. Păun in the end of 1998 [16]. It is inspired by the structure and functioning of

living cell, as well as from the organization of cells in tissues, organs, and other higher order structures. The devices of this paradigm, called *P systems*, provide models for distributed, parallel and non-deterministic computing.

Membrane Computing has received an important attention from the scientific community since then, and many applications have been reported ([3], [21]). It was selected by the Institute for Scientific Information, USA, as a fast *Emerging Research Front in Computer Science*, and [19] was mentioned in [25] as a highly cited paper in October 2003.

Roughly speaking, the main ingredient of a membrane system is a cell-like *membrane structure* (a rooted tree), in the *compartments* of which one places *multisets* of *symbol-objects*. The objects evolve in a synchronous maximally parallel manner according to given *evolution rules*, also associated with the membranes (for introduction see [18] and for further bibliography see [26]).

Several different models of cell-like P systems have been successfully used to solve computationally hard problems efficiently, by trading space for time: an exponential workspace is created in polynomial time by using some kind of rules, and then massive parallelism is used to simultaneously check all the candidate solutions. Inspired by living cell, several ways for obtaining exponential workspace in polynomial time were proposed: membrane division (*mitosis*) [17], membrane creation (*autopoiesis*) [9], and membrane separation (*membrane fission*) [14]. These three ways have given rise to the following models: *P systems with active membranes*, *P systems with membrane creation*, and *P systems with membranes separation*.

A new type of P systems, the so-called *tissue P systems*, was considered in [12]. Instead of considering a hierarchical arrangement, membranes/cells are placed in the nodes of a virtual graph. This variant has two biological justifications (see [13]): intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. The communication among cells is based on symport/antiport rules, which were introduced to P systems in [19]. Symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite directions. From the seminal definitions of tissue P systems [12, 13], several research lines have been developed and other variants have arisen (see, for example, [1, 2, 6, 10, 11, 24]). One of the most interesting variants of tissue P systems was presented in [20], where the definition of tissue P systems is combined with the one of P systems with active membranes, yielding *tissue P systems with cell division*. In this kind of models [20], there exists cell replication, that is, the two new cells generated by a division rule have exactly the same objects except for at most a pair of different objects.

In the biological phenomenon of fission, the contents of the two new cells evolved from a cell can be significantly different, and membrane separation inspired by this biological phenomenon in the framework of cell-like P systems was proved to be an efficient way to obtain exponential workspace in polynomial time

[14]. In [15], a new class of tissue P systems based on cell fission, called *tissue P systems with cell separation*, was presented. Its computational efficiency was investigated, and two important results were obtained: (a) only tractable problems can be efficiently solved by using cell separation and communication rules with length at most 1, and (b) an efficient (uniform) solution to the SAT problem by using cell separation and communication rules with length at most 8 was presented. Hence, in the framework of recognizer tissue P systems with cell separation, the length of the communication rules provide a borderline between efficiency and non-efficiency, that is, a frontier is there when we pass from length 1 to length 6, assuming that $\mathbf{P} \neq \mathbf{NP}$.

In this paper we present an improvement of the previous borderline of the tractability. Specifically, we propose a (uniform) family of tissue P systems with cell separation and communication rules with length at most 3 which solves the SAT problem in linear time. Hence, a new borderline is provided in this paper: passing from 1 to 3 amounts to passing from non–efficiency to efficiency, assuming that $\mathbf{P} \neq \mathbf{NP}$.

The paper is organized as follows: first, we recall some preliminaries, and then, the definition of tissue P systems with cell separation is given. Next, recognizer tissue P systems and computational complexity classes in this framework, are briefly described. In Section 5, an efficient (uniform) solution to the SAT problem by using cell separation and communication rules with length at most 3 is shown. Section 6 is devoted to present a detailed formal verification of the main result. Finally, conclusions and further works are presented.

2 Preliminaries

An *alphabet*, Σ, is a non–empty set whose elements are called *symbols*. An ordered finite sequence of symbols is a *string* o *word*. If u and v are strings over Σ, then so is their *concatenation* uv, obtained by juxtaposition, that is, writing u and v after one another. The number of symbols in a string u is the *length* of the string, and it is denoted by $|u|$. As usual, the empty string (with length 0) will be denoted by λ. The set of all strings over an alphabet Σ is denoted by Σ^*. In algebraic terms, Σ^* is the free monoid generated by Σ under the operation of concatenation. Subsets, finite or infinite, of Σ^* are referred to as *languages* over Σ.

The *Parikh vector* associated with a string $u \in \Sigma^*$ with respect to the alphabet $\Sigma = \{a_1, \ldots, a_r\}$ is $\Psi_\Sigma(u) = (|u|_{a_1}, \ldots, |u|_{a_r})$, where $|u|_{a_i}$ denotes the number of ocurrences of the symbol a_i in the string u. This is called the *Parikh mapping* associated with Σ. Notice that in this definition the ordering of the symbols from Σ is relevant. If $\Sigma_1 = \{a_{i_1}, \ldots, a_{i_s}\} \subseteq \Sigma$ then we define $\Psi_{\Sigma_1}(u) = (|u|_{a_{i_1}}, \ldots, |u|_{a_{i_s}})$, for each $u \in \Sigma^*$.

A *multiset* m over a set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in A \mid f(x) > 0\}$. A multiset is empty (resp. finite) if its support is the empty set (resp. a finite set). If

$m = (A, f)$ is a finite multiset over A, and $supp(m) = \{a_1, \ldots, a_k\}$ then it will be denoted as $m = \{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}$. That is, superscripts indicate the multiplicity of each element, and if $f(x) = 0$ for $x \in A$, then the element x is omitted. A finite multiset $m = \{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}$ can also be represented by the string $a_1^{f(a_1)} \ldots a_k^{f(a_k)}$ over the alphabet $\{a_1, \ldots, a_k\}$. Nevertheless, all permutations of this string precisely identify the same multiset m. Throughout this paper, we speak about "the finite multiset m" where m is a string, and meaning "the finite multiset represented by the string m".

If $m_1 = (A, f_1)$, $m_2 = (A, f_2)$ are multisets over A, then we define the union of m_1 and m_2 as $m_1 + m_2 = (A, g)$, where $g = f_1 + f_2$.

For any sets A and B the *relative complement* $A \setminus B$ of B in A is defined as follows:

$$A \setminus B = \{x \in A \mid \ x \notin B\}$$

In what follows, we assume the reader is already familiar with the basic notions and the terminology of P systems. For details, see [18].

3 Tissue P Systems with Cell Separation

Let us recall that the model of *tissue P systems with cell separation* is based on the cell-like model of P systems with membranes separation [14]. The biological inspiration is the following: alive tissues are not *static* network of cells, since new cells are generated by membrane fission in a natural way. In these models, the cells are not polarized; the two cells obtained by separation have the same labels as the original cell, and if a cell is separated, its interaction with other cells or with the environment is blocked during the separation process. In some sense, this means that while a cell is separating it closes its communication channels.

Definition 3.1 *A tissue P system with cell separation of degree $q \geq 1$ is a tuple*

$$\Pi = (\Gamma, \Gamma_1, \Gamma_2, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out}),$$

where:

1. *Γ is a finite* alphabet *whose elements are called* objects*;*
2. *$\{\Gamma_1, \Gamma_2\}$ is a partition of Γ, that is, $\Gamma = \Gamma_1 \cup \Gamma_2$, $\Gamma_1, \Gamma_2 \neq \emptyset$, $\Gamma_1 \cap \Gamma_2 = \emptyset$;*
3. *$\mathcal{E} \subseteq \Gamma$ is a finite alphabet representing the set of objects initially in the environment of the system, and 0 is the label of the environment (the environment is not properly a cell of the system); let us assume that objects in the environment appear in arbitrary copies each;*
4. *$\mathcal{M}_1, \ldots, \mathcal{M}_q$ are strings over Γ, representing the finite multisets of objects placed in the q cells of the system at the beginning of the computation; $1, 2, \cdots, q$ are labels which identify the cells of the system;*
5. *$\mathcal{R}$ is a finite set of rules of the following forms:*

   (a) Communication rules: $(i, u/v, j)$, *for* $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$, $|uv| > 0$. *When applying a rule* $(i, u/v, j)$, *the objects of the multiset represented by* u *are sent from region* i *to region* j *and, simultaneously, the objects of the multiset* v *are sent from region* j *to region* i;
   (b) Separation rules: $[a]_i \rightarrow [\Gamma_1]_i[\Gamma_2]_i$, *where* $i \in \{1, 2, \ldots, q\}$ *and* $a \in \Gamma$, *and* $i \neq i_{out}$. *In reaction with an object* a, *the cell* i *is separated into two cells with the same label; at the same time, object* a *is consumed; the objects from* Γ_1 *are placed in the first cell, those from* Γ_2 *are placed in the second cell; the output cell* i_{out} *cannot be separated;*
6. $i_{out} \in \{0, 1, 2, \ldots, q\}$ *is the output cell.*

A communication rule $(i, u/v, j)$ is called a *symport rule* if $u = \lambda$ or $v = \lambda$. A symport rule $(i, u/\lambda, j)$, with $i \neq 0, j \neq 0$, provides a virtual arc from cell i to cell j. A communication rule $(i, u/v, j)$ is called an *antiport rule* if $u \neq \lambda$ and $v \neq \lambda$. An antiport rule $(i, u/v, j)$, with $i \neq 0, j \neq 0$, provides two arcs: one from cell i to cell j and another one from cell j to cell i. Thus, every tissue P systems has an underlying directed graph whose nodes are the cells of the system and the arcs are obtained from communication rules. In this context, the environment can be considered as a virtual node of the graph such that their connections are defined by the communication rules of the form $(i, u/v, j)$, with $i = 0$ or $j = 0$.

The length of the communication rule $(i, u/v, j)$ is defined as $|u| + |v|$.

The rules of a system like the above one are used in the non-deterministic maximally parallel manner as customary in Membrane Computing. At each step, all cells which can evolve must evolve in a maximally parallel way (at each step we apply a multiset of rules which is maximal, no further rule can be added being applicable). This way of applying rules has only one restriction: when a cell is separated, the separation rule is the only one which is applied for that cell at that step; thus, the objects inside that cell do not evolve by means of communication rules. The new cells resulting from separation could participate in the interaction with other cells or the environment by means of communication rules at the next step – providing that they are not separated once again. The label of a cell precisely identify the rules which can be applied to it.

An *instanstaneous description* or a *configuration* at any instant of a tissue P system with cell separation is described by all multisets of objects over Γ associated with all the cells present in the system, and the multiset of objects over $\Gamma - \mathcal{E}$ associated with the environment at that moment. Bearing in mind the objects from $\mathcal{E}$ have infinite copies in the environment, they are not properly changed along the computation. The *initial configuration* is $(\mathcal{M}_1, \cdots, \mathcal{M}_q; \emptyset)$. A configuration is a *halting configuration* if no rule of the system is applicable to it.

Let us fix a tissue P system with cell separation Π. We say that configuration C_1 yields configuration C_2 in one *transition step*, denoted $C_1 \Rightarrow_\Pi C_2$, if we can pass from C_1 to C_2 by applying the rules from $\mathcal{R}$ following the previous remarks. A *computation* of Π is a (finite or infinite) sequence of configurations such that:

1. the first term of the sequence is the initial configuration of the system;

2. each non-initial configuration of the sequence is obtained from the previous configuration by aplying rules of the system in a maximally parallel manner with the restrictions previously mentioned; and
3. if the sequence is finite (called *halting computation*) then the last term of the sequence is a halting configuration.

All computations start from an initial configuration and proceed as stated above; only halting computations give a result, which is encoded by the objects present in the output cell i_{out} in the halting configuration.

We denote by $\mathbf{Comp}(\Pi)$ the set of computations of the tissue P system Π. If $\mathcal{C} = \{\mathcal{C}_i\}_{i<r+1}$ of Π ($r \in \mathbf{N}$) is a halting computation, then the length of $\mathcal{C}$ is r, that is, the number of non-initial configurations which appear in the finite sequence $\mathcal{C}$. We denote it by $|\mathcal{C}|$. We also denote by $\mathcal{C}_i(j)$ the contents of cell j at the configuration $\mathcal{C}_i$.

In the framework of tissue P systems with symport/antiport rules, it is interesting to highlight some differences between a division rule of the type $[a]_i \rightarrow [b]_i\ [c]_i$, and a separation rule of the type $[a]_i \rightarrow [\Gamma_1]_i\ [\Gamma_2]_i$:

1. The object a triggers both rules and it is consumed. Nevertheless,
   - ⋆ *Division rule*: Produces an object (b or c) in each new cell.
   - ⋆ *Separation rule*: Does not produce any new object in new cells.
2. The remaining objects in cell i:
   - ⋆ *Division rule*: Are replicated in each new cell.
   - ⋆ *Separation rule*: Are distributed between the new cells, according to sets Γ_1 and Γ_2.
3. If there is n objects in the cell i where the rule is applied:
   - ⋆ *Division rule*: The total number of objects in the cells created is $2n$, each of them contains n objects.
   - ⋆ *Separation rule*: The total number of objects in the cells created is $n - 1$.
4. If the rules are consecutively applied during k transtition steps in a cell i which contains n objects:
   - ⋆ *Division rule*: 2^k new cells are created, and the total number of objects is $n \cdot 2^k$.
   - ⋆ *Separation rule*: $2 \cdot k$ new cells are created, and the total number of objects is $n - k$.

Hence, division and separation rules have the ability to produce an exponential number of new cells in linear time, but only division rules are able to simultaneously produce an exponential number of objects.

3.1 Recognizer Tissue P Systems with Cell Separation

Let us recall that a *decision problem* is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X. Many abstract problems are not decision problems, for example, in *combinatorial optimization problems* some value must be optimized (minimized

or maximized). In order to deal with such problems, they can be transformed into roughly equivalent decision problems by supplying a target/threshold value for the quantity to be optimized, and then asking whether this value can be attained.

A natural correspondence between decision problems and languages over a finite alphabet, can be established as follows. Given a decision problem $X = (I_X, \theta_X)$, its associated language is $L_X = \{w \in I_X : \theta_X(w) = 1\}$. Conversely, given a language L over an alphabet Σ, its associated decision problem is $X_L = (I_{X_L}, \theta_{X_L})$, where $I_{X_L} = \Sigma^*$, and $\theta_{X_L} = \{(x,1) : x \in L\} \cup \{(x,0) : x \notin L\}$. The solvability of decision problems is defined through the recognition of the languages associated with them, by using languages recognizer devices.

In order to study the computational efficiency of membrane systems, the notions from classical *computational complexity theory* are adapted for Membrane Computing, and a special class of cell-like P systems is introduced in [23]: *recognizer P systems* (called *accepting P systems* in a previous paper [22]). For tissue P systems, with the same idea as recognizer cell-like P systems, *recognizer tissue P systems* is introduced in [20].

Definition 3.2 *A* recognizer tissue P system with cell separation *of degree* $q \geq 1$ *is a tuple*

$$\Pi = (\Gamma, \Gamma_1, \Gamma_2, \Sigma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$$

where:

1. $(\Gamma, \Gamma_1, \Gamma_2, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$ *is a tissue P system with cell separation of degree* $q \geq 1$ *(as defined in the previous section).*
2. *The working alphabet* Γ *has two distinguished objects* `yes` *and* `no` *being, at least, one copy of them present in some initial multisets* $\mathcal{M}_1, \ldots, \mathcal{M}_q$*, but none of them are present in* $\mathcal{E}$*.*
3. Σ *is an (input) alphabet strictly contained in* Γ*, and* $\mathcal{E} \subseteq \Gamma \setminus \Sigma$*.*
4. $\mathcal{M}_1, \ldots, \mathcal{M}_q$ *are strings over* $\Gamma \setminus \Sigma$*;*
5. $i_{in} \in \{1, \ldots, q\}$ *is the input cell.*
6. *The output region* i_{out} *is the environment.*
7. *All computations halt.*
8. *If* $\mathcal{C}$ *is a computation of* Π*, then either object* `yes` *or object* `no` *(but not both) must have been released into the environment, and only at the last step of the computation.*

For each $w \in \Sigma^*$, the *computation of the system* Π *with input* $w \in \Sigma^*$ starts from the configuration of the form $(\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_{i_{in}} + w, \ldots, \mathcal{M}_q; \emptyset)$, that is, the input multiset w has been added to the contents of the input cell i_{in}. Therefore, we have an initial configuration associated with each input multiset w (over the input alphabet Σ) in this kind of systems.

Given a recognizer tissue P system with cell division, and a halting computation $\mathcal{C} = \{C_i\}_{i<r+1}$ of Π ($r \in \mathbf{N}$), we define the result of $\mathcal{C}$ as follows:

$$Output(\mathcal{C}) = \begin{cases} \texttt{yes}, & \text{if } \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{r,0}) = (1,0) \wedge \\ & \quad \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{i,0}) = (0,0) \text{ for } i = 0,\ldots,r-1 \\ \texttt{no}, & \text{if } \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{r,0}) = (0,1) \wedge \\ & \quad \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{i,0}) = (0,0) \text{ for } i = 0,\ldots,r-1 \end{cases}$$

where Ψ is the Parikh function, and $M_{i,0}$ is the multiset over $\Gamma \setminus \mathcal{E}$ associated with the environment at configuration C_i, in particular, $M_{r,0}$ is the multiset over $\Gamma \setminus \mathcal{E}$ associated with the environment at the halting configuration C_r.

We say that a computation $\mathcal{C}$ is an *accepting computation* (respectively, *rejecting computation*) if $Output(\mathcal{C}) = \texttt{yes}$ (respectively, $Output(\mathcal{C}) = \texttt{no}$), that is, if object `yes` (respectively, object `no`) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$, and neither object `yes` nor `no` appears in the environment associated with any non–halting configuration of $\mathcal{C}$.

For each natural number $k \geq 1$, we denote by $\mathbf{TSC}(k)$ the class of recognizer tissue P systems with cell separation and communication rules of length at most k. We denote by $\mathbf{TSC}$ the class of recognizer tissue P systems with cell separation and without restriction on the length of communication rules. Obviously, $\mathbf{TSC}(k) \subseteq \mathbf{TSC}$ for all $k \geq 1$.

3.2 Polynomial Complexity Classes of Tissue P systems with Cell Separation

Next, we define what means solving a decision problem in the framework of tissue P systems efficiently and in a uniform way. Bearing in mind that they provide devices with a finite description, a numerable family of tissue P systems will be necessary in order to solve a decision problem.

Definition 1. *We say that a decision problem $X = (I_X, \theta_X)$ is solvable in a uniform way and polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of recognizer tissue P systems with cell separation if the following holds:*

1. *The family $\mathbf{\Pi}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$.*
2. *There exists a pair (cod, s) of polynomial-time computable functions over I_X such that:*
   (a) *for each instance $u \in I_X$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$;*
   (b) *for each $n \in \mathbb{N}$, $s^{-1}(n)$ is a finite set;*
   (c) *the family $\mathbf{\Pi}$ is* polynomially bounded *with regard to (X, cod, s), that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(s(u))$ with input $cod(u)$ is halting and it performs at most $p(|u|)$ steps;*
   (d) *the family $\mathbf{\Pi}$ is* sound *with regard to (X, cod, s), that is, for each $u \in I_X$, if there exists an accepting computation of $\Pi(s(u))$ with input $cod(u)$, then $\theta_X(u) = 1$;*

(e) the family $\mathbf{\Pi}$ *is* complete *with regard to* (X, cod, s)*, that is, for each* $u \in I_X$*, if* $\theta_X(u) = 1$*, then every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting one.*

From the soundness and completeness conditions above we deduce that every P system $\Pi(n)$ is *confluent*, in the following sense: every computation of a system with the *same* input multiset must always give the *same* answer.

Let $\mathbf{R}$ be a class of recognizer tissue P systems. We denote by $\mathbf{PMC_R}$ the set of all decision problems which can be solved in a uniform way and polynomial time by means of families of systems from $\mathbf{R}$.

4 Computational Efficiency of Tissue P Systems with Cell Separation

It is well known that tissue P systems with cell division are able to solve computationally hard problems efficiently. Specifically, **NP**–complete problems have been solved in linear time [5] by using families of tissue P systems with cell division and communication rules of length at most 3.

In [15] two important results related to the computational efficiency of tissue P systems with cell separation were obtained. On the one hand, only tractable problems can be efficiently solved by using families of tissue P systems with cell separation and communication rules of length 1, that is, $\mathbf{P} = \mathbf{PMC}_{TSC(1)}$. On the other hand, an efficient solution to the SAT problem has been given by means of a uniform family of tissue P systems with cell separation and communication rules of length at most 8, that is, $\mathtt{SAT} \in \mathbf{PMC}_{TSC(8)}$, hence $\mathbf{NP} \cup \mathbf{co-NP} \subseteq \mathbf{PMC}_{TSC(8)}$. Therefore, passing the maximum length of communication rules of the systems from 1 to 6 amounts to passing from non–efficiency to efficiency, assuming that $\mathbf{P} \neq \mathbf{NP}$. An interesting challenge is to refine that efficiency borderline, that is, to provide new efficient solutions to computationally hard problems by means of tissue P systems with cell separation by using communication with length under 6.

In the next Section, we improve the result from [15] by giving a family of tissue P systems with cell separation and communication rules of length at most 3 which solves the SAT problem in linear time.

5 Solving the SAT Problem by using TSC(3)

Let us recall that the SAT problem is the following: *given a boolean formula in conjunctive normal form (CNF), to determine whether or not there exists an assignment to its variables on which it evaluates true.* This is a well known **NP**-complete problem [7].

In this Section, we propose a solution following a brute force algorithm implemented in the framework of recognizer tissue P systems with cell separation. The solution consists of the following stages:

- *Generation Stage*: All truth assignments associated with the input formula are produced by using cell separation in an adequate way.
- *Checking Stage*: In each cell, it is checked whether or not the formula is satisfiable by the truth assignment encoded by that cell.
- *Output Stage*: The system sends to the environment the right answer according to the results of the previous stage.

Let us consider the polynomial–time computable function (the *pair function*)

$$\langle m,n\rangle = ((m+n)(m+n+1)/2)+m$$

which is also a primitive recursive and bijective function from $\mathbb{N}\times\mathbb{N}$ to $\mathbb{N}$.

Next, we define a family $\mathbf{\Pi} = \{\Pi(t) : t \in \mathbb{N}\}$ of recognizer tissue P system with cell separation from $\mathbf{TSC}(3)$, such that each system $\Pi(t)$ will process all instances φ of SAT with n variables and m clauses, where $t = \langle m,n\rangle$, provided that the appropriate input multiset $cod(\varphi)$ is supplied to the system.

For each $(m,n)\in\mathbb{N}\times\mathbb{N}$, we consider the recognizer tissue P system with cell separation from $\mathbf{TSC}(3)$,

$$\Pi(\langle m,n\rangle) = (\Gamma,\Gamma_1,\Gamma_2,\Sigma,\mathcal{E},\mathcal{M}_1,\mathcal{M}_2,\mathcal{M}_3,\mathcal{R},i_{in},i_{out})$$

defined as follows:

- The input alphabet is

$$\Sigma = \{x_{i,j},\overline{x}_{i,j} : 1\le i\le n\,,\, 1\le j\le m\}$$

- The working alphabet is $\Gamma = \Sigma \cup \Gamma_1 \cup \Gamma_2$, where:

$$\begin{aligned}\Gamma_1 = {} & \{A_i,B_i : 1\le i\le n+1\} \cup \{a_i,b_i,T_i,F_i,y_i,v_i,w_i : 1\le i\le n\} \cup\\ & \{c_i,t_i,f_i,s_i,z_i : 1\le i\le n-1\} \cup \{E_j : 1\le j\le m+1\} \cup\\ & \{\alpha_i : 0\le i\le 3n+2m+1\} \cup \{\beta_i : 0\le i\le 3n+2m+2\} \cup\\ & \{q_{i,j},r_{i,j},u_{i,j} : 1\le i,j\le n-1\} \cup\\ & \{x_{i,j},\overline{x}_{i,j},e_{i,j},\overline{e}_{i,j} : 1\le i\le n\,,\, 1\le j\le m\} \cup\\ & \{d_{i,j,k},\overline{d}_{i,j,k} : 1\le i\le n\,,\, 1\le j\le m\,,\, 1\le k\le n\} \cup \{q_0,S,\texttt{yes},\texttt{no}\}\end{aligned}$$

$$\Gamma_2 = \{A'_i,B'_i : 1\le i\le n+1\}\cup\{a'_i,b'_i,T'_i,F'_i : 1\le i\le n\}$$

- The alphabet of the environment is:

$$\begin{aligned}\mathcal{E} = {} & \{S\} \cup \{A_i,B_i,A'_i,B'_i : 2\le i\le n+1\} \cup \{T_i,F_i,F'_i,y_i,w_i : 1\le i\le n\} \cup\\ & \{a_i,a'_i,b_i,b'_i,v_i : 2\le i\le n\} \cup \{T'_i,c_i,t_i,f_i,s_i,z_i : 1\le i\le n-1\} \cup\\ & \{E_j : 1\le j\le m+1\} \cup \{\alpha_i : 1\le i\le 3n+2m+1\} \cup\\ & \{\beta_i : 1\le i\le 3n+2m+2\} \cup\\ & \{q_{i,j},r_{i,j},u_{i,j} : 1\le i\le n-1\,,\, 2\le j\le n-1\} \cup\\ & \{e_{i,j},\overline{e}_{i,j} : 1\le i\le n\,,\, 1\le j\le m\} \cup\\ & \{d_{i,j,k},\overline{d}_{i,j,k} : 1\le i,k\le n\,,\, 1\le j\le m\}\end{aligned}$$

- Initial multisets:

$$\mathcal{M}_1 = A_1\, B_1$$
$$\mathcal{M}_2 = a_1\, a'_1\, b_1\, b'_1\, v_1\, q_{1,1}\, \alpha_0\, \texttt{yes}\, \texttt{no}$$
$$\mathcal{M}_3 = \beta_0$$

- The set R of rules consists of the following rules:

(1) $(1\,,\, A_i \,/\, a_i\, a'_i\,,\, 2)$, for $1 \leq i \leq n$, and $(1\,,\, A_{n+1} \,/\, E_1\,,\, 2)$.
(2) $(1\,,\, A'_i \,/\, a_i\, a'_i\,,\, 2)$, for $1 \leq i \leq n$, and $(1\,,\, A'_{n+1} \,/\, E_1\,,\, 2)$.
(3) $(1\,,\, B_i \,/\, b_i\, b'_i\,,\, 2)$, for $1 \leq i \leq n$.
(4) $(1\,,\, B'_i \,/\, b_i\, b'_i\,,\, 2)$, for $1 \leq i \leq n$.
(5) $(1\,,\, T_i \,/\, t_i\,,\, 2)$, for $1 \leq i \leq n-1$.
(6) $(1\,,\, T'_i \,/\, t_i\,,\, 2)$, for $1 \leq i \leq n-1$.
(7) $(1\,,\, F_i \,/\, f_i\,,\, 2)$, for $1 \leq i \leq n-1$.
(8) $(1\,,\, F'_i \,/\, f_i\,,\, 2)$, for $1 \leq i \leq n-1$.
(9) $(1\,,\, t_i \,/\, T_i\, T'_i,\, 0)$, for $1 \leq i \leq n-1$.
(10) $(1\,,\, f_i \,/\, F_i\, F'_i,\, 0)$, for $1 \leq i \leq n-1$.
(11) $(1\,,\, b_i \,/\, B_{i+1}\, S\,,\, 0)$, for $1 \leq i \leq n$, and $(1\,,\, B_{n+1} \,/\, \lambda\,,\, 0)$.
(12) $(1\,,\, b'_i \,/\, B'_{i+1}\,,\, 0)$, for $1 \leq i \leq n$, and $(1\,,\, B'_{n+1} \,/\, \lambda\,,\, 0)$.
(13) $(1\,,\, a_i \,/\, T_i\, A_{i+1}\,,\, 0)$, for $1 \leq i \leq n$.
(14) $(1\,,\, a'_i \,/\, F'_i\, A'_{i+1}\,,\, 0)$, for $1 \leq i \leq n$.
(15) $(2\,,\, A_i \,/\, c_i\,,\, 0)$, for $1 \leq i \leq n-1$, and $(2\,,\, A_i \,/\, \lambda\,,\, 0)$, for $n \leq i \leq n+1$.
(16) $(2\,,\, A'_i \,/\, c_i\,,\, 0)$, for $1 \leq i \leq n-1$, and $(2\,,\, A'_i \,/\, \lambda\,,\, 0)$, for $n \leq i \leq n+1$.
(17) $(2\,,\, B_i \,/\, c_i\,,\, 0)$, for $1 \leq i \leq n-1$, and $(2\,,\, B_n \,/\, \lambda\,,\, 0)$.
(18) $(2\,,\, B'_i \,/\, c_i\,,\, 0)$, for $1 \leq i \leq n-1$, and $(2\,,\, B'_n \,/\, \lambda\,,\, 0)$.
(19) $(2\,,\, c_i \,/\, b_{i+1}\, b'_{i+1}\,,\, 0)$, for $1 \leq i \leq n-1$.
(20) $(2\,,\, v_i \,/\, y_i^2\,,\, 0)$, for $1 \leq i \leq n$.
(21) $(2\,,\, y_i \,/\, z_i\, w_i\,,\, 0)$, for $1 \leq i \leq n-1$, and $(2\,,\, y_n \,/\, w_n\,,\, 0)$.
(22) $(2\,,\, z_i \,/\, v_{i+1}\,,\, 0)$, for $1 \leq i \leq n-1$.
(23) $(2\,,\, w_i \,/\, a_{i+1}\, a'_{i+1}\,,\, 0)$, for $1 \leq i \leq n-1$, and $(2\,,\, w_n \,/\, E_1\,,\, 0)$.
(24) $(2\,,\, q_{1,1} \,/\, r_{1,1}\,,\, 0)$.
(25) $(2\,,\, q_{i,j} \,/\, r_{i,j}^2\,,\, 0)$, for $1 \leq i \leq n-1,\ 2 \leq j \leq n-1$.
(26) $(2\,,\, r_{i,j} \,/\, s_i\, u_{i,j}\,,\, 0)$, for $1 \leq i, j \leq n-1$.
(27) $(2\,,\, s_i \,/\, t_i\, f_i\,,\, 0)$, for $1 \leq i \leq n-1$.
(28) $(2\,,\, u_{1,j} \,/\, q_{1,j+1}\, q_{2,j+1}\,,\, 0)$, for $1 \leq j \leq n-2$.
(29) $(2\,,\, u_{i,j} \,/\, q_{i+1,j+1}\,,\, 0)$, for $2 \leq i, j \leq n-2$.
(30) $(2\,,\, u_{i,n-1} \,/\, \lambda\,,\, 0)$, for $1 \leq i \leq n-1$.
(31) $(2\,,\, T_i \,/\, \lambda\,,\, 0)$, for $1 \leq i \leq n-1$.
(32) $(2\,,\, T'_i \,/\, \lambda\,,\, 0)$, for $1 \leq i \leq n-1$.
(33) $(2\,,\, F_i \,/\, \lambda\,,\, 0)$, for $1 \leq i \leq n-1$.
(34) $(2\,,\, F'_i \,/\, \lambda\,,\, 0)$, for $1 \leq i \leq n-1$.
(35) $[\,S\,]_1 \longrightarrow [\,\Gamma_1\,]_1\, [\,\Gamma_2\,]_1$
(36) $(2\,,\, \alpha_i \,/\, \alpha_{i+1}\,,\, 0)$, for $0 \leq i \leq 3n+2m$.
(37) $(3\,,\, \beta_i \,/\, \beta_{i+1}\,,\, 0)$, for $0 \leq i \leq 3n+2m+1$.
(38) $(3\,,\, x_{i,j} \,/\, d_{i,j,1}^2\,,\, 0)$, $(3\,,\, \bar{x}_{i,j} \,/\, \bar{d}_{i,j,1}^2\,,\, 0)$, for $1 \leq i \leq n$, $1 \leq j \leq m$

(39) $(3\,,\, d_{i,j,k} \,/\, d^2_{i,j,k+1}\,,\, 0)$, $(3\,,\, \bar{d}_{i,j,k} \,/\, \bar{d}^2_{i,j,k+1}\,,\, 0)$, for $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq k \leq n-1$.
(40) $(3\,,\, d_{i,j,n} \,/\, e_{i,j}, 0)$, $(3\,,\, \bar{d}_{i,j,n} \,/\, \bar{e}_{i,j}\,,\, 0)$, for $1 \leq i \leq n$, $1 \leq j \leq m$.
(41) $(1\,,\, T_i E_j \,/\, e_{i,j}\,,\, 3)$, $(1\,,\, F_i E_j \,/\, \bar{e}_{i,j}\,,\, 3)$, $(1\,,\, T'_i E_j \,/\, e_{i,j}\,,\, 3)$, $(1\,,\, F'_i E_j \,/\, \bar{e}_{i,j}\,,\, 3)$, for $1 \leq i \leq n$, $1 \leq j \leq m$.
(42) $(1\,,\, e_{i,j} \,/\, T_i E_{j+1}\,,\, 0)$, $(1\,,\, \bar{e}_{i,j} \,/\, F_i E_{j+1}\,,\, 0)$, for $1 \leq i \leq n$, $1 \leq j \leq m-1$.
(43) $(1\,,\, e_{i,m} \,/\, E_{m+1}\,,\, 0)$, $(1\,,\, \bar{e}_{i,m} \,/\, E_{m+1}\,,\, 0)$, for $1 \leq i \leq n$.
(44) $(3,\, T_i \,/\, \lambda\,, 0)$, $(3\,,\, F_i \,/\, \lambda\,,\, 0)$, $(3\,,\, T'_i \,/\, \lambda\,,\, 0)$, $(3\,,\, F'_i \,/\, \lambda\,,\, 0)$, for $1 \leq i \leq n$.
(45) $(3\,,\, E_j \,/\, \lambda\,,\, 0)$, for $1 \leq j \leq m$.
(46) $(1\,,\, E_{m+1} \,/\, \texttt{yes}\ \alpha_{3n+1+2m}\,,\, 2)$.
(47) $(1\,,\, \texttt{yes} \,/\, \beta_{3n+1+2m+1}\,,\, 3)$.
(48) $(2\,,\, \alpha_{3n+1+2m} \,/\, \beta_{3n+1+2m+1}\,,\, 3)$.
(49) $(2\,,\, \texttt{no}\ \beta_{3n+1+2m+1} \,/\, \lambda\,,\, 0)$.
(50) $(3\,,\, \texttt{yes} \,/\, \lambda\,,\, 0)$.

- The input cell is $i_{in} = 3$.
- The output cell is the environment, $i_{out} = 0$.

5.1 An Overview of the Computation

A family of recognizer tissue P systems with cell separation is constructed above. For an instance of the SAT problem $\varphi = C_1 \wedge \cdots \wedge C_m$, consisting of m clauses $C_j = l_{j,1} \vee \cdots \vee l_{j,r_j},\ 1 \leq j \leq m$, where $Var(\varphi) = \{x_1, \cdots, x_n\}$, $l_{j,k} \in \{x_i, \neg x_i \mid 1 \leq i \leq n\},\ 1 \leq j \leq m, 1 \leq k \leq r_j$. Let us assume that the number of variables, n, and the number of clauses, m, of the input formula φ, are greater or equal to 2.

The *size* mapping on the set of instances is defined as $s(\varphi) = \langle m, n \rangle$, and the encoding of the instance is the multiset

$$cod(\varphi) = \{x_{i,j} :\ x_i \in C_j\} \ \cup\ \{\overline{x}_{i,j} :\ \neg x_i \in C_j\}$$

That is, $x_{i,j}$ (respectively, $\overline{x}_{i,j}$) denotes variable x_i (respectively, $\neg x_i$) belongs to clause C_j. Then the formula φ will be processed by the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$.

Next, we informally describe how system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$ works, in order to process the instance φ of the SAT problem.

At the initial configuration we have objects A_1, B_1 in cell 1, objects a_1, a'_1, b_1, b'_1, $v_1, q_{1,1}, \alpha_0, \texttt{yes}, \texttt{no}$ in cell 2, and $cod(\varphi), \beta_0$ in cell 3.

Let us start with the **generation stage**. This stage spends $3n+1$ steps and has, basically, two parallel processes. On the one hand, n loops are executed, each loop spends 3 steps involving cells 1 and 2. After the loops are finished, an additional step goes on. On the other hand, in cell 3 there is a counter β that evolves from β_0 to β_{3n+1} by applying rules of the type (37), and $cod(\varphi)$ produces $((cod(\varphi))^{2^n}_e$ after the $3n+1$ steps at this stage.

At the first step of the i–th loop $(0 \leq i \leq n)$ involving cells 1 and 2, objects

$$A_{i+1}, A'_{i+1}, B_{i+1}, B'_{i+1}, T_j, T'_j, F_j, F'_j$$

in cell 1 exchange objects

$$a_{i+1}a'_{i+1}, a_{i+1}a'_{i+1}, b_{i+1}b'_{i+1}, b_{i+1}b'_{i+1}, t_j, t_j, f_j, f_j$$

with cell 2, where also v_{i+1} produces y^2_{i+1}, and $q_{1,i+1}, \dots q_{i+1,i+1}$ ($q_{1,1}$ at step 1) produce objects $r^2_{1,i+1}, \dots r^2_{i+1,i+1}$ ($r_{1,1}$ at step 1).

At the second step of the i–th loop ($0 \leq i \leq n$), objects

$$a_{i+1}, a'_{i+1}, b_{i+1}, b'_{i+1}, t_j, f_j$$

in cells 1 produce objects

$$T_{i+1}A_{i+2}, F'_{i+1}A'_{i+2}, B_{i+2}S, B'_{i+2}, T_jT'_j, F_jF'_j$$

according to the rules (9), (10), (11), (12), (13), (14). Simultaneously, at this step objects

$$A_{i+1}, A'_{i+1}, B_{i+1}, B'_{i+1}, T_j, T'_j, F_j, F'_j, y_{i+1}, r_{1,i+1}, \dots r_{i+1,i+1}$$

in cell 2 produce objects

$$c_{i+1}, c_{i+1}, c_{i+1}, c_{i+1}, \lambda, \lambda, \lambda, \lambda, \lambda, z_{i+1}w_{i+1}, s_1u_{1,i+1} \dots s_{i+1}u_{i+1,i+1}$$

respectively, according to the rules (15), (16), (17), (18), (21), (26), (31), (32), (33), (34).

At the third step of the i–th loop ($1 \leq i \leq n-1$), object S triggers the separation of objects of cells 1 in two new cells 1 by applying the separation rule (35), according to Γ_1 (objects without primes) and Γ_2 (objects with primes). At this step, objects

$$c_{i+1}, z_{i+1}, w_{i+1}, s_1, \dots, s_{i+1}, u_{1,i+1}, \dots, u_{i+1,i+1}$$

in cell 2 produce objects

$$b_{i+2}b'_{i+2}, v_{i+2}, a_{i+2}a'_{i+2}, f_1t_1, \dots, f_{i+1}t_{i+1}, q_{1,i+2} \dots q_{i+1,i+2}, q_{i+2,i+2}$$

according to the rules (19), (22), (23), (27), (29), respectively.

After $3(n-1)$ transition steps, we have

(a) 2^{n-1} cells 1 such that 2^{n-2} cells contain objects T_{n-1}, A_n, B_n and a different truth assignment of $\sigma_{n-2,j}$ of the set $\{x_1, \dots, x_{n-2}\}$, and 2^{n-2} cells contain objects F'_{n-1}, A'_n, B'_n and a different truth assignment of $\tau_{n-2,j}$ of the set $\{x_1, \dots, x_{n-2}\}$.

(b) A cell 2 that contains objects

$$a_n^{2^{n-1}}, a_n'^{2^{n-1}}, b_n^{2^{n-1}}, b_n'^{2^{n-1}}, v_n^{2^{n-1}}, f_1^{2^{n-2}}, t_1^{2^{n-2}}, \dots, f_{n-1}^{2^{n-2}}, t_{n-1}^{2^{n-2}}$$

(c) A cell 3 which contains object $\beta_{3(n-1)}$ and $(cod(\varphi))_e^{2^n}$.

By applying rules (1), (2), (3), (4), (5), (6), (7), (8), (20), (36), and (37) at step $3n-2$, and rules (9), (10), (11), (12), (13), (14), (15), (16), (17, (18), (31), (32), (33), (34),(36), and (37) at step $3n-1$, and rules $(1, B_{n+1}/\lambda, 0)$, $(1, B'_{n+1}/\lambda, 0)$ $(2, w_n/E_1, 0)$, (35), (36), and (37) at step $3n$, we reach the following configuration $\mathcal{C}_{3n+1}$:

- There are 2^n cells 1 which contain object E_1 and each of them encodes a different truth assignment of the set $\{x_1, \ldots, x_n\}$.
- There is a cell 2 which contains objects $A_{n+1}^{2^{n+1}}$, $A_{n+1}'^{2^{n+1}}$, α_{3n+1} , `yes` , `no`.
- There is a cell 3 which contains object β_{3n+1} and $(cod(\varphi))_e^{2^n}$.

In this way, after the $(3n+1)$–th step the generation stage finishes and the **checking stage** starts. This stage spends $2m$ steps and consists of m loops each of them spending 2 steps.

At the first step of the j–th loop $(1 \leq j \leq m)$, objects $e_{i,j}$ and $\overline{e}_{i,j}$ from cell 3 are traded for objects E_j from cell 1, in the case that cell 1 encodes a truth assignment making clauses $C_1, \ldots, C_j$ true. Simultaneously, in cell 2 counter α continue evolving and objects `yes` and `no` remain unchanged. In cell 3, counter β continue evolving, and object E_j appears k_j times, where k_j is the number of cells labelled by 2 encoding a truth assignment making clauses $C_1, \ldots, C_j$ true.

At the second step of the j–th loop $(1 \leq j \leq m)$, rules (41) produce objects T_i, E_{j+1} in each cell 1 encoding a truth assignment making clauses $C_1, \ldots, C_j$ true. Simultaneously, in cell 2 counter α continue evolving and objects `yes` and `no` remain unchanged. In cell 3, counter β, and objects E_{j+1} are removed by applying rule (5).

At the end of the checking stage, there are 2^n cells labelled by 1 at configuration $\mathcal{C}_{(3n+1)+2m}$, and the formula φ is satisfiable if and only if there is, at least, one of such cell which contains object E_{m+1}. Also, there is a cell labelled by 2 which contains objects `yes`, `no`, $\alpha_{(3n+1)+2m}$, and a cell labelled by 3 which contains object $\beta_{(3n+1)+2m}$ and some irrelevant objects of the type $e_{i,j'}, \overline{e}_{i,j'}$ with $1 \leq j' \leq m$. Irrelevant objects are those which remain unchanged at the following computation steps and do not take part in the application of any rule of the system.

The **output stage** starts at the $((3n+1)+2m+1)$-th step, and spends 3 steps.

– *Affirmative answer* : If a truth assignment encoded by a cell 1 makes the formula φ true, then an object E_{m+1} appears in that cell. By applying rule (46) one (and only one) object E_{m+1} is replaced by objects `yes` and $\alpha_{3n+1+2m}$ from cell 2. At the next step, object `yes` from cell 1 is exchanged for object $\beta_{3n+1+2m+1}$ from cell 2. Finally, at step $3n+1+2m+3$ object `yes` from cell 3 is sent out to the environment by applying rule (50), and the computation halts.

– *Negative answer* : If none of the truth assignments encoded by a cell 1 makes the formula φ true, then object E_{m+1} does not appear at any cell labelled by 1. Thus, rule (46) is not applicable at configuration $\mathcal{C}_{(3n+1)+2m}$, and only rule

(37) is applicable and produces object $\beta_{3n+1+2m+1}$ in cell 3. Then, only rule (48) is applicable at configuration $\mathcal{C}_{(3n+1)+2m+1}$ and replaces object $\alpha_{3n+1+2m}$ from cell 2 by object $\beta_{3n+1+2m+1}$ from cell 3. Finally, at step $3n+1+2m+3$ objects `no` and $\beta_{3n+1+2m+1}$ from cell 2 are sent out to the environment by applying rule (49), and the computation halts.

6 A Formal Verification

The aim of this section is to present a formal proof that the family of recognizer tissue P systems with cell separation constructed in the previous section solves in a uniform way and polynomial time the `SAT` problem, according to Definition 1.

6.1 Polynomial Uniformity of the Family

In this subsection, we shall show that the family

$$\mathbf{\Pi} = \{\Pi(\langle m, n\rangle) \mid m, n \in \mathbb{N}\}$$

defined above is polynomially uniform by Turing machines. To this aim we prove that $\Pi(\langle m, n\rangle)$ is built in polynomial time with respect to the size parameter m and n of instances of the `SAT` problem.

It is easy to check that the rules of a system $\Pi(\langle m, n\rangle)$ of the family are recursively defined from the values m and n. The amount of resources to build an element of the family is of a polynomial order in the number n of the variables and the number m of clauses, as shown below:

1. Size of the alphabet: $2mn^2 + 5mn + 3n^2 + 5m + 27n + 12 \in \Theta(mn^2)$.
2. Initial number of cells: $3 \in \Theta(1)$.
3. Initial number of objects: $12 \in \Theta(1)$.
4. Number of rules: $mn^2 + 3mn + 3n^2 + 5m + 30n + 12 \in \Theta(mn^2)$.
5. Maximal length of a rule: $3 \in \Theta(1)$.

Therefore, there exists a deterministic Turing machine that builds the system $\Pi(\langle m, n\rangle)$ in a polynomial time with respect to m and n.

6.2 Soundness and Completeness of the Family

Let us start by fixing some notations that will allow us to describe the invariants, appearing in the computation, in a simpler way.

Let $\{x_1, \ldots, x_i\}$ a set of propositional variables. A truth assignment of $\{x_1, \ldots, x_i\}$ will be indistinctly denoted by:

- $\sigma_i = (\alpha_1, \ldots, \alpha_i)$, where $\alpha_j \in \{T, F\}$.
- $\tau_i = (\beta_1, \ldots, \beta_i)$, where $\beta_j \in \{T', F'\}$.
- $\epsilon_i = (\gamma_1, \ldots, \gamma_i)$, where $\gamma_j \in \{t, f\}$.

The 2^i truth assignment of the set $\{x_1,\ldots,x_i\}$ will be indistinctly denoted by $\{\sigma_{i,1},\ldots,\sigma_{i,2^i}\}$, $\{\tau_{i,1},\ldots,\tau_{i,2^i}\}$, or $\{\epsilon_{i,1},\ldots,\epsilon_{i,2^i}\}$, respectively. Notice that given a truth assignment $\sigma_{i,j}$ $(1\leq j\leq 2^i)$ of $\{x_1,\ldots,x_i\}$, we can briefly write the same truth assignment with primes as $\tau_{i,j}$, or in lowercase as $\epsilon_{i,j}$.

Let $\varphi = C_1\wedge\cdots\wedge C_m$, where $C_j = l_{j,1}\vee\cdots\vee l_{j,r_j}$, $1\leq j\leq m$, and each $l_{j,k}$ is an element of the set $\mathrm{Var}(\varphi)=\{x_i,\neg x_i\,|\,1\leq i\leq n\}$. We denote

$$\begin{aligned}
cod(\varphi) &= \{x_{i,j}:\ x_i\in C_j,\ 1\leq i\leq n, 1\leq j\leq m\}\cup\\
&\quad\{\overline{x}_{i,j}:\ \neg x_i\in C_j,\ 1\leq i\leq n, 1\leq j\leq m\}\\
(cod(\varphi))_e &= \{e_{i,j}:\ x_i\in C_j,\ 1\leq i\leq n, 1\leq j\leq m\}\cup\\
&\quad\{\overline{e}_{i,j}:\ \neg x_i\in C_j,\ 1\leq i\leq n, 1\leq j\leq m\}\\
(cod(\varphi))^t_e &= \{e^t_{i,j}:\ x_i\in C_j,\ 1\leq i\leq n, 1\leq j\leq m\}\cup\\
&\quad\{\overline{e}^t_{i,j}:\ \neg x_i\in C_j,\ 1\leq i\leq n, 1\leq j\leq m\}
\end{aligned}$$

For each k $(1\leq k\leq n)$ we denote

$$\begin{aligned}
(cod(\varphi))_{e,>k} &= (cod(\varphi))_e - (\{e_{i,j}:\ x_i\in C_j,\ 1\leq i\leq n, 1\leq j\leq m, 1\leq j\leq k\}\cup\\
&\qquad\{\overline{e}_{i,j}:\ \neg x_i\in C_j,\ 1\leq i\leq n, 1\leq j\leq m, 1\leq j\leq k\})\\
(cod(\varphi))^t_{e,>k} &= (cod(\varphi))^t_e - (\{e^t_{i,j}:\ x_i\in C_j,\ 1\leq i\leq n, 1\leq j\leq m, 1\leq j\leq k\}\cup\\
&\qquad\{\overline{e}^t_{i,j}:\ \neg x_i\in C_j,\ 1\leq i\leq n, 1\leq j\leq m, 1\leq j\leq k\})
\end{aligned}$$

For each i,j,k $(1\leq i,k\leq n, 1\leq j\leq m)$ we denote

$$\begin{aligned}
(cod(\varphi))_{d_{i,j,k}} &= \{d_{i,j,k}:\ x_i\in C_j\}\ \cup\ \{\overline{d}_{i,j,k}:\ \neg x_i\in C_j\}\\
(cod(\varphi))^t_{d_{i,j,k}} &= \{d^t_{i,j,k}:\ x_i\in C_j\}\ \cup\ \{\overline{d}^t_{i,j,k}:\ \neg x_i\in C_j\}
\end{aligned}$$

The 2^n cells labelled by 1 generated by the system will be enumerated by $(1,1),(1,2),\ldots,(1,2^{n-1}),(1,2^{n-1}+1),\ldots,(1,2^n)$, in such a way that cells labelled by $(1,1),(1,2),\ldots,(1,2^{n-1})$ contain T_n and the values of the truth assignment *without primes* $\sigma_{n-1,1},\ldots,\sigma_{n-1,2^{n-1}}$ of the set $\{x_1,\ldots,x_{n-1}\}$, and cells labelled by $(1,2^{n-1}+1),\ldots,(1,2^n)$ contain F'_n and the values of the truth assignment *with primes* $\tau_{n-1,1},\ldots,\tau_{n-1,2^{n-1}}$ of the set $\{x_1,\ldots,x_{n-1}\}$. If $\mathcal{C}=(\mathcal{C}_0,\mathcal{C}_1,\ldots)$ is a computation of the tissue P system $\Pi(\langle m,n\rangle)$ and l is the label of a cell, then we denote by $\mathcal{C}_i(l)$ the contents of cell l at configuration $\mathcal{C}_i$.

Theorem 6.1 *Let $\mathcal{C}=(\mathcal{C}_0,\mathcal{C}_1,\ldots)$ be a computation of the tissue P system $\Pi(\langle m,n\rangle)$. For every i $(1\leq i\leq n-1)$, we have the following:*

(1) At configuration $\mathcal{C}_{3i}$:

(a) There are 2^i cells labelled by 1 from which:

- ⋆ *2^{i-1} cells contain objects T_i, , A_{i+1} , B_{i+1}. Moreover, each of them contains a different truth assignment $\sigma_{i-1,j}$ of the set $\{x_1,\ldots,x_{i-1}\}$.*
- ⋆ *2^{i-1} cells contain objects F'_i, , A'_{i+1} , B'_{i+1}. Moreover, each of them contains a different truth assignment $\tau_{i-1,j}$ of the set $\{x_1,\ldots,x_{i-1}\}$.*

(b) There is a cell labelled by 2. This cell contains objects α_{3i}, `yes`, `no`, and

- ⋆ *If $i < n-1$ then it contains objects*

$$a_{i+1}^{2^i},\ {a'}_{i+1}^{2^i},\ b_{i+1}^{2^i},\ {b'}_{i+1}^{2^i},\ v_{i+1}^{2^i},\ t_1^{2^{i-1}}\ f_1^{2^{i-1}},\ \ldots,\ t_i^{2^{i-1}}\ f_i^{2^{i-1}},$$
$$q_{1,i+1}^{2^{i-1}},\ \cdots,\ q_{i+1,i+1}^{2^{i-1}}$$

- ⋆ *If $i = n-1$ then it contains objects*

$$a_{i+1}^{2^i},\ {a'}_{i+1}^{2^i},\ b_{i+1}^{2^i},\ {b'}_{i+1}^{2^i},\ v_{i+1}^{2^i},\ t_1^{2^{i-1}}\ f_1^{2^{i-1}},\ \ldots,\ t_i^{2^{i-1}}\ f_i^{2^{i-1}}$$

(c) There is a cell labelled by 3. This cell contains object β_{3i}, and

- ⋆ *If $3i \leq n$ then it also contains $(cod(\varphi))^{2^{3i}}_{d_{i,j,3i}}$*
- ⋆ *If $3i > n$ then it also contains $(cod(\varphi))^{2^n}_e$*

(2) At configuration $\mathcal{C}_{3i+1}$:

(a) There are 2^i cells labelled by 1.

- ⋆ *Each of them contains objects a_{i+1}, ${a'}_{i+1}$, b_{i+1}, ${b'}_{i+1}$.*
- ⋆ *Each of them contains a different truth assignment $\epsilon_{i,j}$ of the set $\{x_1, \ldots, x_i\}$.*

(b) There is a cell labelled by 2. This cell contains objects

$$A_{i+1}^{2^{i-1}},\ {A'}_{i+1}^{2^{i-1}},\ B_{i+1}^{2^{i-1}},\ {B'}_{i+1}^{2^{i-1}},\ y_{i+1}^{2^{i+1}},\ \alpha_{3i+1},\ \texttt{yes},\ \texttt{no}$$

$$T_i^{2^{i-1}}\ \sigma_{i-1,1}\ \ldots\ \sigma_{i-1,2^{i-1}}\quad {F'}_i^{2^{i-1}}\ \tau_{i-1,1}\ \ldots\ \tau_{i-1,2^{i-1}}$$

Moreover, if $i < n-1$ then it also contains objects

$$r_{1,i+1}^{2^i},\ \ldots,\ r_{i+1,i+1}^{2^i}$$

(c) There is a cell labelled by 3. This cell contains object β_{3i+1}, and

- ⋆ *If $3i+1 \leq n$ then it also contains $(cod(\varphi))^{2^{3i+1}}_{d_{i,j,3i+1}}$*
- ⋆ *If $3i+1 > n$ then it also contains $(cod(\varphi))^{2^n}_e$*

(3) At configuration $\mathcal{C}_{3i+2}$:

(a) There are 2^i cells labelled by 1.

- ⋆ *Each of them contains objects*

$$A_{i+2},\ {A'}_{i+2},\ B_{i+2},\ {B'}_{i+2},\ S,\ T_{i+1},\ {F'}_{i+1}$$

- ⋆ *Each of them contains a different truth assignment $\sigma_{i,j}$ of the set $\{x_1, \ldots, x_i\}$, as well as an identical copy, $\tau_{i,j}$, but for primes.*

(b) There is a cell labelled by 2. This cell contains objects $\alpha_{3i+2}, \texttt{yes}, \texttt{no}$, and such that:

- ⋆ *If $i < n-1$ then it also contains objects*

$$c_{i+1}^{2^{i+1}},\ z_{i+1}^{2^{i+1}},\ w_{i+1}^{2^{i+1}},\ ,\ s_1^{2^i},\ \ldots,\ s_{i+1}^{2^i},\ u_{1,i+1}^{2^i},\ \ldots,\ u_{i+1,i+1}^{2^i}$$

- ⋆ *If $i = n-1$ then it also contains objects $w_{i+1}^{2^{i+1}}$.*

(c) There is a cell labelled by 3. This cell contains object β_{3i+2}, and such that:

- ⋆ *If $3i+2 \leq n$ then it also contains $(cod(\varphi))_{d_{i,j,3i+2}}^{2^{3i+2}}$*
- ⋆ *If $3i+2 > n$ then it also contains $(cod(\varphi))_e^{2^n}$*

Proof: By induction on i. Let us start analyzing the basic case $i = 1$.

At the initial configuration we have:

$$\begin{cases} \mathcal{C}_0(1) = \{A_1, B_1\} \\ \mathcal{C}_0(2) = \{a_1, a_1', b_1, b_1', v_1, q_{1,1}, \alpha_0, \texttt{yes}, \texttt{no}\} \\ \mathcal{C}_0(3) = \{\beta_0\} \ \cup \ cod(\varphi) \end{cases}$$

Then, rules (1) and (3) allow to exchange objects A_1, B_1 from cell 1 for objects a_1, a_1', b_1, b_1' from cell 2. Simultaneously, the application of rules (20), (24) and (36) produce objects $y_1^2, r_{1,1}, \alpha_1$ in cell 2. Rule (37) produces object β_1 in cell 3, and rule (38) produce objects $d_{i,j,1}^2$ if $x_{i,j} \in cod(\varphi)$, and objects $\overline{d}_{i,j,1}^2$ if $\overline{x}_{i,j} \in cod(\varphi)$, in cell 3. Therefore,

$$\begin{cases} \mathcal{C}_1(1) = \{a_1, a_1', b_1, b_1'\} \\ \mathcal{C}_1(2) = \{A_1, B_1, y_1^2, r_{1,1}, \alpha_1, \texttt{yes}, \texttt{no}\} \\ \mathcal{C}_1(3) = \{\beta_1\} \ \cup \ \{d_{i,j,1}^2 : \ x_{i,j} \in cod(\varphi)\} \ \cup \ \{\overline{d}_{i,j,1}^2 : \ \overline{x}_{i,j} \in cod(\varphi)\} \end{cases}$$

At configuration $\mathcal{C}_1$:

(a) Rules (11), (12), (13) and (14) produce objects $B_2S, B_2', T_1A_2, F_1'A_2'$ in cell 1.
(b) Rules (15), (17), (21), (26) and (36) produce objects $c_1, c_1, z_1^2 w_1^2, s_1 u_{1,1}, \alpha_2$ in cell 2.
(c) Rules (37) and (39) produce objects $\beta_2, d_{i,j,2}^{2^2}$ with $x_{i,j} \in cod(\varphi)$, and $\overline{d}_{i,j,2}^{2^2}$ with $\overline{x}_{i,j} \in cod(\varphi)$ in cell 3.

That is,

$$\begin{cases} \mathcal{C}_2(1) = \{T_1, A_2, F_1', A_2', B_2, S, B_2'\} \\ \mathcal{C}_2(2) = \{c_1^2, z_1^2, w_1^2, s_1, u_{1,1}, \alpha_2, \ \texttt{yes}, \texttt{no}\} \\ \mathcal{C}_2(3) = \{\beta_2\} \ \cup \ \{d_{i,j,2}^{2^2} : \ x_{i,j} \in cod(\varphi)\} \ \cup \ \{\overline{d}_{i,j,2}^{2^2} : \ \overline{x}_{i,j} \in cod(\varphi)\} \end{cases}$$

At configuration $\mathcal{C}_2$:

(a) Object S triggers separation rule (35) creating two new cells 1, one of them (1,1) containing $\{A_2, B_2, T_1\}$, and the other one (1,2) containing $\{A_2', B_2', F_1'\}$.
(b) If $1 = i = n-1$ (that is, $n = 2$) rules (19), (22), (23), (27), and (36) produce objects

$$b_2 b_2', v_2^2 a_2^2 a_2'^2, t_1 f_1, \alpha_3$$

in cell 2. Rule (30) remove object $u_{1,1}$.

⋆ If $1 = i < n - 1$ (that is, $n > 2$) rules (19), (22), (23), (27), (28) and (36) produce objects

$$b_2 b'_2, v_2^2, a_2^2 a_2'^2, t_1 f_1, q_{1,2} q_{2,2}, \alpha_3$$

in cell 2.

(c) If $3 = 3i \leq n$ (that is, $n > 2$) rules (37), and (39) produce objects $\beta_3, d_{i,j,3}^{2^3}$ with $x_{i,j} \in cod(\varphi)$, and $\overline{d}_{i,j,3}^{2^3}$ with $\overline{x}_{i,j} \in cod(\varphi)$ in cell 3.

⋆ If $3 = 3i > n$ (that is, $n = 2$) rules (37), and (40) produce objects $\beta_3, e_{i,j}^{2^2}$ with $x_{i,j} \in cod(\varphi)$, and $\overline{e}_{i,j}^{2^2}$ with $\overline{x}_{i,j} \in cod(\varphi)$ in cell 3.

That is,

$$\begin{cases} \mathcal{C}_3(1,1) = \{A_2, B_2, T_1\} \\ \mathcal{C}_3(1,2) = \{A'_2, B'_2, F'_1\} \\ \mathcal{C}_3(2) = \{a_2^2, a'^2_2, b_2^2, b'^2_2, v_2^2, t_1, f_1, \alpha_3, \texttt{yes}, \texttt{no}\}, \text{ if } n = 2 \\ \mathcal{C}_3(2) = \{a_2^2, a'^2_2, b_2^2, b'^2_2, v_2^2, t_1, f_1, q_{1,2}, q_{2,2}, \alpha_3, \texttt{yes}, \texttt{no}\}, \text{ if } n > 2 \\ \mathcal{C}_3(3) = \{\beta_3\} \cup \{e_{i,j}^{2^2} : x_{i,j} \in cod(\varphi)\} \cup \{\overline{e}_{i,j}^{2^2} : \overline{x}_{i,j} \in cod(\varphi)\}, \text{ if } n = 2 \\ \mathcal{C}_3(3) = \{\beta_3\} \cup \{d_{i,j,3}^{2^3} : x_{i,j} \in cod(\varphi)\} \cup \{\overline{d}_{i,j,3}^{2^3} : \overline{x}_{i,j} \in cod(\varphi)\}, \text{ if } n > 2 \end{cases}$$

At configuration $\mathcal{C}_3$:

(a) Rules (1), (2), (3), (4), (5), and (8) replace objects

$$A_2, A'_2, B_2, B'_2, T_1, F'_1$$

from cell 1 by objects

$$a_2, a'_2, a_2, a'_2, b_2, b'_2, b_2, b'_2, t_1, f_1$$

from cell 2.

(b) Rules (20) and (36) produce objects $y_2^{2^2}, \alpha_4$ in cell 2. Moreover, if $1 = i < n-1$ (that is, $n > 2$) then rule (25) produce objects $r_{1,2}^2, r_{2,2}^2$.

(c) Rule (37) produces object β_4 in cell 3. Moreover, if $4 = 3i + 1 \leq n$ (that is, $3i \leq n$) then rule (39) produce objects $d_{i,j,4}^{2^4}$ with $x_{i,j} \in cod(\varphi)$, and $\overline{d}_{i,j,4}^{2^4}$ with $\overline{x}_{i,j} \in cod(\varphi)$ in cell 3.

⋆ If $4 = 3i + 1 > n$ and $n = 2$, then objects $e_{i,j}^{2^2}$, $\overline{e}_{i,j}^{2^2}$ in cell 3 do not evolve.

⋆ If $4 = 3i + 1 > n$ and $n = 3$, then rule (40) produce objects $e_{i,j}^{2^3}$ with $x_{i,j} \in cod(\varphi)$, and $\overline{e}_{i,j}^{2^3}$ with $\overline{x}_{i,j} \in cod(\varphi)$ in cell 3.

That is,

$$\begin{cases} \mathcal{C}_4(1,1) = \{a_2, a'_2, b_2, b'_2, t_1\} \\ \mathcal{C}_4(1,2) = \{a_2, a'_2, b_2, b'_2, f_1\} \\ \mathcal{C}_4(2) = \{A_2, A'_2, B_2, B'_2, T_1, F'_1, y_2^{2^2}, \alpha_4, \texttt{yes}, \texttt{no}\}, \text{ if } n = 2 \\ \mathcal{C}_4(2) = \{A_2, A'_2, B_2, B'_2, T_1, F'_1, y_2^{2^2}, \alpha_4, r_{1,2}^2, r_{2,2}^2, \texttt{yes}, \texttt{no}\}, \text{ if } n > 2 \\ \mathcal{C}_4(3) = \{\beta_4\} \cup \{e_{i,j}^{2^n} : x_{i,j} \in cod(\varphi)\} \cup \{\overline{e}_{i,j}^{2^n} : \overline{x}_{i,j} \in cod(\varphi)\}, \text{ if } n = 2, 3 \\ \mathcal{C}_4(3) = \{\beta_4\} \cup \{d_{i,j,4}^{2^4} : x_{i,j} \in cod(\varphi)\} \cup \{\overline{d}_{i,j,4}^{2^4} : \overline{x}_{i,j} \in cod(\varphi)\}, \text{ if } n \geq 4 \end{cases}$$

At configuration $\mathcal{C}_4$:

(a) Rules (9), (11), (12), (13), and (14) produce objects

$$T_1T_1', B_3S, B_3', T_2A_3, F_2', A_3'$$

in cell (1,1), and rules (10), (11), (12), and (13) produce objects

$$F_1F_1', B_3S, B_3', T_2A_3, F_2', A_3'$$

in cell (1,2).

(b) Rule (36) produces object α_5 in cell 2. Moreover, if $1 = i < n-1$ (that is, $n > 2$) then rules (15), (16), (17), (18), (21), and (26) produce objects

$$c_2, c_2, c_2, c_2, z_2^{2^2} w_2^{2^2}, s_1^2 u_{1,2}^2, s_2^2 u_{2,2}^2$$

in cell 2.

⋆ If $i = 1 = n-1$ (that is, $n = 2$), then rules (15), (16), (17), and (18) remove objects A_2, A_2', B_2, B_2' from cell 2, and rule (21) produce objects w_2^2. Rules (31) and (34) remove objects T_1, F_1' from cell 2.

(c) Rule (37) produces object β_5 in cell 3. Moreover,

⋆ if $5 = 3i + 2 \leq n$ (thus $3i + 1 < n$) then rule (39) produce objects $d_{i,j,5}^{2^5}$ with $x_{i,j} \in cod(\varphi)$, and $\overline{d}_{i,j,5}^{2^5}$ with $\overline{x}_{i,j} \in cod(\varphi)$ in cell 3.

⋆ If $5 = 3i + 2 > n$ and $n = 2, 3$, then objects $e_{i,j}^{2^n}$, $\overline{e}_{i,j}^{2^n}$ in cell 3 do not evolve.

⋆ If $5 = 3i + 2 > n$ and $n = 4$, then rule (40) produce objects $e_{i,j}^{2^4}$ with $x_{i,j} \in cod(\varphi)$, and $\overline{e}_{i,j}^{2^4}$ with $\overline{x}_{i,j} \in cod(\varphi)$ in cell 3.

That is,

$$\begin{cases} \mathcal{C}_5(1,1) = \{A_3, A_3', B_3, B_3', S, T_1, T_1', T_2, F_2'\} \\ \mathcal{C}_5(1,2) = \{A_3, A_3', B_3, B_3', S, F_1, F_1', T_2, F_2'\} \\ \mathcal{C}_5(2) = \{w_2^{2^2}, \alpha_5, \texttt{yes}, \texttt{no}\}, \text{ if } n = 2 \\ \mathcal{C}_5(2) = \{c_2^{2^2}, z_2^{2^2}, s_1^2, u_{1,2}^2, s_2^2, u_{2,2}^2, w_2^{2^2}, \alpha_5, \texttt{yes}, \texttt{no}\}, \text{ if } n > 2 \\ \mathcal{C}_5(3) = \{\beta_5\} \cup \{e_{i,j}^{2^n} : x_{i,j} \in cod(\varphi)\} \cup \{\overline{e}_{i,j}^{2^n} : \overline{x}_{i,j} \in cod(\varphi)\}, \text{ if } n < 5 \\ \mathcal{C}_5(3) = \{\beta_5\} \cup \{d_{i,j,5}^{2^5} : x_{i,j} \in cod(\varphi)\} \cup \{\overline{d}_{i,j,5}^{2^5} : \overline{x}_{i,j} \in cod(\varphi)\}, \text{ if } n \geq 5 \end{cases}$$

Thus, the result of the theorem hold for $i = 1$.

By induction hypothesis, let i be such that $1 \leq i < n-1$ and let us suppose (1), (2), and (3) hold for i. Let us see that (1), (2), and (3) also hold for $i + 1$.

Then we assume that:

$$\begin{cases} \mathcal{C}_{3i+2}(1,1) = \{A_{i+2}, A_{i+2}', B_{i+2}, B_{i+2}', S, T_{i+1}, F_{i+1}', \sigma_{i,1}, \tau_{i,1}\} \\ \cdots\cdots\cdots \quad \cdots\cdots\cdots\cdots\cdots \\ \mathcal{C}_{3i+2}(1,2^i) = \{A_{i+2}, A_{i+2}', B_{i+2}, B_{i+2}', S, T_{i+1}, F_{i+1}', \sigma_{i,2^i}, \tau_{i,2^i}\} \\ \mathcal{C}_{3i+2}(2) = \{c_{i+1}^{2^{i+1}}, z_{i+1}^{2^{i+1}}, w_{i+1}^{2^{i+1}}, s_1^{2^i}, \ldots, s_{i+1}^{2^i}, u_{1,i+1}^{2^i}, \ldots u_{i+1,i+1}^{2^i}, \alpha_{3i+2}, \texttt{yes}, \texttt{no}\} \\ \mathcal{C}_{3i+2}(3) = \{\beta_{3i+2}\} \cup (cod(\varphi))_{d_{i,j,3i+2}}^{2^{3i+2}}, \text{ if } 3i+2 \leq n \\ \mathcal{C}_{3i+2}(3) = \{\beta_{3i+2}\} \cup (cod(\varphi))_e^{2^n}, \text{ if } 3i+2 > n \end{cases}$$

At configuration $\mathcal{C}_{3i+2}$:

(a) Object S triggers separation rule (35), creating 2^i new cells 1 having a total of 2^{i+1} cells labelled by 1 from which:
- 2^i cells 1 contain objects $A_{i+2}, B_{i+2}, T_{i+1}$. Moreover, each of them contains a different truth assignment $\sigma_{i,j}$ of the set $\{x_1, \ldots, x_i\}$.
- 2^i cells 1 contain objects $A'_{i+2}, B'_{i+2}, F'_{i+1}$. Moreover, each of them contains a different truth assignment $\tau_{i,j}$ of the set $\{x_1, \ldots, x_i\}$.

(b) Rule (36) produces object α_{3i+3}. Objects `yes` and `no` do not evolve at this transition step.

⋆ If $i+1 < n-1$ (that is, $n > i+2$) rules (19), (22), (23), (27), and (25) produce objects

$$b_{i+2}^{2^{i+1}}, {b'}_{i+2}^{2^{i+1}}, v_{i+2}^{2^{i+1}}, a_{i+2}^{2^{i+1}}, {a'}_{i+2}^{2^{i+1}}, t_1^{2^i}, f_1^{2^i}, \ldots, t_{i+1}^{2^i}, f_{i+1}^{2^i}, q_{1,i+2}^{2^i}, \ldots, q_{i+2,i+2}^{2^i}$$

in cell 2.

⋆ If $i+1 = n-1$ (that is, $n = i+2$) rules (19), (22), (23), and (27) produce objects

$$b_{i+2}^{2^{i+1}}, {b'}_{i+2}^{2^{i+1}}, v_{i+2}^{2^{i+1}}, a_{i+2}^{2^{i+1}}, {a'}_{i+2}^{2^{i+1}}, t_1^{2^i}, f_1^{2^i}, \ldots, t_{i+1}^{2^i}, f_{i+1}^{2^i}$$

in cell 2. Rule (30) erases objects $u_{1,i+1}^{2^i}, \ldots, u_{i+1,i+1}^{2^i}$ from cell 2.

(c) Rule (37) produces object β_{3i+3}. Moreover,

⋆ If $3i+3 \leq n$ (that is, $n > 3i+2$) rule (39) produces $(cod(\varphi))_{d_{i,j,3i+3}}^{2^{3i+3}}$ in cell 3.

⋆ If $n < 3i+2$, then objects from $(cod(\varphi))_e^{2^n}$ do not evolve.

⋆ If $n = 3i+2$, then rule (40) produces $(cod(\varphi))_e^{2^n}$ from $(cod(\varphi))_{d_{i,j,n}}^{2^n}$ in cell labelled by 3.

Thus, the result holds for configuration $\mathcal{C}_{3(i+1)}$.

At configuration $\mathcal{C}_{3(i+1)}$:

(a) Rules (1), (2), (3), (4), (5), (6), (7), and (8) trade objects

$$A_{i+2}, A'_{i+2}, B_{i+2}, B'_{i+2}, T_1, \ldots, T_{i+1}, T'_1, \ldots, T'_{i+1}, F_1, \ldots, F_{i+1}, F'_1, \ldots, F'_{i+1}$$

from cell 1 for objects

$$a_{i+2}, a'_{i+2}, b_{i+2}, b'_{i+2}, f_1, t_1, \ldots, f_{i+1}, t_{i+1}$$

from cell 2. Then, we have 2^{i+1} cells labelled by 1 such that each of them contains objects $a_{i+2}, a'_{i+2}, b_{i+2}, b'_{i+2}$ and also contain a different truth assignment $\epsilon_{i+1,j}$ of the set $\{x_1, \ldots, x_{i+1}\}$.

(b) Rule (36) produces object α_{3i+4}. Objects `yes` and `no` do not evolve at this transition step. After the interchange of objects with cell 1, cell 2 contains objects

$$A_{i+2}^{2^i}, {A'}_{i+2}^{2^i}, B_{i+2}^{2^i}, {B'}_{i+2}^{2^i}, T_{i+1}^{2^i}, {F'}_{i+1}^{2^i}, \sigma_{i,1}, \ldots, \sigma_{i,2^i}, \tau_{i,1}, \ldots, \tau_{i,2^i}$$

- ⋆ If $i+1 < n-1$ (that is, $n > i+2$) then rules (20) and (25) produce objects $y_{i+2}^{2^{i+2}}, r_{1,i+2}^{2^{i+1}}, \ldots, r_{i+2,i+2}^{2^{i+1}}$ in cell 2.
- ⋆ If $i+1 = n-1$ (that is, $n = i+2$) rule (20) produces objects $y_{i+2}^{2^{i+2}}$ in cell 2.

(c) Rule (37) produces object $\beta_{3(i+1)+1}$. Moreover,

- ⋆ If $n > 3(i+1)$, then rule (39) produces $(cod(\varphi))_{d_{i,j,3(i+1)+1}}^{2^{3(i+1)+1}}$ in cell 3.
- ⋆ If $n < 3(i+1)$, then objects from $(cod(\varphi))_e^{2^n}$ do not evolve.
- ⋆ If $n = 3(i+1)$, then rule (40) produces $(cod(\varphi))_e^{2^n}$ from $(cod(\varphi))_{d_{i,j,n}}^{2^n}$ in cell 3.

Hence, the result holds for the configuration $\mathcal{C}_{3(i+1)+1}$.

At configuration $\mathcal{C}_{3(i+1)+1}$:

(a) Rules (9), (10), (11), and (12) produce objects

$$T_{i+2}A_{i+3}, F'_{i+2}A'_{i+3}, F_1F'_1, T_1T'_1, \ldots, F_{i+1}F'_{i+1}, T_{i+1}T'_{i+1}, B_{i+3}, S, B'_{i+3}$$

in cell 1. Specifically, there are 2^{i+1} cells labelled by 1 such that each of them contains objects $A_{i+3}, A'_{i+3}, B_{i+3}, S, B'_{i+3}, T_{i+2}, F'_{i+2}$, and also contains a different truth assignment $\sigma_{i+1,j}$ of the set $\{x_1, \ldots x_{i+1}\}$, as well as an identical copy $\tau_{i+1,j}$ of the set $\{x_1, \ldots x_{i+1}\}$ but for primes.

(b) Rule (36) produces object α_{3i+5}. Objects `yes` and `no` do not evolve at this transition step. Moreover,

- ⋆ If $i+1 < n-1$ (that is, $n > i+2$) then rules (15), (16), (17) and (18) produce objects $c_{i+2}^{2^i}, c_{i+2}^{2^i}, c_{i+2}^{2^i}, c_{i+2}^{2^i}$ (that is, $c_{i+2}^{2^{i+2}}$) in cell 2. Also, rules (31), (32), (33) and (34) erase objects T_i, T'_i, F_i, F'_i from cell 2. Rules (21) and (26) produce objects
$$z_{i+2}^{2^{i+2}}, w_{i+2}^{2^{i+2}}, s_1^{2^{i+1}}, u_{1,i+2}^{2^{i+1}}, \ldots, s_{i+2}^{2^{i+1}}, u_{i+2,i+2}^{2^{i+1}}$$
- ⋆ If $i+1 = n-1$ (that is, $n = i+2$) rules (15), (16), (17), (18), (31), (32), (33) and (34) erase objects
$$A_{i+2}, A'_{i+2}, B_{i+2}, B'_{i+2}, T_i, T'_i, F_i, F'_i$$
from cell 2. Also rule (21) produces object $w_{i+2}^{2^{i+2}} \equiv w_n^{2^n}$.

(c) Rule (37) produces object $\beta_{3(i+1)+2}$ in cell 3. Moreover,

- ⋆ If $n > 3(i+1)+1$, then rule (39) produces $(cod(\varphi))_{d_{i,j,3(i+1)+2}}^{2^{3(i+1)+2}}$ in cell 3.
- ⋆ If $n < 3(i+1)+1$, then objects from $(cod(\varphi))_e^{2^n}$ do not evolve.
- ⋆ If $n = 3(i+1)+1$, then rule (40) produces $(cod(\varphi))_e^{2^n}$ from $(cod(\varphi))_{d_{i,j,n}}^{2^n}$ in cell 3.

Hence, the result holds for configuration $\mathcal{C}_{3(i+1)+2}$.

Then the proof of the theorem completes. □

Theorem 6.2 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots)$ be a computation of the tissue P system $\Pi(\langle m, n\rangle)$.* **At configuration $\mathcal{C}_{3n}$,** *we have the following:*

(a) There are 2^n cells labelled by 1 from which:

- ⋆ 2^{n-1} *cells contain objects* T_n, , A_{n+1}. *Moreover, each of them also contains a different truth assignment* $\sigma_{n-1,j}$ *of the set* $\{x_1, \ldots, x_{n-1}\}$.
- ⋆ 2^{n-1} *cells contain objects* F'_n, , A'_{n+1}. *Moreover, each of them also contains a different truth assignment* $\tau_{n-1,j}$, *of the set* $\{x_1, \ldots, x_{n-1}\}$.

(b) There is a cell labelled by 2. This cell contains objects $E_1^{2^n}$ α_{3n} `yes` `no`.

(c) There is a cell labelled by 3. This cell contains objects β_{3n}, *and* $(cod(\varphi))_e^{2^n}$.

Proof: From Theorem 6.1 for $i = n-1$ we deduce that at configuration $\mathcal{C}_{3(n-1)+2} = \mathcal{C}_{3n-1}$ we have:

- There are 2^{n-1} cells labelled by 1 such that:
  (a) Each of them contains objects $A_{n+1}, A'_{n+1}, B_{n+1}, B'_{n+1}, S, T_n, F'_n$.
  (b) Each of them contains a different truth assignment $\sigma_{n-1,j}$ of the set $\{x_1, \ldots, x_{n-1}\}$ as well as an identical copy $\tau_{n-1,j}$ but for primes.
- There is a cell labelled by 2 which contains objects $w_n^{2^n}, \alpha_{3(n-1)+2} = \alpha_{3n-1}$, `yes`, `no`.
- There is a cell labelled by 3 which contains object $\beta_{3(n-1)+2} = \beta_{3n-1}$, and objects from $(cod(\varphi))_e^{2^n}$ (because $3(n-1)+2 > n$)).

By applying rules $(1, B_{n+1}/\lambda, 0)$ and $(1, B'_{n+1}/\lambda, 0)$, objects B_{n+1} and B'_{n+1} are removed from cell 1. By applying separation rule (35), each cell 1 creates two new cells labelled by 1: one of them containing objects with primes, and the other containing objects without primes. That is, at configuration $\mathcal{C}_{3n}$ we have 2^n cell 1 such that:

(a) 2^{n-1} cells contain objects T_n, A_{n+1}. Moreover, each of them contains a different truth assignment $\sigma_{n-1,j}$ of the set $\{x_1, \ldots, x_{n-1}\}$.
(b) 2^{n-1} cells contain objects F'_n, A'_{n+1}. Moreover, each of them contains a different truth assignment $\tau_{n-1,j}$ of the set $\{x_1, \ldots, x_{n-1}\}$.

Rule (36) produces object α_{3n} in cell 2. Rule $(2, w_n/E_1, 0)$ produces objects $E_1^{2^n}$ in cell 2. Neither objects `yes` or `no` evolve at this transition step. That is,

$$\mathcal{C}_{3n}(2) = \{E_1^{2^n}, \alpha_{3n}, \texttt{yes}, \texttt{no}\}$$

Rule (37) produces object β_{3n} in cell 2. Objects from $(cod(\varphi))_e^{2^n}$ do not evolve at this transition step. That is,

$$\mathcal{C}_{3n}(3) = \{\beta_{3n}\} \cup (cod(\varphi))_e^{2^n}$$

□

Theorem 6.3 *Let* $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots)$ *be a computation of the tissue P system* $\Pi(\langle m, nrangle)$. **At configuration** $\mathcal{C}_{3n+1}$, *we have the following:*

(a) There are 2^n *cells labelled by 1 which contain object* E_1. *Besides,*

- ⋆ 2^{n-1} *of those cells, enumerated by* $(1,1),\dots(1,2^{n-1})$*, contain object* T_n*, and each of them contains a different truth assignment* $\sigma_{n-1,j}$ *of the set* $\{x_1,\dots,x_{n-1}\}$.
- ⋆ 2^{n-1} *of those cells, enumerated by* $(1,2^{n-1}+1),\dots(1,2^n)$*, contain object* F'_n*, and each of them contains a different truth assignment* $\tau_{n-1,j}$ *of the set* $\{x_1,\dots,x_{n-1}\}$.

(b) There is a cell labelled by 2. This cell contains objects $\alpha_{3n+1}\,\mathtt{yes}\,\mathtt{no}\,A_{n+1}^{2^{n-1}}\,A'^{2^{n-1}}_{n+1}$.

(c) There is a cell labelled by 3. This cell contains objects β_{3n+1}*, and* $(cod(\varphi))_e^{2^n}$.

Proof: At configuration $\mathcal{C}_{3n}$:

(a) Rules $(1, A_{n+1}/E_1, 2)$ and $(1, A'_{n+1}/E_1, 2)$ exchange objects A_{n+1}, A'_{n+1} from cell 1 for objects E_1 from cell 2. Hence, there are 2^n cells labelled by 2 each of them containing object E_1. Besides:
- ⋆ 2^{n-1} of those cells, enumerated by $(1,1),\dots(1,2^{n-1})$, contain object T_n, and each of them contains a different truth assignment $\sigma_{n-1,j}$ of the set $\{x_1,\dots,x_{n-1}\}$.
- ⋆ 2^{n-1} of those cells, enumerated by $(1,2^{n-1}+1),\dots(1,2^n)$, contain object F'_n, and each of them contains a different truth assignment $\tau_{n-1,j}$ of the set $\{x_1,\dots,x_{n-1}\}$.

(b) Rule (36) produces object α_{3n+1} in cell 2. Objects `yes` and `no` do not evolve at this transition step. That is,

$$\mathcal{C}_{3n+1}(2) = \{A_{n+1}^{2^{n-1}}, A'^{2^{n-1}}_{n+1}, \alpha_{3n+1}, \mathtt{yes}, \mathtt{no}\}$$

(c) Rule (37) produces object β_{3n+1} in cell 2. Objects from $(cod(\varphi))_e^{2^n}$ do not evolve at this transition step. That is,

$$\mathcal{C}_{3n+1}(3) = \{\beta_{3n+1}\} \cup (cod(\varphi))_e^{2^n}$$

□

In this way, the **generating stage** finishes at step $3n+1$ and the **checking stage** would start at the next step.

Theorem 6.4 *Let* $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ *be a computation of the tissue P system* $\Pi(\langle m,n\rangle)$. **At configuration** $\mathcal{C}_{(3n+1)+1}$*, the following holds:*

(a) There are 2^n *cells labelled by 1. Besides,*
- ⋆ *If the truth assignment* $\sigma_{n,s}$ *associated with a cell* $(1,t)$*, where* $1 \le t \le 2^n$*, makes the clause* C_1 *true, then*
  - *If* $1 \le t \le 2^{n-1}$ *then it contains* $e_{i,1} + (\sigma_{n,s} - \{T_i\})$*, for some* i *such that* $x_i \in C_1$*, or it contains* $\overline{e}_{i,1} + (\sigma_{n,s} - \{F_i\})$*, for some* i *such that* $\neg x_i \in C_1$.

- *If* $2^{n-1}+1 \leq t \leq 2^n$ *then it contains* $e_{i,1} + (\tau_{n,s} - \{T'_i\})$*, for some* i *such that* $x_i \in C_1$*, or it contains* $\overline{e}_{i,1} + (\tau_{n,s} - \{F'_i\})$*, for some* i *such that* $\neg x_i \in C_1$*.*

⋆ *If the truth assignment* $\sigma_{n,s}$ *associated with a cell* $(1,t)$*, where* $1 \leq t \leq 2^n$*, makes clause* C_1 *false , then their contents coincide with the corresponding contents in the previous configuration* $\mathcal{C}_{3n+1}$*. In particular, that cell does not contain any object* $e_{i,1}$ *nor* $\overline{e}_{i,1}$*.*

(b) There is a cell labelled by 2. This cell contains objects $\alpha_{(3n+1)+1}, \texttt{yes}, \texttt{no}$*.*

(c) There is a cell labelled by 3. This cell contains:

⋆ k_1 *copies of object* E_1*, being* k_1 *the number of truth assignments making clause* C_1 *of* φ *true.*

⋆ $(cod(\varphi))^{2^n}_{e,>1}$ *representing* 2^n *copies of the objects* $e_{i,j}$ *and* $\overline{e}_{i,j}$ *such that* $j > 1$ *and* $x_i \in C_j$ *in the first case, and* $\neg x_i \in C_j$ *in the second one.*

⋆ *Object* $\beta_{(3n+1)+1}$*.*

⋆ *Some irrelevant objects of the type* T_i, T'_i, F_i, F'_i *that will dissappear at the next step.*

⋆ *Some irrelevant objects of the type* $e_{i,1}, \overline{e}_{i,1}$ *that will not be considered anymore.*

Proof: At configuration $\mathcal{C}_{3n+1}$:

(a) Rules of type (41) are applied to cells labelled by 1 trading objects

$$E_1, T_i, T'_i, F_i, F'_i$$

from cell 1 for objects $e_{i,1}, \overline{e}_{i,1}$ from cell 3 according to the following conditions: if a cell 1 encodes a truth assignment making clause C_1 true, then it replaces objects E_1T_i or $E_1T'_i$ (respectively, objects E_1F_i or $E_1F'_i$) by objects $e_{i,1}$ (respectively, objects $\overline{e}_{i,1}$), if $x_i \in C_1$ (respectively, if $\neg x_i \in C_1$). This transition step is non-deterministic because object E_1 can choose different truth values T, T', F or F' from cells labelled by 1 making clause C_1 true.

Let us suppose that the truth assignment $\sigma_{n,s}$ associated with a cell $(1,t)$ $(1 \leq t \leq 2^n)$ makes the clause C_1 true (on the contrary, rule (41) is not applicable to configuration $\mathcal{C}_{3n+1}$, so $\mathcal{C}_{(3n+1)+1}(1,t) = \mathcal{C}_{(3n+1)}(1,t)$).

⋆ Case 1: $1 \leq t \leq 2^{n-1}$.
If $x_i \in C_1$ then objects E_1T_i from cell $(1,t)$ are replaced by object $e_{i,1}$ from cell 3. So, the contents of cell $(1,t)$ is $e_{i,1} + (\sigma_{n,s} - \{T_i\})$.
If $\neg x_i \in C_1$ then objects E_1F_i from cell $(1,t)$ are replaced by object $\overline{e}_{i,1}$ from cell 3. So, the contents of cell $(1,t)$ is $\overline{e}_{i,1} + (\sigma_{n,s} - \{F_i\})$.

⋆ Case 2: $2^{n-1}+1 \leq t \leq 2^n$.
If $x_i \in C_1$ then objects $E_1T'_i$ from cell $(1,t)$ are exchanged for object $e_{i,1}$ from cell 3. So, the contents of cell $(1,t)$ is $e_{i,1} + (\tau_{n,s} - \{T'_i\})$.
If $\neg x_i \in C_1$ then objects $E_1F'_i$ from cell $(1,t)$ are exchanged for object $\overline{e}_{i,1}$ from cell 3. So, the contents of cell $(1,t)$ is $\overline{e}_{i,1} + (\tau_{n,s} - \{F'_i\})$.

(b) Rules (15) and (16) remove objects $A^{2^{n-1}}_{n+1}, {A'}^{2^{n-1}}_{n+1}$ from cell 2. Rule (36) produces object α_{3n+2}. Hence, $\mathcal{C}_{3n+2}(2) = \{\alpha_{3n+2}, \texttt{yes}, \texttt{no}\}$.

(c) Rule (37) produces object β_{3n+2} in cell 3 which also contains:
- ⋆ A number k_1 of copies of object E_1 equal to the number of truth assignment making clause C_1 true.
- ⋆ $(cod(\varphi))^{2^n}_{e,>1}$.
- ⋆ Garbagge objects T_i, T'_i, F_i, F'_i which will be removed at the next step.
- ⋆ Garbagge objects $e_{i,1}, \overline{e}_{i,1}$ which will not be considered anymore.

□

Theorem 6.5 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(\langle m, n\rangle)$. For every j $(1 \leq j \leq m-1)$ we have:*

(1) At configuration $\mathcal{C}_{(3n+1)+2j}$*, the following holds:*

(a) There are 2^n cells labelled by 1. Besides,
- ⋆ *If the truth assignment $\sigma_{n,s}$ associated with a cell $(1,t)$, where $1 \leq t \leq 2^n$, makes $C_1 \wedge \cdots \wedge C_j$ true, then it contains object E_{j+1}. Moreover,*
  - *If $1 \leq t \leq 2^{n-1}$ then it contains object T_i, for some i such that $x_i \in C_j$, or it contains object F_i, for some i such that $\neg x_i \in C_j$. Besides, objects T_i and F_i of that cell 1 at configuration $\mathcal{C}_{(3n+1)+2j-1}$ remain at configuration $\mathcal{C}_{(3n+1)+2j}$.*
  - *If $2^{n-1}+1 \leq t \leq 2^n$ then it contains T_i, for some i such that $x_i \in C_j$, or it contains F_i, for some i such that $\neg x_i \in C_j$. Besides, objects T'_i and F'_i of that cell 1 at configuration $\mathcal{C}_{(3n+1)+2j-1}$ remain at configuration $\mathcal{C}_{(3n+1)+2j}$.*
- ⋆ *If the truth assignment $\sigma_{n,s}$ associated with a cell $(1,t)$, where $1 \leq t \leq 2^n$, makes $C_1 \wedge \cdots \wedge C_j$ false, then their contents coincide with the corresponding contents in the previous configuration $\mathcal{C}_{(3n+1)+2j-1}$. In particular, that cell does not contain object E_{j+1}.*

(b) There is a cell labelled by 2. This cell contains objects $\alpha_{(3n+1)+2j}$, `yes`, `no`.

(c) There is a cell labelled by 3. This cell contains:
- ⋆ *$(cod(\varphi))^{2^n}_{e,>j}$ representing 2^n copies of the objects $e_{i,j'}$ and $\overline{e}_{i,j'}$ such that $j' > j$ and $x_i \in C_{j'}$ in the first case, and $\neg x_i \in C_{j'}$ in the second one.*
- ⋆ *Object $\beta_{(3n+1)+2j}$.*
- ⋆ *Some irrelevant objects of the type $e_{i,j'}, \overline{e}_{i,j'}$, with $1 \leq j' \leq j$ that will not be considered anymore.*

(2) At configuration $\mathcal{C}_{(3n+1)+2j+1}$*, the following holds:*

(a) There are 2^n cells labelled by 1. Besides,
- ⋆ *If the truth assignment $\sigma_{n,s}$ associated with a cell $(1,t)$, where $1 \leq t \leq 2^n$, makes $C_1 \wedge \cdots \wedge C_{j+1}$ true, then*
  - *If $1 \leq t \leq 2^{n-1}$ then it contains $e_{i,j+1} + (\sigma_{n,s} - \{T_i\})$, for some i such that $x_i \in C_{j+1}$, or it contains $\overline{e}_{i,j+1} + (\sigma_{n,s} - \{F_i\})$, for some i such that $\neg x_i \in C_{j+1}$.*
  - *If $2^{n-1}+1 \leq t \leq 2^n$ then it contains $e_{i,j+1} + (\tau_{n,s} - \{T'_i\})$, for some i such that $x_i \in C_{j+1}$, or it contains $\overline{e}_{i,j+1} + (\tau_{n,s} - \{F'_i\})$, for some i such that $\neg x_i \in C_{j+1}$.*

- ⋆ *If the truth assignment $\sigma_{n,s}$ associated with a cell $(1,t)$, where $1 \leq t \leq 2^n$, makes $C_1 \wedge \cdots \wedge C_{j+1}$ false, then their contents coincide with the corresponding contents in the previous configuration $\mathcal{C}_{(3n+1)+2j}$. In particular, that cell does not contain any object $e_{i,j+1}$ nor $\overline{e}_{i,j+1}$.*

(b) There is a cell labelled by 2. This cell contains objects $\alpha_{(3n+1)+2j+1}, \texttt{yes}, \texttt{no}$.

(c) There is a cell labelled by 3. This cell contains:

- ⋆ *k_{j+1} copies of object E_{j+1}, being k_{j+1} the number of truth assignment making clauses $C_1, \ldots, C_{j+1}$ of φ true.*
- ⋆ *$(cod(\varphi))^{2^n}_{e,>(j+1)}$ representing 2^n copies of the objects $e_{i,j'}$ and $\overline{e}_{i,j'}$ such that $j' > j+1$ and $x_i \in C_{j'}$ in the first case, and $\neg x_i \in C_{j'}$ in the second one.*
- ⋆ *Object $\beta_{(3n+1)+2j+1}$.*
- ⋆ *Some irrelevant objects of the type T_i, T_i', F_i, F_i' that will dissappear at the next step.*
- ⋆ *Some irrelevant objects of the type $e_{i,j'}, \overline{e}_{i,j'}$ with $1 \leq j' \leq j+1$ that will not be considered anymore.*

Proof: By induction on j. Let us start analyzing the basic case $j = 1$.

At **configuration** $\mathcal{C}_{(3n+1)+1}$:

(a) Rule (42) produces objects T_iE_2 in a cell 1 which contains object $e_{i,1}$, and produces objects F_iE_2 in a cell 1 which contains object $\overline{e}_{i,1}$. So, there are 2^n cells labelled by 1 such that:

- ⋆ If the truth assignment associated with a cell $(1,t)$ makes clause C_1 true, then it contains objects E_2. Moreover, it contains object T_i for some i such that $x_i \in C_1$, or object F_i for some i such that $\overline{x}_i \in C_1$. Besides, the remaining objects at configuration $\mathcal{C}_{3n+2}$ stay unchanged at this transition step.
- ⋆ If the truth assignment associated with a cell $(1,t)$ makes clause C_1 false, then their contents coincide with the corresponding contents of the previous configuration $\mathcal{C}_{(3n+1)+1}$.

(b) Only rule (36) is applicable to cell 2 at configuration $\mathcal{C}_{3n+2}$. So,

$$\mathcal{C}_{3n+3}(2) = \{\alpha_{(3n+1)+2}, \texttt{yes}, \texttt{no}\}$$

(c) Rule (37) produces object β_{3n+3} in cell 3. Rules (44) and (45) remove objects $E_1, T_i, T_i', F_i, F_i'$ from cell 3.

At **configuration** $\mathcal{C}_{(3n+1)+2}$:

(a) If the truth assignment $\sigma_{n,s}$ associated with a cell $(1,t)$ makes clause C_2 true, then

- ⋆ If $1 \leq t \leq 2^{n-1}$, rules (41) replace objects T_iE_2 from cell 1 by objects $e_{i,2}$ from cell 3, for some i such that $x_i \in C_2$, or objects F_iE_2 from cell 1 by objects $\overline{e}_{i,2}$ from cell 3, for some i such that $\overline{x}_i \in C_2$. Hence, such a cell 1 contains

$$\begin{cases} e_{i,2} + (\sigma_{n,s} - \{T_i\}), \text{ if objects } T_iE_2 \text{ have been exchanged} \\ e_{i,2} + (\sigma_{n,s} - \{F_i\}), \text{ if objects } F_iE_2 \text{ have been exchanged} \end{cases}$$

⋆ If $2^{n-1}+1 \leq t \leq 2^n$, rule (41) either replaces objects T_iE_2 or objects T'_iE_2 by objects $e_{i,2}$ from cell 3, for some i such that $x_i \in C_2$, either objects F_iE_2 or objects F'_iE_2 by objects $\overline{e}_{i,2}$ from cell 3, for some i such that $\overline{x}_i \in C_2$. Hence, such a cell 1 contains

$$\begin{cases} e_{i,2} + (\tau_{n,s} - \{T_i\}), \text{ if objects } T_iE_2 \text{ have been exchanged} \\ e_{i,2} + (\tau_{n,s} - \{T'_i\}), \text{ if objects } T'_iE_2 \text{ have been exchanged} \\ e_{i,2} + (\tau_{n,s} - \{F_i\}), \text{ if objects } F_iE_2 \text{ have been exchanged} \\ e_{i,2} + (\tau_{n,s} - \{F'_i\}), \text{ if objects } F'_iE_2 \text{ have been exchanged} \end{cases}$$

(b) Only rule (36) is applicable to cell 2 at configuration $\mathcal{C}_{(3n+1)+2}$. So,

$$\mathcal{C}_{3n+4}(2) = \{\alpha_{(3n+1)+3}, \texttt{yes}, \texttt{no}\}$$

(c) Also rule (37) is applicable to cell 3 producing object β_{3n+4}. Then, cell 3 contains:
- k_2 copies of object E_2, being k_2 the number of truth assignment making clauses C_1, C_2 of φ true.
- $(cod(\varphi))^{2^n}_{e,>2}$ representing 2^n copies of the objects $e_{i,j'}$ and $\overline{e}_{i,j'}$ such that $j' > 2$ and $x_i \in C_{j'}$ in the first case, and $\neg x_i \in C_{j'}$ in the second one.
- Object $\beta_{(3n+1)+3}$.
- Garbagge objects of the type T_i, T'_i, F_i, F'_i that will dissappear at the next step.
- Garbagge objects of the type $e_{i,j'}, \overline{e}_{i,j'}$ with $1 \leq j' \leq j+1$ that will not be considered anymore.

By induction hypothesis, let j such that $1 \leq j < m-1$ and let us the result holds for j. Let us see that the result also holds for $j+1$.

At **configuration** $\mathcal{C}_{(3n+1)+2j+1}$:

(a) Rule (42) produces objects T_iE_{j+2} in a cell 1 which contains object $e_{i,j}$, and produces objects F_iE_{j+2} in a cell 1 which contains object $\overline{e}_{i,j}$. So, there are 2^n cells labelled by 1 such that:
⋆ If the truth assignment associated with a cell $(1,t)$ makes $C_1 \wedge \cdots \wedge C_{j+2}$ true, then it contains objects E_{j+2}. Moreover, it contains object T_i for some i such that $x_i \in C_{j+2}$, or object F_i for some i such that $\overline{x}_i \in C_{j+2}$. Besides, the remaining objects at configuration $\mathcal{C}_{(3n+1)+2j+1}$ stay unchanged at this transition step.
⋆ If the truth assignment associated with a cell $(1,t)$ makes $C_1 \wedge \cdots \wedge C_{j+2}$ false, then their contents coincide with the corresponding contents of the previous configuration $\mathcal{C}_{(3n+1)+2j+1}$.

(b) Only rule (36) is applicable to cell 2 at configuration $\mathcal{C}_{(3n+1)+2j+1}$. So,

$$\mathcal{C}_{(3n+1)+2j+2}(2) = \{\alpha_{(3n+1)+2j+2}, \texttt{yes}, \texttt{no}\}$$

(c) Rule (37) produces object $\beta_{(3n+1)+2j+2}$ in cell 3. Rules (44) and (45) remove objects $E_1, T_i, T_i', F_i, F_i'$ from cell 3.

At **configuration** $\mathcal{C}_{(3n+1)+2j+2}$:

(a) If the truth assignment $\sigma_{n,s}$ associated with a cell $(1,t)$ makes $C_1 \wedge \cdots \wedge C_{j+2}$ true, then

- If $1 \leq t \leq 2^{n-1}$, rules (41) replace objects $T_i E_{j+2}$ from cell 1 by objects $e_{i,j+2}$ from cell 3, for some i such that $x_i \in C_{j+2}$, or objects $F_i E_{j+2}$ from cell 1 by objects $\overline{e}_{i,j+2}$ from cell 3, for some i such that $\overline{x}_i \in C_{j+2}$. Hence, such a cell 1 contains

$$\begin{cases} e_{i,j+2} + (\sigma_{n,s} - \{T_i\}) \text{ , if objects } T_i E_{j+2} \text{ have been exchanged} \\ e_{i,j+2} + (\sigma_{n,s} - \{F_i\}) \text{ , if objects } F_i E_{j+2} \text{ have been exchanged} \end{cases}$$

- If $2^{n-1} + 1 \leq t \leq 2^n$, rules (41) either replace objects $T_i E_{j+2}$ or objects $T_i' E_{j+2}$ from cell 1 by objects $e_{i,j+2}$ from cell 3, for some i such that $x_i \in C_{j+2}$, either objects $F_i E_{j+2}$ or objects $F_i' E_{j+2}$ from cell 1 by objects $\overline{e}_{i,j+2}$ from cell 3, for some i such that $\overline{x}_i \in C_{j+2}$. Hence, such a cell 1 contains

$$\begin{cases} e_{i,j+2} + (\tau_{n,s} - \{T_i\}), \text{ if objects } T_i E_{j+2} \text{ have been exchanged} \\ e_{i,j+2} + (\tau_{n,s} - \{T_i'\}), \text{ if objects } T_i' E_{j+2} \text{ have been exchanged} \\ e_{i,j+2} + (\tau_{n,s} - \{F_i\}), \text{ if objects } F_i E_{j+2} \text{ have been exchanged} \\ e_{i,j+2} + (\tau_{n,s} - \{F_i'\}), \text{ if objects } F_i' E_{j+2} \text{ have been exchanged} \end{cases}$$

(b) Only rule (36) is applicable to cell 2 at configuration $\mathcal{C}_{(3n+1)+2j+2}$. So,

$$\mathcal{C}_{(3n+1)+2j+3}(2) = \{\alpha_{(3n+1)+2j+3}, \texttt{yes}, \texttt{no}\}$$

(c) Also rule (37) is applicable to cell 3 producing object $\beta_{(3n+1)+2j+3}$. Then, cell 3 contains:

- k_{j+2} copies of object E_{j+2}, being k_{j+2} the number of truth assignment making $C_1 \wedge \cdots \wedge C_{j+2}$ true.
- $(cod(\varphi))^{2^n}_{e,>j+2}$ representing 2^n copies of the objects $e_{i,j'}$ and $\overline{e}_{i,j'}$ such that $j' > j+2$ and $x_i \in C_{j'}$ in the first case, and $\neg x_i \in C_{j'}$ in the second one.
- Object $\beta_{(3n+1)+2j+3}$.
- Garbagge objects of the type T_i, T_i', F_i, F_i' that will dissappear at the next step.
- Garbagge objects of the type $e_{i,j'}, \overline{e}_{i,j'}$ with $1 \leq j' \leq j+2$ that will not be considered anymore.

Hence, the result is also true for $j+1$. Then the proof of the theorem completes. □

Theorem 6.6 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(\langle m, n \rangle)$.* **At configuration** $\mathcal{C}_{(3n+1)+2m}$*, the following holds:*

(a) There are 2^n cells labelled by 1, and the formula φ is satisfiable if and only if there is, at least, one of such cell which contains object E_{m+1}.
(b) There is a cell labelled by 2. This cell contains objects $\alpha_{(3n+1)+2m}, \texttt{yes}, \texttt{no}$.
(c) There is a cell labelled by 3. This cell contains object $\beta_{(3n+1)+2m}$, and some irrelevant objects of the type $e_{i,j'}, \overline{e}_{i,j'}$ with $1 \leq j' \leq m$ that will not be considered anymore.

Proof: From Theorem 6.5, at configuration $\mathcal{C}_{(3n+1)+2(m-1)+1}$ we have:
(a) There are 2^n cells labelled by 1 each such that:
- ⋆ Let $\sigma_{n,s}$ a truth assignment associated with a cell $(1,t)$, where $1 \leq t \leq 2^n$, making $C_1 \wedge \cdots \wedge C_m$ true. Then
  - If $1 \leq t \leq 2^{n-1}$ then it contains $e_{i,m} + (\sigma_{n,s} - \{T_i\})$, for some i such that $x_i \in C_m$, or $\overline{e}_{i,m} + (\sigma_{n,s} - \{F_i\})$, for some i such that $\neg x_i \in C_m$.
  - If $2^{n-1}+1 \leq t \leq 2^n$ then it contains $e_{i,m} + (\tau_{n,s} - \{T'_i\})$, for some i such that $x_i \in C_m$, or $\overline{e}_{i,m} + (\tau_{n,s} - \{F'_i\})$, for some i such that $\neg x_i \in C_m$.
- ⋆ Let $\sigma_{n,s}$ a truth assignment associated with a cell $(1,t)$, where $1 \leq t \leq 2^n$, making $C_1 \wedge \cdots \wedge C_m$ false. Then their contents coincide with the corresponding contents in the previous configuration $\mathcal{C}_{(3n+1)+2(m-1)}$. In particular, that cell does not contain any object $e_{i,m}$ nor $\overline{e}_{i,m}$.

(b) There is a cell labelled by 2 which contains objects $\alpha_{(3n+1)+2(m-1)+1}, \texttt{yes}, \texttt{no}$.
(c) There is a cell labelled by 3 which contains object $\beta_{(3n+1)+2(m-1)+1}$, and:
- k_m copies of object E_m, being k_m the number of truth assignments making clauses $C_1, \ldots, C_m$ true, that is, k_m is the number of truth assignment making true the formula φ.
- Some irrelevant objects of the type T_i, T'_i, F_i, F'_i that will dissappear at the next step.
- Some irrelevant objects of the type $e_{i,j'}, \overline{e}_{i,j'}$ with $1 \leq j' \leq m$ that will not be considered anymore.

Then

(a) Rule (43) produces objects E_{m+1} in every cell 1 which encodes a truth assignment making the formula φ true. Moreover, if a cell labelled by 1 encodes a truth assignment making the formula φ false, then it does not contain object E_{m+1}.
(b) Rule (36) produces object $\alpha_{(3n+1)+2m}$ in cell 2. Thus,
$$\mathcal{C}_{(3n+1)+2m}(2) = \{\alpha_{(3n+1)+2m}, \texttt{yes}, \texttt{no}\}$$
(c) Rules (44) and (45) remove objects $E_{m+1}, T_i, T'_i, F_i, F'_i$ from cell 3. In addition, rule (37) is applicable to cell 3 producing object $\beta_{(3n+1)+2m}$. Cell 3 also contains irrelevant objects of the type $e_{i,j'}, \overline{e}_{i,j'}$, with $1 \leq j' \leq m$, that appear at the previous configuration. □

Theorem 6.7 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots)$ be a computation of the tissue P system $\Pi(\langle m, n\rangle)$.* **At configuration** $\mathcal{C}_{(3n+1)+2m+1}$*, the following holds:*

(a) There are 2^n cells labelled by 1. Besides,

- ⋆ *If the formula φ is satisfiable, then there is one (and only one) cell labelled by 1 which contains objects $\alpha_{(3n+1)+2m}, \texttt{yes}$.*
- ⋆ *If the formula φ is not satisfiable, then their contents coincide with the contents in the previous configuration $\mathcal{C}_{(3n+1)+2m}$.*

(b) There is a cell labelled by 2. Besides,

- ⋆ *If the formula φ is satisfiable, then it contains objects $E_{m+1}, \texttt{no}$.*
- ⋆ *If the formula φ is not satisfiable, then it contains objects $\alpha_{(3n+1)+2m}, \texttt{yes}, \texttt{no}$.*

(c) There is a cell labelled by 3. The contents of this cell is the same that in the previous configuration $\mathcal{C}_{(3n+1)+2m}$, except object $\beta_{(3n+1)+2m}$ that evolves to $\beta_{(3n+1)+2m+1}$.

Proof: At configuration $\mathcal{C}_{(3n+1)+2m+1}$:

(a) There are 2^n cells labelled by 1, and

- ⋆ If the formula φ is satisfiable, then there are cells labelled by 1 which contain objects E_{m+1}. Then, one (and only one) of these objects can be used to apply rule (46), allowing its trade for objects $\alpha_{(3n+1)+2m}, \texttt{yes}$ from cell 2.
- ⋆ If the formula φ is not satisfiable, then their contents coincide with the contents in the previous configuration $\mathcal{C}_{(3n+1)+2m}$. In particular, rule (46) can not be applied to any cell labelled by 1, because any such cell encodes a truth assignment making the formula φ true.

(b) There is a cell labelled by 2 such that

- ⋆ If the formula φ is satisfiable, then

$$\mathcal{C}_{(3n+1)+2m+1}(2) = \{E_{m+1}, \texttt{no}\}$$

- ⋆ If the formula φ is not satisfiable, then no rule of the system is applicable to that cell 2. Therefore,

$$\mathcal{C}_{(3n+1)+2m+1} = \{\alpha_{(3n+1)+2m}, \texttt{yes}, \texttt{no}\}$$

(c) There is a cell labelled by 3. Only rule (37) is applicable at this cell and produces object $\beta_{(3n+1)+2m+1}$.

□

Theorem 6.8 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(\langle m, n\rangle)$.* **At configuration $\mathcal{C}_{(3n+1)+2m+2}$,** *the following holds:*

(a) There are 2^n cells labelled by 1. Besides,

- ⋆ *If the formula φ is satisfiable, then there is one (and only one) cell labelled by 1 which contains objects $\alpha_{(3n+1)+2m}$ and $\beta_{(3n+1)+2m+1}$.*
- ⋆ *If the formula φ is not satisfiable, then their contents coincide with the contents in the previous configuration $\mathcal{C}_{(3n+1)+2m+1}$.*

(b) There is a cell labelled by 2. Besides,

- ⋆ *If the formula φ is satisfiable, their contents coincide with the contents in the previous configuration $\mathcal{C}_{(3n+1)+2m+1}$.*
- ⋆ *If the formula φ is not satisfiable, then it contains objects* $\mathtt{yes}, \mathtt{no}$, $\beta_{(3n+1)+2m+1}$.

(c) There is a cell labelled by 3. Besides,

- ⋆ *If the formula φ is satisfiable, then it contains object* $\mathtt{yes}$.
- ⋆ *If the formula φ is not satisfiable, then it contains object $\alpha_{(3n+1)+2m}$.*

Proof: At configuration $\mathcal{C}_{(3n+1)+2m+1}$:

(a) There are 2^n cells labelled by 1, and

- ⋆ If the formula φ is satisfiable, there is one (and only one) such cell 1 which contains objects $\alpha_{(3n+1)+2m}, \mathtt{yes}$. By applying rule 47, object $\mathtt{yes}$ from such cell is traded for object $\beta_{(3n+1)+2m+1}$ from cell 3. Thus, there is one (and only one) cell 1 which contains objects $\alpha_{(3n+1)+2m}$ and $\beta_{(3n+1)+2m+1}$.
- ⋆ If the formula φ is not satisfiable, then their contents coincide with the contents at the previous configuration $\mathcal{C}_{(3n+1)+2m+1}$. In particular, rule (47) cannot be applied to any cell labelled by 1.

(b) There is a cell labelled by 2. This cell verifies:

- ⋆ If the formula φ is satisfiable, then any rule is applicable to such cell. Therefore,
$$\mathcal{C}_{(3n+1)+2m+2}(2) = \{E_{m+1}, \mathtt{no}\}$$
- ⋆ If the formula φ is not satisfiable, then rule (48) is applicable allowing the exchange of object $\alpha_{(3n+1)+2m}$ from cell 2 for object $\beta_{(3n+1)+2m+1}$ from cell 3. Hence,
$$\mathcal{C}_{(3n+1)+2m+2}(2) = \{\beta_{(3n+1)+2m+1}, \mathtt{yes}, \mathtt{no}\}$$

(c) There is a cell labelled by 3. This cell verifies:

- ⋆ If the formula φ is satisfiable, then rule (47) produces object $\mathtt{yes}$ in this cell.
- ⋆ If the formula φ is not satisfiable, then rule (48) produces object $\alpha_{(3n+1)+2m}$ in this cell.

□

Theorem 6.9 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(\langle m, n\rangle)$.* **At configuration $\mathcal{C}_{(3n+1)+2m+3}$,** *the following holds:*

(a) If the formula φ is satisfiable, then $\mathtt{yes} \in \mathcal{C}_{(3n+1)+2m+3}(0)$.
(b) If the formula φ is not satisfiable, then $\mathtt{no} \in \mathcal{C}_{(3n+1)+2m+3}(0)$.
(c) The configuration $\mathcal{C}_{(3n+1)+2m+3}$ is a halting configuration.

Proof:

(a) Let us suppose that formula φ is satisfiable. Then no rule is applicable to any cell labelled by 1 at configuration $\mathcal{C}_{(3n+1)+2m+2}$. Bearing in mind that $\mathcal{C}_{(3n+1)+2m+2}(2) = \{E_{m+1}, \mathtt{no}\}$, and $\mathtt{yes} \in \mathcal{C}_{(3n+1)+2m+2}(3)$, only rule (50) is applicable to configuration $\mathcal{C}_{(3n+1)+2m+2}$. Hence, $\mathtt{yes} \in \mathcal{C}_{(3n+1)+2m+3}(0)$.

(b) Let us suppose that formula φ is not satisfiable. Then no rule is applicable to any cell labelled by 1 at configuration $\mathcal{C}_{(3n+1)+2m+2}$. Bearing in mind that $\mathcal{C}_{(3n+1)+2m+2}(2) = \{\beta_{3n+1+2m+1}, \texttt{yes}, \texttt{no}\}$, and $\alpha_{3n+1+2m} \in \mathcal{C}_{(3n+1)+2m+2}(3)$, only rule (49) is applicable to configuration $\mathcal{C}_{(3n+1)+2m+2}$. Hence, $\texttt{no} \in \mathcal{C}_{(3n+1)+2m+3}(0)$.
(c) From (a) and (b), it is easy to check that no rule of the system is applicable to configuration $\mathcal{C}_{(3n+1)+2m+3}$.

□

Corollary 6.10 *The family* $\mathbf{\Pi}$ *is polynomially bounded.*

Proof: From Theorem 6.9 we deduce that any computation $\mathcal{C}$ of the tissue P system $\Pi(\langle m, n\rangle)$ spends $(3n+1)+2m+3 = 3n+2m+4$ transition steps exactly. □

6.3 Computational Efficiency of TSC(3)

The family of tissue P systems with cell separation constructed in Section 5 verifies the following:

(a) The defined family $\mathbf{\Pi}$ is *consistent*, in the sense that all systems of the family are recognizer tissue P systems with cell separation: (1) the working alphabet Γ has two distinguished objects `yes` and `no`, at least one copy of them present in some initial multisets but none of them are present in $\mathcal{E}$; (2) the output region i_{out} is the environment; (3) all computations halt; and (4) if $\mathcal{C}$ is a computation of a system, then either object `yes` or object `no` (but not both) has been released into the environment, and only at the last step of the computation. Besides, these systems use communication rules with length at most 3.
(b) The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines (Subsection 5.1).
(c) (cod, s) is a pair of polynomial–time computable functions.
(d) The family $\mathbf{\Pi}$ is polynomially bounded with regard to $(\texttt{SAT}, cod, s)$ (Corollary 5.10).
(e) The family $\mathbf{\Pi}$ is sound and complete with regard to $(\texttt{SAT}, cod, s)$ (Subsection 5.2).

Therefore, according to Definition 1, the uniform family $\mathbf{\Pi}$ of tissue P systems constructed in Section 5 solve the `SAT` problem in polynomial time with respect to the number of variables and the number of clauses.

Hence, we have the following result:

Theorem 6.11 $\texttt{SAT} \in \mathbf{PMC}_{TSC(3)}$.

Corollary 6.12 $\mathbf{NP} \cup \mathbf{co-NP} \subseteq \mathbf{PMC}_{TSC(3)}$.

Proof: It suffices to notice that the `SAT` problem is **NP**-complete, $\texttt{SAT} \in \mathbf{PMC}_{TSC(3)}$, and this complexity class is closed under polynomial-time reduction and under complement.

□

7 Conclusions and Future Work

The space-time tradeoff method is used to efficiently solve computationally hard problems in the framework of *Membrane Computing*. The efficiency of tissue P systems with cell division for solving **NP**-complete problems has been previously studied [4, 5, 20]. Cell division rules allow the duplication of all objects in the new created cells except the object that activate the cell division operation. Therefore, the cell division can be used to generate an exponential workspace, expressed in terms of the number of cells and the number of objects, in linear time.

In the framework of tissue P systems with cell division, the length of communication rules provide a frontier for the tractability of decision problems. In [8] the limitation on the efficiency of tissue P systems with cell division and communication rules of length 1 it has been established that only tractable problems can be solved efficiently in that framework. Nevertheless, in [5] a linear time solution to `Vertex Cover` problem by using a family of tissue P systems with cell division and communication rules of length at most 3 has been provided. Hence, in tissue P systems with cell division, passing from communication rules of length 1 to communication rules of length at most 3 amounts to passing from non–efficiency to efficiency, assuming that $\mathbf{P} \neq \mathbf{NP}$.

Recently [15], cell separation rules have been introduced into tissue P systems, inspired by the cellular fission, and its computational efficiency was investigated. This kind of rules allows the creation of two new cells from one cell although there is no replication of objects between the new cells, that is, the contents of the cell is distributed between the new created cells, except the object triggering the rule which is consumed. Therefore, by using cell separation it is possible to construct an exponential workspace, expressed only in terms of the number of cells, in linear time. In [15] two important results were obtained in that framework: (a) only tractable problems can be efficiently solved by using cell separation and communication rules with length at most 1, and (b) a uniform and linear time solution to the `SAT` problem by using cell separation and communication rules with length at most 8 was presented.

In this paper, the previous result has been improved by showing a family of tissue P systems with cell separation and communication rules with length at most 3, solving the `SAT` problem in a uniform way and linear time. Hence, with regard to tissue P systems with cell separation, a similar result concerning the frontier of tractability can be formulated in the new framework: by using families of tissue P systems with cell separation, passing from communication rules of length 1 to communication rules of length at most 3, amounts to passing from non–efficiency to efficiency, assuming that $\mathbf{P} \neq \mathbf{NP}$. It is worth to highlight that separation rules seem weaker than division rules from the point of view of computational complexity.

Next, we propose several open problems related to the efficiency of tissue P systems:

(a) What is the computational efficiency of tissue P systems with cell separation or with cell division, and communication rules with length at most 2 are allowed?
(b) What happens if only symport (respectively, only antiport) rules are allowed in tissue P systems with cell division or cell separation?
(c) In [4] tissue P systems with cell division and without environment were introduced, that is, tissue P systems where the alphabet $\mathcal{E}$ of the environment is empty. In this kind of P systems there are no objects appearing in the system in arbitrary copies each. What is the relationship between the polynomial complexity classes of tissue P systems with cell division (respectively, with cell separation) and the corresponding tissue P systems without environment?

Acknowledgements

The work of the first author was supported by Project TIN2009-13192 of the Ministerio de Ciencia e Innovación of Spain and Project of Excellence with *Investigador de Reconocida Valía*, from Junta de Andalucía, grant P08 – TIC 04200. The work of the second author was supported by the Silesian University in Opava under the Student Funding Scheme, project no SGS/7/2011.

This work was also supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

References

1. Alhazov, A., Freund, R. and Oswald, M. Tissue P Systems with Antiport Rules ans Small Numbers of Symbols and Cells. *Lecture Notes in Computer Science* **3572**, (2005), 100–111.
2. Bernardini, F. and Gheorghe, M. Cell Communication in Tissue P Systems and Cell Division in Population P Systems. *Soft Computing* **9**, 9, (2005), 640–649.
3. Ciobanu, G, Păun, Gh. and Pérez-Jiménez, M.J. *Applications of Membrane Computing*, Natural Computing Series, Springer, 2006.
4. Christinal, H.A., Díaz-Pernil, D., Gutiérrez-Naranjo, M.A. and Pérez-Jiménez, M.J. Tissue-like P systems without environment. In M.A. Martínez-del-Amor, Gh. Păun, I. Pérez-Hurtado, A. Riscos-Núñez (eds.) *Proceedings of the Eight Brainstorming Week on Membrane Computing*, Sevilla, Spain, February 1-5, 2010, Fénix Editora, Report RGNC 01/2010, pp. 53–64.
5. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez , A. and Romero–Campero, F.J. Computational efficiency of cellular division in tissue-like P systems. *Romanian Journal of Information Science and Technology* **11**, 3, (2008), 229–241.
6. Freund, R., Păun, Gh. and Pérez-Jiménez, M.J. Tissue P Systems with channel states. *Theoretical Computer Science* **330**, (2005), 101–116.
7. Garey, M.R. and Johnson, D.S. *Computers and Intractability A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, (1979).
8. Gutiérrez-Escudero, R., Pérez-Jiménez, M.J. and Rius–Font, M. Characterizing tractability by tissue-like P systems. *Lecture Notes in Computer Science* **5957**, (2010), 289–300.

9. Ito, M., Martín Vide, C. and Păun, Gh. A characterization of Parikh sets of ET0L laguages in terms of P systems. In M. Ito, Gh. Păun, S. Yu (eds.) *Words, Semigroups and Transducers*, World Scientific, Singapore, 2001, 239-254.
10. Krishna, S.N., Lakshmanan K. and Rama, R. Tissue P Systems with Contextual and Rewriting Rules. *Lecture Notes in Computer Science* **2597**, (2003), 339–351.
11. Lakshmanan K. and Rama, R. On the Power of Tissue P Systems with Insertion and Deletion Rules. In A. Alhazov, C. Martín-Vide and Gh. Păun (eds.) *Preproceedings of the Workshop on Membrane Computing*, Tarragona, Report RGML 28/03, (2003), pp. 304–318.
12. Martín Vide, C. Pazos, J. Păun, Gh. and Rodríguez Patón, A. A New Class of Symbolic Abstract Neural Nets: Tissue P Systems. *Lecture Notes in Computer Science* **2387**, (2002), 290–299.
13. Martín Vide, C. Pazos, J. Păun, Gh. and Rodríguez Patón, A. Tissue P systems. *Theoretical Computer Science*, **296**, (2003), 295–326.
14. Pan, L. and Ishdorj, T.-O. P systems with active membranes and separation rules. *Journal of Universal Computer Science*, **10**, 5, (2004), 630–649.
15. Pan, L. and Pérez-Jiménez, M.J. Computational complexity of tissue–like P systems. *Journal of Complexity*, **26**, 3 (2010), 296–315.
16. Păun, Gh. Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1, (2000), 108–143. Also in Turku Center for Computer Science–TUCS, Report 208, November 1998.
17. Păun, Gh. Attacking **NP**-complete problems. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou, C. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, pp. 94-115.
18. Păun, Gh. *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, (2002).
19. Păun, A. and Păun, Gh. The power of communication: P systems with symport/antiport. *New Generation Computing*, **20**, 3, (2002), 295–305.
20. Păun, Gh., Pérez-Jiménez, M.J. and Riscos-Núñez, A. Tissue P System with cell division. *In. J. of Computers, Communications and Control*, **3**, 3, (2008), 295–303.
21. Gh. Păun, G. Rozenberg and A. Salomaa. *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2009.
22. Pérez-Jiménez, M.J., Romero-Jiménez, A. and Sancho-Caparrini, F. Complexity classes in models of cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265–285.
23. Pérez-Jiménez, M.J., Romero-Jiménez, A. and Sancho-Caparrini, F. A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics*, **11**, 4, (2006), 423-434.
24. Prakash, V.J. On the Power of Tissue P Systems Working in the Maximal-One Mode. In A. Alhazov, C. Martín-Vide and Gh. Păun (eds.). *Preproceedings of the Workshop on Membrane Computing*, Tarragona, Report RGML 28/03, (2003), pp. 356–364.
25. ISI web page `http://esi-topics.com/erf/october2003.html`
26. P systems web page `http://ppage.psystems.eu/`

An Optimal Frontier of the Efficiency of Tissue P Systems with Cell Division

Antonio E. Porreca[1], Niall Murphy[2,3], Mario J. Pérez-Jiménez[4]

[1] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano–Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
[2] Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid
Campus de Montegancedo s/n, Boadilla del Monte, 28660 Madrid, Spain
[3] CEI Campus Moncloa, UCM-UPM, Madrid, Spain
[4] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: porreca@disco.unimib.it, niall.murphy@upm.es, marper@us.es

Summary. In the framework of tissue P systems with cell division, the length of communication rules provides a frontier for the tractability of decision problems. On the one hand, the limitation on the efficiency of tissue P systems with cell division and communication rules of length 1 has been established. On the other hand, polynomial time solutions to **NP**–complete problems by using families of tissue P systems with cell division and communication rules of length at most 3 has been provided.

In this paper, we improve the previous result by showing that the HAM-CYCLE problem can be solved in polynomial time by a family of tissue P systems with cell division by using communication rules with length at most 2. Hence, a new tractability boundary is given: passing from 1 to 2 amounts to passing from non–efficiency to efficiency, assuming that $\mathbf{P} \neq \mathbf{NP}$.

1 Preliminaries

An *alphabet*, Σ, is a non–empty set whose elements are called *symbols*. An ordered finite sequence of symbols is a *string* or *word*. If u and v are strings over Σ, then so is their *concatenation* uv, obtained by juxtaposition, that is, writing u and v one after the other. The number of symbols in a string u is the *length* of the string and it is denoted by $|u|$. As usual, the empty string (with length 0) will be denoted by λ. The set of all strings over an alphabet Σ is denoted by Σ^*. In algebraic terms, Σ^* is the free monoid generated by Σ under the operation of concatenation. Subsets, finite or infinite, of Σ^* are referred to as *languages* over Σ.

The *Parikh vector* associated with a string $u \in \Sigma^*$ with respect to alphabet $\Sigma = \{a_1, \ldots, a_r\}$ is $\Psi_\Sigma(u) = (|u|_{a_1}, \ldots, |u|_{a_r})$, where $|u|_{a_i}$ denotes the number of ocurrences of symbol a_i in string u. This is called the *Parikh mapping* associated with Σ. Notice that, in this definition, the ordering of the symbols from Σ is relevant. If $\Sigma_1 = \{a_{i_1}, \ldots, a_{i_s}\} \subseteq \Sigma$, then we define $\Psi_{\Sigma_1}(u) = (|u|_{a_{i_1}}, \ldots, |u|_{a_{i_s}})$, for each $u \in \Sigma^*$.

A *multiset* m over a set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in A \mid f(x) > 0\}$. A multiset is empty (resp. finite) if its support is the empty set (resp. a finite set). If $m = (A, f)$ is a finite multiset over A and $supp(m) = \{a_1, \ldots, a_k\}$, then it will be denoted as $m = \{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}$. That is, superscripts indicate the multiplicity of each element, and if $f(x) = 0$ for $x \in A$, then element x is omitted. A finite multiset $m = \{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}$ can also be represented by the string $a_1^{f(a_1)} \ldots a_k^{f(a_k)}$ over the alphabet $\{a_1, \ldots, a_k\}$. Nevertheless, all permutations of this string identify the same multiset m precisely. Throughout this paper, we speak about "the finite multiset m" where m is a string, meaning "the finite multiset represented by the string m".

If $m_1 = (A, f_1)$, $m_2 = (A, f_2)$ are multisets over A, then we define the union of m_1 and m_2 as $m_1 + m_2 = (A, g)$, where $g = f_1 + f_2$, that is, $g(a) = f_1(a) + f_2(a)$, for each $a \in A$.

For any sets A and B the *relative complement* $A \setminus B$ of B in A is defined as follows:

$$A \setminus B = \{x \in A \mid \ x \notin B\}$$

In what follows, we assume the reader is already familiar with the basic notions and terminology of P systems. For details, see [9].

2 Introduction

Several different models of cell-like P systems have been successfully used to solve computationally hard problems efficiently by trading space for time. An exponential workspace is created in polynomial time by using some kind of rules, and then massive parallelism is used to simultaneously check all the candidate solutions. Inspired by living cells, several ways for obtaining exponential workspace in polynomial time were proposed: membrane division (*mitosis*) [8], membrane creation (*autopoiesis*) [4], and membrane separation (*membrane fission*) [6]. These three ways have given rise to the following models: *P systems with active membranes*, *P systems with membrane creation*, and *P systems with membrane separation*.

A new type of P systems, the so-called *tissue P systems*, was considered in [5]. Instead of considering a hierarchical arrangement, membranes/cells are placed in the nodes of a virtual graph. This variant has two biological justifications: intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a net of processors dealing with symbols

and communicating these symbols along channels specified in advance. Communication among cells is based on symport/antiport rules, which were introduced to P systems in [10]. One of the most interesting variants of tissue P systems was presented in [11], where the definition of tissue P systems is combined with aspects of the definition of P systems with active membranes, yielding *tissue P systems with cell division*. In these models [11], cells may replicate, that is, the two new cells generated by a division rule have exactly the same objects except for at most one differing pair of objects.

2.1 Tissue P Systems with communication rules

Definition 2.1 *A tissue P system with symport/antiport rules of degree* $q \geq 1$ *is a tuple* $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$*, where:*

1. Γ *is a finite* alphabet.
2. $\mathcal{E} \subseteq \Gamma$.
3. $\mathcal{M}_1, \ldots, \mathcal{M}_q$ *are strings over* Γ.
4. $\mathcal{R}$ *is a finite set of communication rules of the form* $(i, u/v, j)$*, for* $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$*,* $u, v \in \Gamma^*$*,* $|uv| > 0$.
5. $i_{out} \in \{0, 1, 2, \ldots, q\}$.

A *tissue P system with symport/antiport rules* $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$, of degree $q \geq 1$ can be viewed as a set of q cells, labelled by $1, \ldots, q$, with an environment labelled by 0 such that: (a) $\mathcal{M}_1, \ldots, \mathcal{M}_q$ are strings over Γ representing the finite multisets of objects (elements in Γ) initially placed in the q cells of the system; (b) $\mathcal{E}$ is the set of objects located initially in the environment of the system, all of them appearing in an *arbitrary number of copies*; and (c) $i_{out} \in \{0, 1, 2, \ldots, q\}$ represents a distinguished cell or the environment which will encode the output of the system.

When applying a rule $(i, u/v, j)$, the objects of the multiset represented by u are sent from region i to region j and, simultaneously, the objects of multiset v are sent from region j to region i. The length of the communication rule $(i, u/v, j)$ is defined as $|u| + |v|$, that is, the total number of objects which appear in the rule.

A communication rule $(i, u/v, j)$ is called a *symport rule* if $u = \lambda$ or $v = \lambda$. A symport rule $(i, u/\lambda, j)$, with $i \neq 0, j \neq 0$, provides a virtual arc from cell i to cell j. A communication rule $(i, u/v, j)$ is called an *antiport rule* if $u \neq \lambda$ and $v \neq \lambda$. An antiport rule $(i, u/v, j)$, with $i \neq 0, j \neq 0$, provides two arcs: one from cell i to cell j and another one from cell j to cell i. Thus, every tissue P systems has an underlying directed graph whose nodes are the cells of the system and the arcs are obtained from communication rules. In this context, the environment can be considered as a virtual node of the graph such that their connections are defined by communication rules of the form $(i, u/v, j)$, with $i = 0$ or $j = 0$.

The rules of a system like the one above are used in a non-deterministic maximally parallel manner as it is customary in Membrane Computing. At each step, all cells which can evolve must evolve in a maximally parallel way (at each step

we apply a multiset of rules which is maximal, no further applicable rule can be added).

An *instantaneous description* or a *configuration* at any instant of a tissue P system is described by all multisets of objects over Γ associated with all the cells present in the system, and the multiset of objects over $\Gamma - \mathcal{E}$ associated with the environment at that moment. Bearing in mind that the objects from $\mathcal{E}$ have infinite copies in the environment, they are not properly changed along the computation. The *initial configuration* is $(\mathcal{M}_1, \cdots, \mathcal{M}_q; \emptyset)$. A configuration is a *halting configuration* if no rule of the system is applicable to it.

Let us fix a tissue P system with symport/antiport rules Π. We say that configuration $\mathcal{C}_1$ yields configuration $\mathcal{C}_2$ in one *transition step*, denoted $\mathcal{C}_1 \Rightarrow_\Pi \mathcal{C}_2$, if we can pass from $\mathcal{C}_1$ to $\mathcal{C}_2$ by applying the rules from $\mathcal{R}$ following the previous remarks. A *computation* of Π is a (finite or infinite) sequence of configurations such that:

1. the first term of the sequence is an initial configuration of the system;
2. each non-initial configuration of the sequence is obtained from the previous configuration by applying the rules of the system in a maximally parallel manner with the restrictions previously mentioned; and
3. if the sequence is finite (called *halting computation*), then the last term of the sequence is a halting configuration.

All computations start from an initial configuration and proceed as stated above; only halting computations give a result, which is encoded by the objects present in the output region (a cell or the environment) i_{out} in the halting configuration.
Notation: If $\mathcal{C} = \{\mathcal{C}_i\}_{i<r+1}$ $(r \in \mathbf{N})$ is a halting computation of Π, then the length of $\mathcal{C}$ is r, that is, the number of non-initial configurations which appear in the finite sequence $\mathcal{C}$. We denote it by $|\mathcal{C}|$. We also denote by $\mathcal{C}_i(j)$ the contents of cell j at configuration $\mathcal{C}_i$.

2.2 Tissue P Systems with Cell Division

Cell division is an elegant process that enables organisms to grow and reproduce. Mitosis is a process of cell division which results in the production of two daughter cells from a single parent cell. Daughter cells are identical to one another and to the original parent cell. Through a sequence of steps, the replicated genetic material in a parent cell is equally distributed to two daughter cells. While there are some subtle differences, mitosis is remarkably similar across organisms.

Before a dividing cell enters mitosis, it undergoes a period of growth where the cell replicates its genetic material and organelles. Replication is one of the most important functions of a cell. DNA replication is a simple and precise process that creates two complete strands of DNA (one for each daughter cell) where only one existed before (from the parent cell).

Let us recall that the model of *tissue P systems with cell division* is based on the cell-like model of P systems with active membranes [8]. In these models, the

cells are not polarized; the cells obtained by division have the same labels as the original cell, and if a cell is divided, its interaction with other cells or with the environment is locked during the division process. In some sense, this means that while a cell is dividing it closes its communication channels.

Definition 2.2 *A tissue P system with cell division of degree* $q \geq 1$ *is a tuple* $\Pi = (\Gamma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$*, where:*

1. Γ *is a finite* alphabet.
2. $\mathcal{E} \subseteq \Gamma$.
3. $\mathcal{M}_1, \ldots, \mathcal{M}_q$ *are strings over* Γ.
4. $\mathcal{R}$ *is a finite set of rules of the following forms:*
   (a) Communication rules*:* $(i, u/v, j)$*, for* $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$*,* $u, v \in \Gamma^*$*,* $|u \cdot v| \neq 0$*;*
   (b) Division rules*:* $[a]_i \rightarrow [b]_i[c]_i$*, where* $i \in \{1, 2, \ldots, q\}$*,* $i \neq i_{out}$ *and* $a, b, c \in \Gamma$.
5. $i_{out} \in \{0, 1, 2, \ldots, q\}$.

A *tissue P system with cell division* is a *tissue P system with symport/antiport rules* where division rules of cells are allowed.

When applying a division rule $[a]_i \rightarrow [b]_i[c]_i$, under the influence of object a, the cell with label i is divided into two cells with the same label; in the first copy, object a is replaced by object b, in the second one, object a is replaced by object c; all the other objects are replicated and copies of them are placed in the two new cells. The output cell i_{out} cannot be divided.

The rules of a tissue P systems with cell division are applied in a nondeterministic maximally parallel manner as it is customary in membrane computing. At each step, all cells which can evolve must evolve in a maximally parallel way (at each step we apply a multiset of rules which is maximal, no further rule can be added), with the following important remark: if a cell divides, only the division rule is applied to that cell at that step; the objects inside that cell do not evolve by means of communication rules. In other words, we can think that before division a cell interrupts all its communication channels with the other cells and with the environment. The new cells resulting from division will only interact with other cells or with the environment at the next step – providing they do not divide once again. The label of a cell identifies the rules which can be applied to it precisely.

2.3 Recognizer Tissue P Systems with Cell Division

Let us recall that a *decision problem* is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X. There are many different ways to describe instances of a decision problem, but we assume that each problem has associated with it a fixed *reasonable encoding scheme* (in the sense of [2], page 10) which provides a string associated

with each problem instance. The *size* of an instance $u \in I_X$ is the length of the string associated with it by means of a reasonable encoding scheme.

Many abstract problems are not decision problems, for example, in *combinatorial optimization problems* some value must be optimized (minimized or maximized). In order to deal with such problems, they can be transformed into roughly equivalent decision problems by supplying a target/threshold value for the quantity to be optimized, and then asking whether this value can be attained.

A natural correspondence between decision problems and languages over a finite alphabet, can be established as follows. Given a decision problem $X = (I_X, \theta_X)$, its associated language is $L_X = \{w \in I_X : \theta_X(w) = 1\}$. Conversely, given a language L over an alphabet Σ, its associated decision problem is $X_L = (I_{X_L}, \theta_{X_L})$, where $I_{X_L} = \Sigma^*$, and $\theta_{X_L} = \{(x,1) : x \in L\} \cup \{(x,0) : x \notin L\}$. The solvability of decision problems is defined through the recognition of the languages associated with them by means of languages recognizer devices.

In order to study the computational efficiency of membrane systems, the notions from classical *computational complexity theory* are adapted for Membrane Computing, and a special class of cell-like P systems is introduced in [13]: *recognizer P systems* (called *accepting P systems* in a previous paper [12]). Similarly, *recognizer tissue P systems* are introduced in [11].

Definition 2.3 *A* recognizer tissue P system with cell division *of degree* $q \geq 1$ *is a tuple* $\Pi = (\Gamma, \Sigma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$*, where:*

1. $(\Gamma, \mathcal{E}, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{out})$ *is a tissue P system with cell division of degree* $q \geq 1$ *(as defined in the previous section).*
2. *The working alphabet* Γ *has two distinguished objects* `yes` *and* `no` *being, at least, one copy of them present in some initial multisets* $\mathcal{M}_1, \ldots, \mathcal{M}_q$*, but none of them are present in* $\mathcal{E}$*.*
3. Σ *is an (input) alphabet strictly contained in* Γ *such that* $\mathcal{E} \cap \Sigma = \emptyset$*.*
4. $\mathcal{M}_1, \ldots, \mathcal{M}_q$ *are strings over* $\Gamma \setminus \Sigma$*.*
5. $i_{in} \in \{1, \ldots, q\}$ *is the input cell.*
6. $i_{out} = 0$*, that is, the output region is the environment.*
7. *All computations halt.*
8. *If* $\mathcal{C}$ *is a computation of* Π*, then either object* `yes` *or object* `no` *(but not both) must have been released into the environment, and only at the last step of the computation.*

For each multiset m over Σ, the *computation of the system* Π *with input* m starts from the configuration of the form $(\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_{i_{in}} + m, \ldots, \mathcal{M}_q; \emptyset)$, that is, the input multiset m has been added to the contents of the input cell i_{in}. Therefore, we have an initial configuration associated with each input multiset m (over the input alphabet Σ) in this kind of systems.

Given a recognizer tissue P system with cell division Π, and a halting computation $\mathcal{C} = \{\mathcal{C}_i\}_{i<r+1}$ of Π ($r \in \mathbf{N}$), we define the result of $\mathcal{C}$ as follows:

$$Output(\mathcal{C}) = \begin{cases} \texttt{yes}, & \text{if } \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{r,0}) = (1,0) \wedge \\ & \quad \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{i,0}) = (0,0) \text{ for } i = 0,\ldots,r-1 \\ \texttt{no}, & \text{if } \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{r,0}) = (0,1) \wedge \\ & \quad \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{i,0}) = (0,0) \text{ for } i = 0,\ldots,r-1 \end{cases}$$

where Ψ is the Parikh function, and $M_{i,0}$ is the multiset over $\Gamma \setminus \mathcal{E}$ associated with the environment at configuration $\mathcal{C}_i$. In particular, $M_{r,0}$ is the multiset over $\Gamma \setminus \mathcal{E}$ associated with the environment at the halting configuration $\mathcal{C}_r$.

We say that a computation $\mathcal{C}$ is an *accepting computation* (respectively, *rejecting computation*) if $Output(\mathcal{C}) = \texttt{yes}$ (respectively, $Output(\mathcal{C}) = \texttt{no}$), that is, if object `yes` (respectively, object `no`) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$, and neither object `yes` nor `no` appears in the environment associated with any non–halting configuration of $\mathcal{C}$.

For each natural number $k \geq 1$, we denote by $\mathbf{TDC}(k)$ the class of recognizer tissue P systems with cell division and communication rules with length at most k.

2.4 Polynomial Complexity Classes of Tissue P systems with Cell Division

Now, we define what it means to solve a decision problem in the framework of tissue P systems efficiently and in a uniform way. Since we define each tissue P system to work on a finite number of inputs, to solve a decision problem we define a numerable family of tissue P systems.

Definition 2.4 *We say that a decision problem $X = (I_X, \theta_X)$ is solvable in a uniform way and polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of recognizer tissue P systems with cell division if the following holds:*

1. *The family $\mathbf{\Pi}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$.*
2. *There exists a pair (cod, s) of polynomial-time computable functions over I_X such that:*
   (a) *for each instance $u \in I_X$, $s(u)$ is a natural number[5] and $cod(u)$ is an input multiset of the system $\Pi(s(u))$;*
   (b) *for each $n \in \mathbb{N}$, $s^{-1}(n)$ is a finite set;*
   (c) *the family $\mathbf{\Pi}$ is* polynomially bounded *with regard to (X, cod, s), that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(s(u))$ with input $cod(u)$ is halting and it performs at most $p(|u|)$ steps;*
   (d) *the family $\mathbf{\Pi}$ is* sound *with regard to (X, cod, s), that is, for each $u \in I_X$, if there exists an accepting computation of $\Pi(s(u))$ with input $cod(u)$, then $\theta_X(u) = 1$;*

[5] Note, for this definition to be compatible with the notion of uniformity in Boolean circuit complexity [15] we restrict $s(u)$ to be some function on $|u|$, the length of u.

(e) the family $\mathbf{\Pi}$ *is* complete *with regard to* (X, cod, s)*, that is, for each* $u \in I_X$*, if* $\theta_X(u) = 1$*, then every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting one.*

From the soundness and completeness conditions above we deduce that every P system $\Pi(n)$ is *confluent*, in the following sense: every computation of a system with the *same* input multiset must always give the *same* answer.

Let $\mathbf{R}$ be a class of recognizer tissue P systems. We denote by $\mathbf{PMC_R}$ the set of all decision problems which can be solved in a uniform way and polynomial time by means of families of systems from $\mathbf{R}$. The class $\mathbf{PMC_R}$ is closed under complement and polynomial–time reductions [12].

3 Computational Efficiency of Tissue P Systems with Cell Division

It is well known that tissue P systems with cell division are able to solve computationally hard problems efficiently. Specifically, **NP**–complete problems have been solved in linear time [1] by using families of tissue P systems with cell division and communication rules of length at most 3. Thus, $\mathbf{NP} \cup \mathbf{co-NP} \subseteq \mathbf{PMC}_{TDC(3)}$. In [3] has been proved $\mathbf{P} = \mathbf{PMC}_{TDC(1)}$, that is, only tractable problems can be efficiently solved by using families of tissue P systems with cell division and communication rules of length 1. Therefore, in the framework of tissue P systems with cell division, passing the maximum length of communication rules of the systems from 1 to 3 amounts to passing from non–efficiency to efficiency, assuming that $\mathbf{P} \neq \mathbf{NP}$. An interesting challenge is to provide new efficient solutions to computationally hard problems by means of tissue P systems with cell division by using communication rules of length at most 2.

In the next Section, we give a family of tissue P systems with cell division and communication rules of length at most 2 which solves the `HAM-CYCLE` problem, a well known **NP**–complete problem, in polynomial time.

4 On efficiency of TDC(2)

We start by giving some concepts and notations related to graph theory that we will use throughout this paper.

4.1 Hamiltonian cycles in directed graphs

First of all, let us recall some concepts related to graph theory which are relevant in this paper.

Definition 4.1 *Let* $G = (V, E)$ *be a directed graph. Let* $V = \{1, \ldots, n\}$*,* $E = \{(u_1, v_1), \ldots, (u_p, v_p)\} \subset V \times V$*. A finite sequence* $\gamma = (u_{\alpha_1}, u_{\alpha_2}, \ldots, u_{\alpha_r}, u_{\alpha_{r+1}})$ *of nodes of* G *is a simple path of* G *of length* $r \geq 1$ *if the following holds:*

- $\forall i\ (1 \leq i \leq r \rightarrow (u_{\alpha_i} u_{\alpha_{i+1}}) \in E)$.
- $|\{u_{\alpha_1}, u_{\alpha_2}, \ldots, u_{\alpha_r}\}| = r$.

If $u_{\alpha_{r+1}} \notin \{u_{\alpha_1}, u_{\alpha_2}, \ldots, u_{\alpha_r}\}$, then we say that γ is a simple path of length r from u_{α_1} to $u_{\alpha_{r+1}}$. If $u_{\alpha_{r+1}} = u_{\alpha_1}$, then we say that γ is a simple cycle of length r (in this case, we assume $r \geq 2$). A Hamiltonian path of G from $a \in V$ to $b \in V$ $(a \neq b)$ is a simple path $\gamma = (u_{\alpha_1}, u_{\alpha_2}, \ldots, u_{\alpha_r}, u_{\alpha_{r+1}})$ from a to b such that $a = u_{\alpha_1}$, $b = u_{\alpha_{r+1}}$, and $V = \{u_{\alpha_1}, u_{\alpha_2}, \ldots, u_{\alpha_r}, u_{\alpha_{r+1}}\}$. A Hamiltonian cycle of G is a simple cycle $\gamma = (u_{\alpha_1}, u_{\alpha_2}, \ldots, u_{\alpha_r}, u_{\alpha_{r+1}})$ of G such that $V = \{u_{\alpha_1}, u_{\alpha_2}, \ldots, u_{\alpha_r}\}$.

If $\gamma = (u_{\alpha_1}, u_{\alpha_2}, \ldots, u_{\alpha_r}, u_{\alpha_{r+1}})$ is a simple path of G then we also denote it by the set $\{(u_{\alpha_1}, u_{\alpha_2})_1, (u_{\alpha_2}, u_{\alpha_3})_2, \ldots, (u_{\alpha_r}, u_{\alpha_{r+1}})_r\}$. That is, $(u_{\alpha_k}, u_{\alpha_{k+1}})_k$ can be interpreted as the k-th arc of the path γ, for each k $(1 \leq k \leq r)$.

Given a directed graph $G = (V, E)$, throughout this paper we denote

$$A_G = \{(u,v)_k \mid u, v, k \in \{1, \ldots, n\} \ \wedge \ (u,v) \in E\}$$
$$A'_G = \{(u,v)'_k \mid u, v, k \in \{1, \ldots, n\} \ \wedge \ (u,v) \in E\}$$
$$A''_G = \{(u,v)''_k \mid u, v, k \in \{1, \ldots, n\} \ \wedge \ (u,v) \in E\}$$

Proposition 4.2 *Let $G = (V, E)$ be a directed graph. Let $V = \{1, \ldots, n\}$ and $A_G = \{(u,v)_k |\ u, v, k \in \{1, \ldots, n\} \ \wedge \ (u,v) \in E\}$. If $B \subseteq A_G$ then the following assertions are equivalent:*

1. *B is a Hamiltonian cycle.*
2. *$|B| = n$ and the following holds: for each $\forall u, u', v, v', k, k' \in \{1, \ldots, n\}$,*
   (a) $[(u,v)_k \in B \wedge (u',v')_{k'} \in B \wedge (u,v)_k \neq (u',v')_{k'} \rightarrow k \neq k']$
   (b) $[(u,v)_k \in B \wedge (u',v')_{k'} \in B \wedge (u,v)_k \neq (u',v')_{k'} \rightarrow u \neq u']$
   (c) $[(u,v)_k \in B \wedge (u',v')_{k'} \in B \wedge (u,v)_k \neq (u',v')_{k'} \rightarrow v \neq v']$
   (d) $[(u,v)_k \in B \wedge (u',v')_{k+1} \in B \rightarrow v = u']$

Proof: Let $B = \{(u_{\alpha_1}, u_{\alpha_2})_1, (u_{\alpha_2}, u_{\alpha_3})_2 \ldots, (u_{\alpha_m}, u_{\alpha_{r+1}})_n\}$ be a Hamiltonian cycle of G. Then, $|B| = n$ and the conditions (a), (b), (c) and (d) from (2) hold.

Let $B \subseteq A_G$ such that $|B| = n$ and the conditions (a), (b), (c) and (d) from (2) hold. Then, from (a) the set B must to be of the form

$$B = \{(u_{\alpha_1}, v_{\alpha_1})_1, (u_{\alpha_2}, v_{\alpha_2})_2 \ldots, (u_{\alpha_n}, v_{\alpha_n})_n\}$$

where:

- From (d) we deduce that $\forall i\ (1 \leq i \leq n-1 \rightarrow v_{\alpha_i} = u_{\alpha_{i+1}})$.
- From (b) we have $V = \{u_{\alpha_1}, u_{\alpha_2}, \ldots, u_{\alpha_n}\}$.

Finally, on the one hand we have $v_{\alpha_n} \in \{u_{\alpha_1}, u_{\alpha_2} \ldots, u_{\alpha_n}\}$. On the other hand, by condition (c) we deduce that $v_{\alpha_n} \notin \{v_{\alpha_1}, \ldots, v_{\alpha_{n-1}}\} = \{u_{\alpha_2}, \ldots, u_{\alpha_n}\}$. Thus $v_{\alpha_n} = u_{\alpha_1}$.

□

Remark 1: If $B \subseteq A_G$ is a Hamiltonian cycle of G, then it cannot have different pairs of elements of the types $(i,j)_k$ and $(i,j')_{k'}$, or of the types $(i,j)_k$ and $(i',j)_{k'}$, or $(i,j)_k$ and $(i',j')_k$, or $(i,j)_k$ and $(i',j')_{k+1}$ with $j \neq i'$.

Remark 2: Let us notice that if $(u_{\alpha_1}, u_{\alpha_2}, \ldots, u_{\alpha_n}, u_{\alpha_1})$ is a Hamiltonian cycle of G of length n, then we can describe it by the following subset of A_G:

$$B_1 = \{(u_{\alpha_1}, u_{\alpha_2})_1, (u_{\alpha_2}, u_{\alpha_3})_2, \ldots, (u_{\alpha_n}, u_{\alpha_1})_n\}$$

But $(u_{\alpha_2}, u_{\alpha_3}, \ldots, u_{\alpha_m}, u_{\alpha_1}, u_{\alpha_2})$ is also a Hamiltonian cycle of G of length m. It can be described as follows:

$$B_2 = \{(u_{\alpha_2}, u_{\alpha_3})_1, (u_{\alpha_3}, u_{\alpha_4})_2, \ldots, (u_{\alpha_1}, u_{\alpha_2})_n\}$$

Thus, given a Hamiltonian cycle γ of G, there are exactly n different subsets of A_G codifying exactly the cycle γ.
Remark 3: Let us supose that the total number of Hamiltonian cycles of G is q. Then, the number of different subsets B of A_G verifying conditions (a), (b), (c), and (d) of the previous Proposition is exactly $n \cdot q$.

4.2 An efficient, uniform solution of HAM-CYCLE in TDC(2)

In this Section we provide a uniform and polynomial time solution for the `HAM-CYCLE` problem by using a family of tissue P systems with cell division and communication rules of length at most 2.

Let us recall that the `HAM-CYCLE` problem is the following: *given a directed graph, to determine whether or not there exists a Hamiltonian cycle in the graph.* This is a well known **NP**-complete problem [2].

The proposed solution follows a brute force algorithm implemented in the framework of recognizer tissue P systems with cell division. The solution consists of the following stages:

- *Generation Stage*: From the input cell labelled by *in*, all possible combinations of arcs including a code of their position in potential paths, are generated in those cells and by using cell division in an adequate way.
- *Checking Stage*: In each cell labelled by *in*, it is checked whether or not the different combinations of arcs encode Hamiltonian cycles of the graph.
- *Output Stage*: The system sends the right answer to the environment according to the results of the previous stage.

Then, we define a family $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ of recognizer tissue P system with cell division from $\mathbf{TDC}(2)$, such that each system $\Pi(n)$ will process all instances G of `HAM-CYCLE` with n nodes.

For each $n \in \mathbb{N}$, we consider the recognizer tissue P system with cell division from $\mathbf{TDC}(2)$,

$$\Pi(n) = (\Gamma, \Sigma, \mathcal{E}, \mathcal{M}_{in}, \mathcal{M}_h, \mathcal{M}_y, \mathcal{M}_{yes}, \mathcal{M}_{no}, \mathcal{M}_{out}, \mathcal{M}_{e_{i,j,k}}(1 \leq i, j, k \leq n), \mathcal{M}_{c_i}(1 \leq i, \leq n), \mathcal{R}, i_{in}, i_{out})$$

defined as follows:

- The input alphabet is $\Sigma = \{(i,j)_k \mid 1 \leq i,j,k \leq n\}$.
- The working alphabet is

$$\begin{aligned}\Gamma = \{&(i,j)_k, (i,j)'_k, (i,j)''_k, (i,j)_{k,r}, (i,j)'_{k,r}, (i,j)''_{k,r} \mid 1 \leq i,j,k \leq n \wedge \\ &1 \leq r \leq n^3\} \cup \\ &\{w_i \mid 1 \leq i \leq n^3+6\} \cup \{c_r, h_r, y_r \mid 1 \leq r \leq n^3\} \cup \\ &\{w, c, c', c'', h, h', h'', h''', y, y'y'', y''', y'''', x, yes, no, \#\}\end{aligned}$$

- The alphabet of the environment is:

$$\mathcal{E} = \{w_i \mid 1 \leq i \leq n^3+5\} \cup \{w, c'', y'', h'', y''', h''', y''''\}$$

- Initial multisets:

$$\begin{cases} \mathcal{M}_{in} = c^n\ y\ h \\ \mathcal{M}_{e_{i,j,k}} = (i,j)''_{k,n^3},\ 1 \leq i,j,k \leq n \\ \mathcal{M}_{c_i} = c_{n^3},\ 1 \leq i \leq n \\ \mathcal{M}_h = h_{n^3} \\ \mathcal{M}_y = y_{n^3} \\ \mathcal{M}_{yes} = yes \\ \mathcal{M}_{no} = w_{n^3+6}\ no \\ \mathcal{M}_{out} = x \end{cases}$$

- The set R of rules consists of the following rules:

(1) $(no\,,\, w_r \,/\, w_{r-1}\,,\, 0)$, for $2 \leq r \leq n^3+6$.
(2) $(no\,,\, w_1 \,/\, w\,,\, 0)$.
(3) $[\,(i,j)_k\,]_{in} \to [\,(i,j)'_k\,]_{in}\ [\,\#\,]_{in}$, for $1 \leq i,j,k \leq n$.
(4) $[\,(i,j)''_{k,r}\,]_{e_{i,j,k}} \to [\,(i,j)''_{k,r-1}\,]_{e_{i,j,k}}\ [\,(i,j)''_{k,r-1}\,]_{e_{i,j,k}}$, for $1 \leq i,j,k \leq n$ and $2 \leq r \leq n^3$.
(5) $[\,(i,j)''_{k,1}\,]_{e_{i,j,k}} \to [\,(i,j)''_k\,]_{e_{i,j,k}}\ [\,(i,j)''_k\,]_{e_{i,j,k}}$, for $1 \leq i,j,k \leq n$.
(6) $[\,c_r\,]_{c_i} \to [\,c_{r-1}\,]_{c_i}\ [\,c_{r-1}\,]_{c_i}$, for $1 \leq i \leq n\ \wedge\ 1 \leq r \leq n^3$.
(7) $[\,y_r\,]_y \to [\,y_{r-1}\,]_y\ [\,y_{r-1}\,]_y$, for $1 \leq r \leq n^3$.
(8) $[\,h_r\,]_h \to [\,h_{r-1}\,]_h\ [\,a_{r-1}\,]_h$, for $1 \leq r \leq n^3$.
(9) $(in\,,\, (i,j)'_k \,/\, (i,j)''_k\,,\, e_{i,j,k})$, for $1 \leq i,j,k \leq n$.
(10) $(in\,,\, c \,/\, c'\,,\, c_i)$, for $1 \leq i \leq n$.
(11) $(in\,,\, y \,/\, y',\, y)$.
(12) $(in\,,\, h \,/\, h',\, h)$.
(13) $(in,\, (i,j)''_k\,(i,j')''_{k'} \,/\, \lambda,\, 0)$, for $1 \leq i,j,j',k,k' \leq n$.
(14) $(in,\, (i,j)''_k\,(i',j)''_{k'} \,/\, \lambda,\, 0)$, for $1 \leq i,i',j,k,k' \leq n$.
(15) $(in,\, (i,j)''_k\,(i',j')''_{k+1} \,/\, \lambda,\, 0)$, for $1 \leq i,i',j,j',k \leq n$, and $j \neq i'$.
(16) $(in,\, (i,j)''_k\,(i',j')''_k \,/\, \lambda,\, 0)$, for $1 \leq i,i',j,j',k \leq n$.
(17) $(in\,,\, c' \,/\, c''\,,\, 0)$.
(18) $(in\,,\, y' \,/\, y''\,,\, 0)$.
(19) $(in\,,\, h' \,/\, h''\,,\, 0)$.
(20) $(in\,,\, (i,j)''_k\,c'' \,/\, \lambda\,,\, 0)$ for $1 \leq i,j,k \leq n$.
(21) $(in\,,\, y'' \,/\, y'''\,,\, 0)$.

(22) $(in\,,\, h''\,/\,h'''\,,\, 0)$.
(23) $(in\,,\, c''\,h'''\,/\,\lambda\,,\, 0)$.
(24) $(in\,,\, y'''\,/\,y''''\,,\, 0)$.
(25) $(in\,,\, h'''\,y''''\,/\,\lambda\,, yes)$.
(26) $(yes\,,\, y''''\,yes\,/\,\lambda\,,\, out)$.
(27) $(out\,,\, x\,yes\,/\,\lambda\,,\, 0)$.
(28) $(no\,,\, w\,no\,/\,\lambda\,,\, out)$.
(29) $(out\,,\, x\,no\,/\,\lambda\,,\, 0)$.

- The input cell is $i_{in} = in$.
- The output region is the environment, $i_{out} = 0$.

4.3 An Overview of the Computations

A family of recognizer tissue P systems with cell division is constructed above. Let $G = (V, E)$, with $V = \{1, \ldots, n\}$ and $E = \{(u_1, v_1), \ldots, (u_p, v_p)\}$, be an arbitrary instance of the `HAM-CYCLE` problem.

The *size* mapping[6]on the set of instances is defined as $s(G) = n$, and the encoding of the instance is the multiset

$$cod(G) = \{(u_i, v_i)_k \mid 1 \leq i \leq p \ \wedge \ 1 \leq k \leq n \ \wedge \ (u_i, v_i) \in E\}$$

That is, $(u_i, v_i)_k$ denotes arc (u_i, v_i) "placed" in "position k". Then the graph G will be processed by system $\Pi(s(G))$ with input multiset $cod(G)$.

Then, we informally describe how system $\Pi(s(G))$ with input multiset $cod(G)$ works, in order to process the instance G of the `HAM-CYCLE` problem.

At the initial configuration of $\Pi(s(G)) + cod(G)$ we have the following:

- n copies of object c, objects y, h, and $(u_i, v_i)_j$, for $(u_i, v_i) \in E$, $1 \leq k \leq n$, in cell labelled by in,
- Objects $(i, j)''_{k,n^3}$ in cell labelled by $e_{i,j,k}$.
- Objects c_{n^3} in each cell labelled by c_i, for $1 \leq i \leq n$.
- Object h_{n^3} in cell labelled by h, object y_{n^3} in cell labelled by y, object yes in cell labelled by h, objects no and w_{n^3+6} in cell labelled by no, and object x in cell labelled by out.

Let us start with the **generation stage**. This stage spends n^3 steps. At this stage, we try to generate all the possible subsets of arcs of the graph which contain their potential positions in a path according the notations introduced in Section 4.1 (in fact, subsets of A'_G).

If $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \ldots)$ be a computation of the tissue P system $\Pi(n)$, then at configuration $\mathcal{C}_{n^3}$:

[6] Note, for this family to be considered uniform in the sense of Boolean circuit families [15] we may modify s so that its mapping to the number n depends only on the length of of G. For example, if the graph G is encoded as a binary adjacency matrix, then $s(u) = \sqrt{|u|} = n$.

1. There are $2^{n \cdot p}$ cells labelled by in such that each of them contains a different subset of $A'_G = \{(u_i, v_i)'_k \mid 1 \leq i \leq p \wedge 1 \leq k \leq n \ \wedge \ (u_i, v_i) \in E\}$ as well as object y, object h and n copies of object c.
2. For each i, j, k $(1 \leq i, j, k \leq n)$ there are 2^{n^3} cells labelled by $e_{i,j,k}$, each of them only containing object $(i, j)''_k$.
3. For each i $(1 \leq i \leq n)$ there are 2^{n^3} cells labelled by c_i, 2^{n^3} cells labelled by h, and 2^{n^3} cells labelled by y, only containing object c', object h', object y' respectively.
4. There is a cell labelled by no, a cell labelled by yes and a cell labelled by out such that $\mathcal{C}_{n^3}(no) = \{w_6, no\}$, $\mathcal{C}_{n^3}(yes) = \{yes\}$, $\mathcal{C}_{n^3}(out) = \{x\}$.

Now, the **checking stage** starts. This stage spends 3 steps. At this stage, we try to determine whether or not there exists a cell labelled by in which contains a subset of A''_G that encodes a Hamiltonian cycle of G. For that purpose, we will use rules of types (13), (14), (15), and (16) in order to select possible paths of the graph. After that, rules of type (20), (21), and (22) allow us to determine cells labelled by in at configuration $\mathcal{C}_{n^3+3}$ which encode Hamiltonian cycles.

Finally, the **output stage** spends 3 steps if the answer is affirmative and 4 if it is negative. At configuration $\mathcal{C}_{n^3+3}$, the existence of Hamiltonian cycles in the graph is characterized by the absence of objects c'' in some cell labelled by in. At configuration $\mathcal{C}_{n^3+4}$, the previous condition is expressed by the existence of some cell labelled by in which contains object h'''. At configuration $\mathcal{C}_{n^3+5}$, the existence of Hamiltonian cycles in the graph is characterized by the presence of some object y'''' in cell labelled by yes. Rules of type (26), (27), (28), and (29) produce the right answer.

5 A Formal Verification

The aim of this section is to present a formal proof on the fact that the family of recognizer tissue P systems with cell division constructed in the previous section solves the `HAM-CYCLE` problem in a uniform way and polynomial time, according to Definition 2.4.

5.1 Polynomial Uniformity of the Family

Then, we will show that the family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ defined above is polynomially uniform by Turing machines. To this aim we prove that $\Pi(n)$ is built in polynomial time with respect to the number of nodes of the instance G of the `HAM-CYCLE` problem.

It is easy to check that the rules of a system $\Pi(n)$ of the family are recursively defined from n. The amount of resources needed to build an element of the family is of a polynomial order in n, as shown below:

1. Size of the alphabet: $3n^4 + 7n^3 + 23 \in \Theta(n^4)$.

2. Initial number of cells: $n^3 + n + 6 \in \Theta(n^3)$.
3. Initial number of objects: $|E| + n^3 + 2n + 8 \in \Theta(n^3)$.
4. Number of rules: $n^6 + 4n^5 - n^4 + 7n^3 + n + 20n \in \Theta(n^6)$.
5. Maximal length of a rule: $2 \in \Theta(1)$.

Therefore, there exists a deterministic Turing machine that builds the system $\Pi(n)$ in time polynomial with respect to n.

5.2 Soundness and Completeness of the Family

In order to show the soundness and completeness of the family $\mathbf{\Pi}$ with respect to $(\texttt{HAM-CYCLE}, cod, s)$, we describe the full contents of any cells in any instant of each computation of the tissue $\Pi(s(G)) + cod(G)$ that processes instance G.

Theorem 5.1 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. For every t $(1 \leq t \leq n \cdot p)$, the configuration $\mathcal{C}_t$ verifies the following properties:*

(1) *There are 2^t cells labelled by in such that each of them contains a different subset of $A'_G = \{(u_i, v_i)'_k \mid 1 \leq i \leq p \wedge 1 \leq k \leq n \ \wedge \ (u_i, v_i) \in E\}$ of size lower than or equal than t, as well as object y, object h and n copies of object c.*

(2) *For each i, j, k $(1 \leq i, j, k \leq n)$ there are 2^t cells labelled by $e_{i,j,k}$ each of them only containing object $(i, j)''_{k,n^3-t}$.*

(3) *For each i $(1 \leq i \leq n)$ there are 2^t cells labelled by c_i, 2^t cells labelled by h, and 2^t cells labelled by y, only containing object c_{n^3-t}, object h_{n^3-t}, object y_{n^3-t} respectively.*

(4) *There is a cell labelled by no, a cell labelled by yes and a cell labelled by out such that $\mathcal{C}_t(no) = \{w_{n^3-t+6}, no\}$, $\mathcal{C}_t(yes) = \{yes\}$, $\mathcal{C}_t(out) = \{x\}$.*

Proof: By induction on t. Let us start analyzing the basic case $t = 1$.

(1) At the first step of computation $\mathcal{C}$, a rule of the form $[\,(u_0, v_0)_{k_0}\,]_{in} \to [\,(u_0, v_0)'_{k_0}\,]_{in}\ [\,\#\,]_{in}$, with $(u_0, v_0) \in E$, will be applied to cell labelled by in. Then, two new cells labelled by in will be created, each of them containing a subset of A'_G of size lower than or equal to 1: one is $(u_0, v_0)'_{k_0}$ and the another is $\emptyset$. The initial objects y, h and n copies of object c remain unchanged.

(2) For each i, j, k $(1 \leq i, j, k \leq n)$, at the first step of computation $\mathcal{C}$, the rule $[\,(i, j)''_{k,n^3}\,]_e \to [\,(i, j)''_{k,n^3-1}\,]_e\ [\,(i, j)''_{k,n^3-1}\,]_e$ will be applied to cell labelled by $e_{i,j,k}$. Then, two new cells labelled by $e_{i,j,k}$ will be created, each of them only containing object $(i, j)''_{k,n^3-1}$.

(3) For each i $(1 \leq i \leq n)$, at the first step of computation $\mathcal{C}$, a rule of the type (6) is applied to cell labelled by c_i. It produces two new cells labelled by c_i, each of them only containing object c_{n^3-1}.
At the first step of computation $\mathcal{C}$, a rule of the type (7) is applied to the cell labelled by y, and a rule of the type (8) is applied to cell labelled by h. They produce two new cells labelled by y, each of them only containing object y_{n^3-1}, and two new cells labelled by h each of them only containing the object h_{n^3-1}.

(4) At the first step of computation $\mathcal{C}$, a rule of the type (1) is applied to the cell labelled by *no*. The new content of that cell labelled by *no* is $\{w_{n^3-1}\, no\}$. No rule is applied neither to cell *yes* nor cell *out*.

By induction hypothesis, let t be such that $1 \leq t < n \cdot p$ and let us suppose the result holds for t. Let us see that the result also holds for $t+1$. For this purpose, let us notice that configuration $\mathcal{C}_{t+1}$ is obtained from configuration $\mathcal{C}_t$ by applying:

- A rule of the type (3) $[\,(u_0, v_0)_k\,]_{in} \rightarrow [\,(u_0, v_0)'_k\,]_{in}\ [\,\#\,]_{in}$ which is selected in a nondeterministic manner among all possible applicable rules to each cell labelled by *in* (there exist such rules because of $t < n \cdot p$).
- For each i, j, k $(1 \leq i, j, k \leq n)$, rules of the type (4) corresponding to $r = n^3 - t$ in each cell labelled by $e_{i,j,k}$:

$$[\,(i,j)''_{k,n^3-t}\,]_{e_{i,j,k}} \rightarrow [\,(i,j)''_{k,n^3-t-1}\,]_{e_{i,j,k}}\ [\,(i,j)''_{k,n^3-t-1}\,]_{e_{i,j,k}}$$

- For each i $(1 \leq i \leq n)$, rules of the type (6) corresponding to $r = n^3 - t$ in each cell labelled by c_i: $[\,c_{n^3-t}\,]_{c_i} \rightarrow [\,c_{n^3-t-1}\,]_{c_i}\ [\,c_{n^3-t-1}\,]_{c_i}$.
- Rules of the type (7) corresponding to $r = n^3 - t$ in each cell labelled by y:

$$[\,y_{n^3-t}\,]_y \rightarrow [\,y_{n^3-t-1}\,]_y\ [\,y_{n^3-t-1}\,]_y$$

- Rules of the type (8) corresponding to $r = n^3 - t$ in each cell labelled by h:

$$[\,h_{n^3-t}\,]_h \rightarrow [\,h_{n^3-t-1}\,]_h\ [\,h_{n^3-t-1}\,]_h$$

- A rule of the type (1) corresponding to $r = n^3 - t$ in cell labelled by *no*:

$$(no\,,\, w_{n^3-t} \,/\, w_{n^3-t-1}\,,\, 0)$$

Therefore, the following conclusions are reached at:

(1) By induction hypothesis, in $\mathcal{C}_t$ there are 2^t cells labelled by *in*, each of them containing object y, object h, object n copies of object c, and a different subset of A'_G with size $\leq t$. Thus, when applying a rule of the type $[\,(u_0, v_0)_k\,]_{in} \rightarrow [\,(u_0, v_0)'_k\,]_{in}\ [\,\#\,]_{in}$, with $(u_0, v_0) \in E$, we will have 2^{t+1} cells labelled by *in* such that 2^t of that cells have the same content that they had at configuration $\mathcal{C}_t$, and the rest of 2^t objects $(u_0, v_0)'_k$ are added. That is, at configuration $\mathcal{C}_{t+1}$, we will have 2^{t+1} cells labelled by *in*, each of them containing object y, object h, n copies of object c, and a different subset of A'_G with size $\leq t+1$.

(2) By induction hypothesis, for each i, j, k $(1 \leq i, j, k \leq n)$ in $\mathcal{C}_t$ there are 2^t cells labelled by $e_{i,j,k}$, each of them only containing object $(i,j)''_{k,n^3-t}$. By applying rule $[\,(i,j)''_{k,n^3-t}\,]_{e_{i,j,k}} \rightarrow [\,(i,j)''_{k,n^3-t-1}\,]_{e_{i,j,k}}\ [\,(i,j)''_{k,n^3-t-1}\,]_{e_{i,j,k}}$, we will have 2^{t+1} cells labelled by $e_{i,j,k}$, each of them only containing object $(i,j)''_{k,n^3-t-1}$.

(3) By induction hypothesis, for each i $(1 \leq i \leq n)$ in $\mathcal{C}_t$ there are 2^t cells labelled by c_i, each of them only containing object c_{n^3-t}. By applying rule $[\,c_{n^3-t}\,]_{c_i} \rightarrow [\,c_{n^3-t-1}\,]_{c_i}\ [\,c_{n^3-t-1}\,]_{c_i}$ we will have 2^{t+1} cells labelled by c_i, each of them only containing object c_{n^3-t-1}.

By induction hypothesis, in $\mathcal{C}_t$ there are 2^t cells labelled by h, each of them only containing object h_{n^3-t}, and 2^t cells labelled by y, each of them only containing object y_{n^3-t}. By applying the rules $[\,h_{n^3-t}\,]_h \to [\,h_{n^3-t-1}\,]_h\ [\,h_{n^3-t-1}\,]_h$ and $[\,y_{n^3-t}\,]_y \to [\,y_{n^3-t-1}\,]_y\ [\,y_{n^3-t-1}\,]_y$ we will have 2^{t+1} cells labelled by h, each of them only containing object h_{n^3-t-1}, and 2^{t+1} cells labelled by y, each of them only containing object y_{n^3-t-1}.

(4) By induction hypothesis, in $\mathcal{C}_t$ there is a cell labelled by *no* which contains object w_{n^3-t+6} and object *no*. By applying rule $(no\,,\, w_{n^3-t}\,/\,w_{n^3-t-1}\,,\, 0)$, it will contain object $w_{n^3-(t+1)+6}$ and object *no*. By the way, no rules are applicable to cells *yes* or *out* at configuration $\mathcal{C}_t$.

□

From the previous proposition, we can describe the configuration $\mathcal{C}_{n\cdot p}$ of each computation $\mathcal{C}$ of $\Pi(n)$.

Corollary 5.2 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. Configuration $\mathcal{C}_{n\cdot p}$ verifies the following:*

(1) *There are $2^{n\cdot p}$ cells labelled by in such that each of them contains a different subset of $A'_G = \{(u_i, v_i)'_k \mid 1 \le i \le p \wedge 1 \le k \le n\ \wedge\ (u_i, v_i) \in E\}$, as well as object y, object h and n copies of object c.*

(2) *For each i, j, k $(1 \le i, j, k \le n)$ there are $2^{n\cdot p}$ cells labelled by $e_{i,j,k}$ each of them only containing object $(i,j)''_{k,n^3-n\cdot p}$.*

(3) *For each i $(1 \le i \le n)$ there are $2^{n\cdot p}$ cells labelled by c_i, $2^{n\cdot p}$ cells labelled by h, and $2^{n\cdot p}$ cells labelled by y, only containing object $c_{n^3-n\cdot p}$, object $h_{n^3-n\cdot p}$, object $y_{n^3-n\cdot p}$ respectively.*

(4) *There is a cell labelled by no, a cell labelled by yes and a cell labelled by out such that $\mathcal{C}_{n\cdot p}(no) = \{w_{n^3-n\cdot p+6}, no\}$, $\mathcal{C}_{n\cdot p}(yes) = \{yes\}$, $\mathcal{C}_{n\cdot p}(out) = \{x\}$.*

Theorem 5.3 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. For every t $(n \cdot p + 1 \le t \le n^3)$, configuration $\mathcal{C}_t$ verifies the following:*

(1) *There are $2^{n\cdot p}$ cells labelled by in whose content is equal to the content of those cells in configuration $\mathcal{C}_{n\cdot p}$.*

(2) *For each i, j, k $(1 \le i, j, k \le n)$ there are 2^t cells labelled by $e_{i,j,k}$, each of them only containing object $(i,j)''_{k,n^3-t}$ (by considering $(i,j)''_{k,0} = (i,j)''_k$).*

(3) *For each i $(1 \le i \le n)$ there are 2^t cells labelled by c_i, 2^t cells labelled by h, and 2^t cells labelled by y, only containing object c_{n^3-t} (by considering $c_0 = c'$), object h_{n^3-t} (by considering $h_0 = h'$), object y_{n^3-t} (by considering $y_0 = y'$) respectively.*

(4) *There is a cell labelled by no, a cell labelled by yes and a cell labelled by out such that $\mathcal{C}_t(no) = \{w_{n^3-t+6}, no\}$, $\mathcal{C}_t(yes) = \{yes\}$, $\mathcal{C}_t(out) = \{x\}$.*

Proof: First of all, let us notice that at configuration $\mathcal{C}_{n\cdot p}$ no rule is applicable to any cell labelled by *in*, and no rule is applicable to cell labelled by *yes* or to cell labelled by *out*.

Now, let us show (2), (3) and (4) by induction on t. Let us start analyzing the basic case $t = n \cdot p + 1$.

(2) For each i, j, k $(1 \leq i, j, k \leq n)$, at configuration $\mathcal{C}_{n \cdot p}$ there are $2^{n \cdot p}$ cells labelled by $e_{i,j,k}$, each of them only containing object $(i,j)''_{k,n^3-n \cdot p}$. By applying a rule of the type (4) we will have $2^{n \cdot p+1}$ cells labelled by $e_{i,j,k}$ each of them containing $(i,j)''_{k,n^3-n \cdot p-1}$.

(3) For each i $(1 \leq i \leq n)$, at configuration $\mathcal{C}_{n \cdot p}$ there are $2^{n \cdot p}$ cells labelled by c_i, each of them only containing object $c_{n^3-n \cdot p}$. By applying a rule of the type (6) we will have $2 \cdot 2^{n \cdot p} = 2^{n \cdot p+1}$ cells labelled by c_i whose content is $c_{n^3-n \cdot p-1}$.
At configuration $\mathcal{C}_{n \cdot p}$, there are $2^{n \cdot p}$ cells labelled by h, each of them only containing object $h_{n^3-n \cdot p}$, and $2^{n \cdot p}$ cells labelled by y each of them only containing object $y_{n^3-n \cdot p}$. By applying a rule of type (8) we will have $2 \cdot 2^{n \cdot p} = 2^{n \cdot p+1}$ cells labelled by h whose content is $h_{n^3-n \cdot p-1}$. By applying a rule of type (7) we will have $2 \cdot 2^{n \cdot p} = 2^{n \cdot p+1}$ cells labelled by y whose content is $y_{n^3-n \cdot p-1}$.

(4) At configuration $\mathcal{C}_{n \cdot p}$, there is a cell labelled by *no* which contains object $w_{n^3-n \cdot p+6}$ and object *no*. By applying a rule of type (1), the new content of that cell will be objects $w_{n^3-n \cdot p-1+6}$ and *no*.

By induction hypothesis, let t be such that $n \cdot p+1 \leq t < n^3$ and let us suppose the result holds for t. Let us see that the result also holds for $t + 1$.

First of all, let us notice that at configuration $\mathcal{C}_t$ no rule is applicable to any cell labelled by *in*, and no rule is applicable to cell labelled by *yes* nor to cell labelled by *out* neither.

(2) For each i, j, k $(1 \leq i, j, k \leq n)$, according to induction hypothesis at configuration $\mathcal{C}_t$, there are 2^t cells labelled by $e_{i,j,k}$, and each of them only containing object $(i,j)''_{k,n^3-t}$. By applying a rule of type (4) we will have 2^{t+1} cells labelled by $e_{i,j,k}$ whose content is object $(i,j)''_{k,n^3-t-1}$.

(3) For each i $(1 \leq i \leq n)$, by induction hypothesis at configuration $\mathcal{C}_t$ there are 2^t cells labelled by c_i, each of them only containing object c_{n^3-t}. By applying a rule of the type (6) we will have $2 \cdot 2^t = 2^{t+1}$ cells labelled by c_i whose content is c_{n^3-t-1}.
By induction hypothesis, at configuration $\mathcal{C}_t$ there are 2^t cells labelled by h, each of them only containing object h_{n^3-t}, and 2^t cells labelled by y, each of them only containing object y_{n^3-t}. By applying a rule of type (8) we will have $2 \cdot 2^t = 2^{t+1}$ cells labelled by h whose content is object h_{n^3-t-1}. By applying a rule of the type (7) we will have $2 \cdot 2^t = 2^{t+1}$ cells labelled by y whose content is object y_{n^3-t-1}.

(4) By induction hypothesis, at configuration $\mathcal{C}_t$ there is a cell labelled by *no* which contains object w_{n^3-t+6} and object *no*. By applying a rule of the type (1) the new content of that cell is object $w_{n^3-t-1+6}$ and object *no*.

□

Corollary 5.4 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. Configuration $\mathcal{C}_{n^3}$ verifies the following:*

(1) *There are $2^{n \cdot p}$ cells labelled by in such that each of them contains a different subset of $A'_G = \{(u_i, v_i)'_k \mid 1 \leq i \leq p \wedge 1 \leq k \leq n \ \wedge \ (u_i, v_i) \in E\}$, as well as object y, object h and n copies of object c.*

(2) *For each i, j, k $(1 \leq i, j, k \leq n)$ there are 2^{n^3} cells labelled by $e_{i,j,k}$ each of them only containing object $(i, j)''_k$.*

(3) *For each i $(1 \leq i \leq n)$ there are 2^{n^3} cells labelled by c_i, 2^{n^3} cells labelled by h, and 2^{n^3} cells labelled by y, only containing object c', object h', object y' respectively.*

(4) *There is a cell labelled by no, a cell labelled by yes and a cell labelled by out such that $\mathcal{C}_{n^3}(no) = \{w_6, no\}$, $\mathcal{C}_{n^3}(yes) = \{yes\}$, $\mathcal{C}_{n^3}(out) = \{x\}$.*

Theorem 5.5 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. Configuration $\mathcal{C}_{n^3+1}$ verifies the following:*

(1) *There are $2^{n \cdot p}$ cells labelled by in such that each of them contains a different subset of $A''_G = \{(u_i, v_i)''_k \mid 1 \leq i \leq p \wedge 1 \leq k \leq n \ \wedge \ (u_i, v_i) \in E\}$, as well as object y, object h and n copies of object c.*

(2) *For each i, j, k $(1 \leq i, j, k \leq n)$ there are 2^{n^3} cells labelled by $e_{i,j,k}$ with the same content that at configuration $\mathcal{C}_{n^3}$ except whose cells $e_{v_i,v_j,k}$ in $\mathcal{C}_{n^3}$ which contains objects $(u_i, v_i)''_k \in A''_G$. At configuration $\mathcal{C}_{n^3+1}$ these objects are replaced by $(u_i, v_i)'_k$ respectively.*

(3) *For each i $(1 \leq i \leq n)$ (a) there are 2^{n^3} cells labelled by c_i from which $2^{n \cdot p}$ only contains object c, and the rest only contains object c', (b) there are 2^{n^3} cells labelled by h from which $2^{n \cdot p}$ only contains object h, and the rest only contains object h', and (c) there are 2^{n^3} cells labelled by y from which $2^{n \cdot p}$ only contains object y, and the rest only contains object y'.*

(4) *There is a cell labelled by no, a cell labelled by yes and a cell labelled by out such that $\mathcal{C}_{n^3+1}(no) = \{w_5, no\}$, $\mathcal{C}_{n^3+1}(yes) = \{yes\}$, $\mathcal{C}_{n^3+1}(out) = \{x\}$.*

Proof: It is enough to notice that configuration $\mathcal{C}_{n^3+1}$ is reached from $\mathcal{C}_{n^3}$ by applying:

- Rules of type (9) (from this follows (1) and (2)).
- Rules of type (10), (11), (12) and (1) (from this follows (3) and (4)).

1. Let us remark that at the $(n^3 + 1)$th step, we have replaced objects $(u_i, v_i)'_k$, with $1 \leq i \leq p \wedge 1 \leq k \leq n$, from cells labelled by in, by the respective objects $(u_i, v_i)''_k$ from cells labelled by $e_{u_i,v_i,k}$. Let us recall that at configuration $\mathcal{C}_{n^3}$ we had $2^{n \cdot p - 1}$ copies of objects $(u_i, v_i)'_k$, for each $1 \leq i \leq p \ \wedge \ 1 \leq k \leq n$ in cells labelled by in. Therefore, we need $2^{n \cdot p - 1}$ copies of objects $(u_i, v_i)''_k$ in cells labelled by $e_{u_i,v_i,k}$, but at configuration $\mathcal{C}_{n^3}$ we had 2^{n^3} copies of cells labelled by $e_{u_i,v_i,k}$, each of them only containing object $(u_i, v_i)''_k$.

2. Let us remark that at the (n^3+1)th step we have replaced m copies of object c in cell labelled by in by m respective copies of object c' from cells labelled by c_i $(1 \leq i \leq n)$. Then we need $n \cdot 2^{n \cdot p}$ copies of object c' in cells labelled by c_i. Let us recall that in total we had $n \cdot 2^{n^3}$ cells labelled by c_i, each of them only containing object c'. The same applies to objects h' and y' in cells labelled by h and y respectively.

□

Theorem 5.6 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. Configuration $\mathcal{C}_{n^3+2}$ verifies the following:*

(1) *There are $2^{n \cdot p}$ cells labelled by in such that each of them contains object y'', object h'', n copies of object c'', and a different subset of*

$$A''_G = \{(u_i, v_i)''_k \mid 1 \leq i \leq p \wedge 1 \leq k \leq n \ \wedge \ (u_i, v_i) \in E\}$$

of the form $\{(u_{\alpha_1}, v_{\alpha_1})''_{q_1}, (u_{\alpha_2}, v_{\alpha_2})''_{q_2}, \dots, (u_{\alpha_r}, v_{\alpha_r})''_{q_r}\}$, where

$$q_1 < q_2 < \ldots < q_r \ \wedge \ r \leq n \wedge (q_{i+1} = q_i + 1 \Rightarrow v_{\alpha_i} = u_{\alpha_{i+1}})$$

Moreover, each subset of A''_G verifying the previous conditions is contained inside some cell labelled by in.

(2) *Cells labelled by $e_{i,j,k}$, c_i, h, and y have the same content than at configuration $\mathcal{C}_{n^3+1}$.*

(3) *There is a cell labelled by no, a cell labelled by yes and a cell labelled by out such that $\mathcal{C}_{n^3+2}(no) = \{w_4, no\}$, $\mathcal{C}_{n^3+2}(yes) = \{yes\}$, $\mathcal{C}_{n^3+2}(out) = \{x\}$.*

Proof: It is enough to notice that configuration $\mathcal{C}_{n^3+1}$ yields $\mathcal{C}_{n^3+2}$ by applying rules of types (13), (14), (15), (16) and (1).

The first four rules are symport rules. Thus,at configuration $\mathcal{C}_{n^3+2}$ we will have $2^{n \cdot p}$ cells labelled by in in such manner that the subsets B of A''_G contained in each of them must verify the following conditions:

$(u, v)''_k \in B \wedge (u', v')''_{k'} \in B \wedge (u, v)''_k \neq (u', v')''_{k'} \Rightarrow u \neq u'$ (rule (13)).
$(u, v)''_k \in B \wedge (u', v')''_{k'} \in B \wedge (u, v)''_k \neq (u', v')''_{k'} \Rightarrow v \neq v'$ (rule (14)).
$(u, v)''_k \in B \wedge (u', v')''_{k'} \in B \wedge (u, v)''_k \neq (u', v')''_{k'} \Rightarrow k \neq k'$ (rule (16)).
$(u, v)''_k \in B \wedge (u', v')''_{k+1} \in B \Rightarrow v = u'$ (rule (15)).

Moreover, let us recall that at configuration $\mathcal{C}_{n^3+1}$, every subset B of A''_G is contained inside a different cell labelled by in. Therefore, each subset B of A''_G verifying the previous conditions will be contained inside one unique cell labelled by in at configuration $\mathcal{C}_{n^3+2}$.

□

Remark: From Proposition 4.2, we deduce that a subset B from A''_G represents a Hamiltonian cycle of G if and only if $|B| = n$ and B satisfies the conditions $(\alpha), (\beta), (\gamma), (\delta)$. Thus, to determine whether or not graph G has a Hamiltonian cycle will be equivalent to determine whether or not in some cell of configuration $\mathcal{C}_{n^3+2}$ labelled by in there exists a subset B from A''_G whose cardinality is n.

Theorem 5.7 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. Configuration $\mathcal{C}_{n^3+3}$ verifies the following:*

(1) *There are $2^{n \cdot p}$ cells labelled by in such that each of them contains a copy of object y''' and a copy of object h'''. Besides, if a subset $B \subseteq A''_G = \{(u_i, v_i)''_k \mid 1 \leq i \leq p \wedge 1 \leq k \leq n \wedge (u_i, v_i) \in E\}$ contained in a cell labelled by in has exactly n elements, then no object c'' appears inside that cell. Otherwise, inside any cell labelled by in which contains a subset $B \subseteq A''_G$, some objects c'' (exactly $n - t_1$ copies, where t_1 is the size of the subset B) will remain.*

(2) *Cells labelled by $e_{i,j,k}$, c_i, h, and y have the same content than at configuration $\mathcal{C}_{n^3+1}$.*

(3) *There is a cell labelled by no, a cell labelled by yes and a cell labelled by out such that $\mathcal{C}_{n^3+3}(no) = \{w_3, no\}$, $\mathcal{C}_{n^3+3}(yes) = \{yes\}$, $\mathcal{C}_{n^3+3}(out) = \{x\}$.*

Proof: It is enough to notice that configuration $\mathcal{C}_{n^3+2}$ yields $\mathcal{C}_{n^3+3}$ by applying rules of types (20), (21), (22), and (1).

By applying rules of types (21) and (22), objects h'' and y'' evolve to h''' and y''' respectively.

By applying rules of type (20), for each element $(u, v)''_k$ in the set encoded by that cell, one object c'' will be consumed.

By applying a rule of type (1), object w_4 evolves to object w_3.

□

Corollary 5.8 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. The following assertions are equivalent:*

- *Graph G has a Hamiltonian cycle.*
- *At configuration $\mathcal{C}_{n^3+3}$, there is, at least, a cell labelled by in such that it does not contain any object c''.*

Proof: It suffices to notice that, at configuration $\mathcal{C}_{n^3+2}$, Hamiltonian cycles are characterized by membranes labelled by *in* which contain a subset of A''_G of size n. Then, by using a rule of type (20), at configuration $\mathcal{C}_{n^3+3}$ Hamiltonian cycles are characterized by membranes labelled by *in* such that they do not contain any object c''.

Theorem 5.9 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. Configuration $\mathcal{C}_{n^3+4}$ verifies the following properties:*

(1) *There are $2^{n \cdot p}$ cells labelled by in such that each of them contains one copy of objects y''''. Besides, if object c'' appeared in some cell labelled by in at configuration $\mathcal{C}_{n^3+3}$, then that copy of c'' and object h''' are released out to the environment. Otherwise, object h''' will remain inside that cell in at configuration $\mathcal{C}_{n^3+4}$.*

(2) *Cells labelled by $e_{i,j,k}$, c_i, h, and y have the same content than at configuration $\mathcal{C}_{n^3+1}$.*

(3) *There is a cell labelled by no, a cell labelled by yes and a cell labelled by out such that* $\mathcal{C}_{n^3+4}(no) = \{w_2, no\}$, $\mathcal{C}_{n^3+4}(yes) = \{yes\}$, $\mathcal{C}_{n^3+4}(out) = \{x\}$.

Proof: It is enough to notice that configuration $\mathcal{C}_{n^3+3}$ yields $\mathcal{C}_{n^3+4}$ by applying rules of types (23), (24), and (1).

By applying rules of type (23), object h'' and a copy of object c'' (if any in a cell labelled by *in*) will be released to the environment.

By applying a rule of type (24), object y''' evolves to object y''''.

By applying a rule of type (1), object w_3 evolves to object w_2.

□

Corollary 5.10 *Let* $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ *be a computation of the tissue P system* $\Pi(n)$. *The following assertions are equivalent:*

- *Graph G has a Hamiltonian cycle.*
- *At configuration* $\mathcal{C}_{n^3+4}$, *there is, at least, a cell labelled by in at configuration* $\mathcal{C}_{n^3+4}$ *such that it contains an object* h'''.

Besides, graph G has exactly q Hamiltonian cycles if and only if there are exactly $n \cdot q$ *cells labelled by in such that at configuration* $\mathcal{C}_{n^3+4}$, *it contains an object* h'''.

Proof: It suffices to notice that, at configuration $\mathcal{C}_{n^3+3}$, Hamiltonian cycles are characterized by membranes labelled by *in* such that they do not contain any object c''. Then, a rule of type (23) will be applicable to each cell labelled by *in* that contains some object c''. In this case, at configuration $\mathcal{C}_{n^3+4}$, object h''' will only appear at membranes labelled by *in* which encode Hamiltonian cycles.

Theorem 5.11 *Let* $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ *be a computation of the tissue P system* $\Pi(n)$. *Configuration* $\mathcal{C}_{n^3+5}$ *verifies the following:*

(1) *There are* $2^{n \cdot p}$ *cells labelled by in such that each of them contains one copy of objects* y'''. *Besides, if object* h''' *appeared in some cell labelled by in at configuration* $\mathcal{C}_{n^3+4}$, *then object* h''' *together with object* y''' *are sent to cell labelled by yes. Otherwise, that cell in remain unchanged at the next configuration.*

(2) *Cells labelled by* $e_{i,j,k}$, c_i, h, *and* y *have the same content than at configuration* $\mathcal{C}_{n^3+1}$.

(3) *There is a cell labelled by no such that:* $\mathcal{C}_{n^3+5}(no) = \{w_1, no\}$.

(4) *There is a cell labelled by yes which contains either only object yes, or* $n \cdot q$ *copies of objects* y'''' *and* $n \cdot q$ *copies of objects* h''', *and object yes, being q the total number of Hamiltonian cycle of G. Besides, there is a cell labelled by out which contains only object x.*

Proof: It is enough to notice that configuration $\mathcal{C}_{n^3+4}$ yields $\mathcal{C}_{n^3+5}$ by applying rules of types (23), (24), and (1).

By applying rules of type (25) to each cell labelled by *in* that encodes a Hamiltonian cycle, object h''' and a copy of object c'' will be released to the environment. If G has exactly q Hamiltonan cycles, then there are exactly $n \cdot q$ cells labelled by

in that encodes a Hamiltonian cycle. In this case, the contents of cell labelled by *yes* will be $n \cdot k$ copies of object y'''', $n \cdot k$ copies of object h''' and object *yes*.

By applying a rule of type (1), object w_2 evolves to object w_1 in cell labelled by *no*.

□

Corollary 5.12 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. The following assertions are equivalent:*

- *Graph G has a Hamiltonian cycle.*
- *At configuration $\mathcal{C}_{n^3+5}$, cell labelled by yes has, at least, a copy of object y'''' and a copy of object yes.*

Besides, graph G has exactly q Hamiltonian cycles if and only if at configuration $\mathcal{C}_{n^3+5}$, the cell labelled by yes has exactly $n \cdot q$ copies of objects y''' and a copy of object yes.

Proof: It suffices to notice that, at configuration $\mathcal{C}_{n^3+4}$, Hamiltonian cycles are characterized by membranes labelled by *in* which contain object h'''. Then, a rule of type (25) will be applicable to these cell, producing object h'''' in the cell labelled by *yes*.

Theorem 5.13 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. Configuration $\mathcal{C}_{n^3+6}$ verifies the following:*

(1) *Cells labellled by in, $e_{i,j,k}$, c_i, h, and y have the same content than at configuration $\mathcal{C}_{n^3+5}$.*
(2) *There is a cell labelled by no which contains object w and object no.*
(3) *There is a cell labelled by yes which contains either only object yes (in this case there is a cell labelled by out which contains only object x), or contains $n \cdot q - 1$ copies of objects y'''' and $n \cdot q$ copies of objects h''', being q the total number of Hamiltonian cycles of G (in this case, there is a cell labelled by out which contains object y'''', object yes and object x).*

Proof: It is enough to notice that configuration $\mathcal{C}_{n^3+5}$ yields $\mathcal{C}_{n^3+6}$ by applying rules of types (26) and (2).

If there is, at least, an object y'''' in cell labelled by *yes*, then by applying rule (26), a copy of object y'''' and object *yes* are sent to the cell labelled by *out*. Otherwise, that rule is not applicable to cell *yes*.

In any case, by applying rule (2) to cell labelled by *no*, object w_1 evolves to object w.

□

Corollary 5.14 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. The following assertions are equivalent:*

- *Graph G has a Hamiltonian cycle.*

- *At configuration $\mathcal{C}_{n^3+6}$, cell labelled by out has a copy of object yes and a copy of object x.*

Proof: It suffices to notice that, at configuration $\mathcal{C}_{n^3+5}$, Hamiltonian cycles are characterized by the following condition: cell labelled by *yes* contains objects h'''' and *yes*. Then, a rule of type (26) is applicable to cell labelled by *yes* sending these objects to the celll labelled by *out*.

Theorem 5.15 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. Configuration $\mathcal{C}_{n^3+7}$ verifies the following properties:*

(1) *Cells labelled by in, $e_{i,j,k}$, c_i, h, y, and yes have the same content than at configuration $\mathcal{C}_{n^3+6}$.*
(2) *If G has a Hamiltonian cycle, then $\mathcal{C}_{n^3+7}(no) = \emptyset$, $\mathcal{C}_{n^3+7}(yes) = \{h'''\}$, $\mathcal{C}_{n^3+7}(out) = \{y''', w, no\}$, and $yes, x \in \mathcal{C}_{n^3+7}(0)$. The configuration $\mathcal{C}_{n^3+7}$ is a halting configuration. Moreover, it is an accepting configuration.*
(3) *If G doesn't have a Hamiltonian cycle, then $\mathcal{C}_{n^3+7}(no) = \emptyset$, $\mathcal{C}_{n^3+7}(yes) = \{yes\}$, $\mathcal{C}_{n^3+7}(out) = \{w, no, x\}$.*

Proof: It is enough to notice that configuration $\mathcal{C}_{n^3+6}$ yields $\mathcal{C}_{n^3+7}$ by applying rules of the type (27), in the case that G has a Hamiltonian cycle, and (28) in any case. Besides, by applying rule of type (27) objects x and *yes* are sent to the environment. Thus, $x \notin \mathcal{C}_{n^3+7}(out)$ and the rule of type (29) will not be applicable at the next step. Hence, configuration $\mathcal{C}_{n^3+7}$ is an accepting configuration.

□

Theorem 5.16 *Let $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \dots)$ be a computation of the tissue P system $\Pi(n)$. Let us suppose that G doesn't have a Hamiltonian cycle, then configuration $\mathcal{C}_{n^3+8}$ verifies the following:*

(1) *Cells labelled by in, $e_{i,j,k}$, c_i, h, y, no and yes, have the same content than at configuration $\mathcal{C}_{n^3+7}$.*
(2) *$\mathcal{C}_{n^3+8}(no) = \emptyset$, $\mathcal{C}_{n^3+8}(yes) = \{yes\}$, $\mathcal{C}_{n^3+8}(out) = \{w\}$, and $no, x \in \mathcal{C}_{n^3+8}(0)$. The configuration $\mathcal{C}_{n^3+8}$ is a halting configuration. Moreover, it is a rejecting configuration.*

Proof: It is enough to notice that if G doesn't have a Hamiltonian cycle, then configuration $\mathcal{C}_{n^3+8}$ is reached from $\mathcal{C}_{n^3+7}$ by applying the rule of type (29).

□

Corollary 5.17 *The family $\mathbf{\Pi}$ defined at Section 4.2 is polynomially bounded with regard to* $(\texttt{HAM-CYCLE}, cod, s)$.

Proof: From Theorem 5.15 and Theorem 5.16, we deduce that any computation $\mathcal{C}$ of the tissue P system $\Pi(n)$ spends $n^3 + 7$ or $n^3 + 8$ transition steps, for each $n \in \mathbb{N}$.

□

Corollary 5.18 *The family* $\mathbf{\Pi}$ *defined at Section 4.2 is sound and complete with regard to* (HAM-CYCLE, *cod*, *s*)

Proof: Let G be a directed graph that has a Hamiltonian cycle. Let $\mathcal{C}$ be an arbitrary computation of $\Pi(s(G)) + cod(G)$. From Theorem 5.15, we deduce that $\mathcal{C}$ is an accepting computation.

Now, let G be a directed graph such that there exists an accepting computation $\mathcal{C}$ of $\Pi(s(G)) + cod(G)$. Then, G has a Hamiltonian cycle. Otherwise, computation $\mathcal{C}$ must be a rejecting computation according to Theorem 5.16.

□

Theorem 5.19 HAM-CYCLE $\in \mathbf{PMC}_{TDC(2)}$.

Proof: The family of tissue P systems with cell division constructed in Subsection 4.2 verifies the following:

(a) Every system of the family $\mathbf{\Pi}$ is a recognizer tissue P system with cell division and communication rules with length at most 2.
(b) The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines (Subsection 5.1).
(c) The pair (cod, s) of polynomial–time computable functions defined in Subsection 4.3 verifies: for each instance G of HAM-CYCLE, $s(G)$ is a natural number, $cod(G)$ is an input multiset of the system $\Pi(s(G))$, and for each $n \in \mathbb{N}$, $s^{-1}(n)$ is a finite set.
(d) The family $\mathbf{\Pi}$ is polynomially bounded with regard to (HAM-CYCLE, *cod*, *s*) (Corollary 5.17).
(e) The family $\mathbf{\Pi}$ is sound and complete with regard to (HAM-CYCLE, *cod*, *s*) (Corollary 5.18).

Therefore, according to Definition 2.4, the uniform family $\mathbf{\Pi}$ of tissue P systems constructed in Section 4 solves the HAM-CYCLE problem in polynomial time with respect to the number of variables and the number of clauses.

□

Corollary 5.20 $\mathbf{NP} \cup \mathbf{co\text{-}NP} \subseteq \mathbf{PMC}_{TDC(2)}$.

Proof: It suffices to notice that the HAM-CYCLE problem is **NP**-complete, HAM-CYCLE$\in \mathbf{PMC}_{TDC(2)}$, and this complexity class is closed under polynomial-time reduction and under complement.

□

6 Conclusions

The length of communication rules plays a relevant role for tissue P systems with cell division from the efficiency point of view. A uniform and efficient solution to the Vertex Cover problem by using a family of tissue P systems with cell division and communication rules of length at most 3 was given in [1]. By using the dependency

graph technique of cell–like P systems, it was shown that only tractable problems can be efficiently solved by using families of tissue P systems with cell division and communication rules of length 1 [3]. Hence, assuming that $\mathbf{P} \neq \mathbf{NP}$, in the framework of tissue P systems with cell division, passing from communication rules of length 1 to communication rules of length at most 3 amounts to passing from non–efficiency to efficiency.

In this paper, that borderline of efficiency has been optimized by proving that a well known **NP**–complete problem, the `HAM-CYCLE` problem, can be solved in a uniform and efficient way, by using a family of tissue P systems with cell division and communication rules of length at most 2.

In [7], cell separation rules were introduced into tissue P systems (inspired by the cellular fission) and their computational efficiency was investigated. Two important results were obtained in that framework: (a) only tractable problems can be efficiently solved by using cell separation and communication rules with length at most 1, and (b) a uniform and linear time solution to the `SAT` problem by using cell separation and communication rules with length at most 8 was presented. Recently [14] this result was improved by showing a family of tissue P systems with cell separation and communication rules with length at most 3, solving the `SAT` problem in a uniform way and linear time.

Now, we propose three open problems related to the efficiency of tissue P systems:

(a) What is the computational efficiency of tissue P systems with cell separation which allow communication rules with length at most 2?
(b) What happens if only symport (or only antiport) rules are allowed in tissue P systems with cell division or cell separation?
(c) At the initial configuration of a tissue P system the symbols of the alphabet $\mathcal{E}$ appear in the environment in an *arbitrary number of copies*. We can consider tissue P systems without environment, that is, tissue P systems where alphabet $\mathcal{E}$ is empty. What is the relationship between the polynomial complexity classes of tissue P systems with cell division (or with cell separation) and the corresponding tissue P systems without environment?

Acknowledgements

The work was supported by TIN2009-13192 Project of the Ministerio de Ciencia e Innovación of Spain and Project of Excellence with *Investigador de Reconocida Valía*, from Junta de Andalucía, grant P08 – TIC 04200. Antonio E. Porreca was partially supported by Università degli Studi di Milano-Bicocca, Fondo di Ateneo per la Ricerca (FAR) 2011. Research by Niall Murphy was partly supported by a PICATA postdoctoral fellowship of the Moncloa Campus of International Excellence (UCM-UPM)

References

1. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero–Campero. Computational efficiency of cellular division in tissue-like P systems. *Romanian Journal of Information Science and Technology* **11**, 3, (2008), 229–241.
2. M.R. Garey, D.S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, (1979).
3. R. Gutiérrez-Escudero, M.J. Pérez-Jiménez, M. Rius–Font. Characterizing tractability by tissue-like P systems. *Lecture Notes in Computer Science* **5957**, (2010), 289–300.
4. M. Ito, C. Martín Vide, Gh. Păun. A characterization of Parikh sets of ET0L laguages in terms of P systems. In M. Ito, Gh. Păun, S. Yu (eds.) *Words, Semigroups and Transducers*, World Scientific, Singapore, 2001, 239-254.
5. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón. A New Class of Symbolic Abstract Neural Nets: Tissue P Systems. *Lecture Notes in Computer Science* **2387**, (2002), 290–299.
6. L. Pan, T.-O. Ishdorj. P systems with active membranes and separation rules. *Journal of Universal Computer Science*, **10**, 5, (2004), 630–649.
7. L. Pan, M.J. Pérez-Jiménez. Computational complexity of tissue–like P systems. *Journal of Complexity*, **26**, 3 (2010), 296–315.
8. Gh. Păun. Attacking **NP**-complete problems. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou, C. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, pp. 94-115.
9. Gh. Păun. *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, (2002).
10. A. Păun, Gh. Păun. The power of communication: P systems with symport/antiport. *New Generation Computing*, **20**, 3, (2002), 295–305.
11. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez. Tissue P System with cell division. *In. J. of Computers, Communications and Control*, **3**, 3, (2008), 295–303.
12. M.J. Pérez-Jiménez, A. Romero-Jiménez, Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265–285.
13. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini. A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics*, **11**, 4, (2006), 423-434.
14. M.J. Pérez-Jiménez, P. Sosík. Improving the efficiency of tissue P systems with cell separation. Submitted, 2012.
15. H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer–Verlag, New York, (1999).

Cell Complexes and Membrane Computing for Thinning 2D and 3D Images

Raúl Reina-Molina[1], Daniel Díaz-Pernil[1], Miguel A. Gutiérrez-Naranjo[2]

[1]Research Group on Computational Topology and Applied Mathematics
Department of Applied Mathematics
University of Sevilla
raureimol@alum.us.es, sbdani@us.es

[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
magutier@us.es

Summary. In this paper, we show a new example of bridging Algebraic Topology, Membrane Computing and Digital Images. In [24], a new algorithm for thinning multidimensional black and white digital images by using cell complexes was presented. Such cell complexes allow a discrete partition of the space and the algorithm preserves topological and geometrical properties of the image. In this paper, we present a parallel adaptation of such algorithm to P systems, by introducing some concepts of Algebraic Topology in the Membrane Computing framework. The chosen model for the implementation is tissue-like P systems with promoters, inhibitors and priorities.

1 Introduction

Computer vision [36] is one of the challenges for Computer Science in the next years. From a biological point of view, vision is an extremely complex process involving the transformation of the light energy into a signal which leaves the eye by way of the optic nerve and arrives to the brain, where is interpreted. From the computational side, a 2D digital image can be roughly defined as a function from a two dimensional surface which maps each point from the surface onto a set of attributes as bright or color. Analogously, a 3D image maps a region of a tridimensional space onto a set of attributes. The different treatments of such mappings provide a big amount of current applications in computer vision as biometrics [1], surveillance [11] or medical imaging [2].

Many problems in the processing of 2D or 3D digital images have features which make it suitable for techniques inspired by nature. The subset of the integer plane or space taken to be the *support* of the image and the set of possible features

associated to each 2D or 3D point can be considered finite and hence, the transformation of an image into another can be made in a *discrete* way. Other of such features is that the treatment of the image can be parallelized and locally modified. Regardless how large is the picture, the process can be performed in parallel in different local areas of it. Another interesting feature is that the information of the image can also be easily encoded in the data structures used in Natural Computing.

In the literature, we can find many examples of the use of Natural Computing techniques for dealing with such problems. One of the classic examples is the use of cellular automata [33, 35]. Other efforts are related to artificial neural networks as in [18, 38].

In Membrane Computing, there is a large tradition in the study of dealing information structured as two dimensional objects (see, e.g., [5, 6, 12, 23]). The main motivation for these studies is to bring together Membrane Computing and Picture Grammars. From a technical point of view, arrays are two-dimensional objects placed inside the membranes as strings are one-dimensional objects in the model of P systems with string objects [19, 31].

Recently, a new research line has been open by applying well-known membrane computing techniques for solving problems from digital imagery. For example, the *segmentation* problem, [8, 10, 13, 14], *thresholding* [7] or *smoothing* [29]. Special attention deserves Gimel'farb *et al.* [20], where the *symmetric dynamic programming stereo* (SDPS) algorithm [21] for stereo matching was implemented by using simple P modules with duplex channels.

In this paper, we focus on the problem of skeletonizing a 2D or 3D image. Skeletonization is one of the approaches for representing a shape with a small amount of information by converting the initial image into a more compact representation and keeping the meaning features. The conversion should remove redundant information, but it should also keep the basic structure. There are many different definitions of the skeleton of a black and white image and many skeletonizing algorithms[1], but in general, the image B is a skeleton of the image A, if it has fewer black pixels than A, preserves its topological properties and, in some sense, keeps its *meaning*. The most important features concerning a shape are its *topology* (represented by connected components, holes, etc.) and its *geometry* (elongated parts, ramifications, etc.), thus these terms have to be preserved. When the skeletonizing process is made by the iterative removal of non-significant elements of the image, the process is known as *thinning*.

In this paper, we present an implementation of the Liu's algorithm [24] for thinning images based on Membrane Computing techniques. The basic notion of this algorithm is the *cell complex*. It can be seen as a mathematical abstraction of a space unit. This space unit is built in some n dimensional space and embedded in a space of higher dimension, as a 2-dimensional square can be embedded in a 3D space. All these concepts will be formalized below.

[1] A detailed description of skeletonizing algorithms is out of the scope of this paper. For a survey in this topic, see e.g., [34].

In Liu's work [24], a cell complex is processed in order to obtain another complex with the same topology, and the same geometry. We will start from a black and white 2D or 3D digital image by building a cell complex from it. This complex will be, then, processed by consecutive parallel removal of certain cells. The removal process does not change the topology nor the geometry of the starting cell complex. At the end of this process, the set of non-removed cells will make the skeleton.

For implementing these ideas in the Membrane Computing framework, we present a family of tissue-like P systems endowed with priorities among rules, promoters and inhibitors. This paper follows the research line open with [9], but, to the best of our knowledge, this is the first work which put together Membrane Computing, Cells Complexes and thinning processes.

The paper is organized as follows. In the first section, all technical requirements of Algebraic Topology are reviewed. Next, the basics for understanding the proposed algorithm are introduced, followed by the presentation of the Membrane Computing framework and the bioinspired 2D and 3D black and white image thinning algorithm. Next, an overview of the computation is presented, finishing with conclusions and future work.

2 Cubical Complexes

As pointed above, cubical complexes are mathematical abstractions to handle structured portions of a n dimensional space. On such abstractions, we can define several operators as the *border* one, which associates, for example, a 3D cell (cube) with six 2D cells (squares), or properties to define *free cells* or *isolated cells*.

We follow T. Kaczyński, K. Mischaikow and M. Mrozek [22] in the description of a kind of combinatorial structure on a topological space.

Definition 1. *[22] An* elementary interval *is a closed interval $\overline{I} \subset \mathbb{R}$ of the form $\overline{I} = [l, l+1]$ or $\overline{I} = [l, l]$ for some $l \in \mathbb{Z}$. The former are called* nondegenerated*, while the latter are called* degenerated*. The interval $[l, l]$ that contains only one point will be denoted by $[l]$.*

Degenerated elementary intervals are simply points with 0 dimensions. Nondegenerated elementary intervals are segments (objects with one dimension). Next, we generalize this notion to any dimension.

Definition 2. *An elementary cube $\overline{\sigma}$ is a finite product of elementary intervals:*

$$\overline{\sigma} = \overline{I}_1 \times \overline{I}_2 \times \cdots \times \overline{I}_d \subset \mathbb{R}^d$$

where each $\overline{I}_j$ is an elementary interval, $j \in \{1, \ldots, d\}$. The set of all elementary cubes in $\mathbb{R}^d$ is denoted by $\mathcal{K}^d$. The set of all elementary cubes is

$$\mathcal{K} = \bigcup_{d=1}^{\infty} \mathcal{K}^d$$

For example $\{(0,0,0)\}$, $\{(x,0,0)\,|\,0 \leq x \leq 1\}$, $\{(x,y,0)\,|\,0 \leq x,y \leq 1\}$ and $\{(x,y,z)\,|\,0 \leq x,y,z \leq 1\}$ are elementary cubes. Given an elementary cube $\overline{\sigma} = \overline{I}_1 \times \overline{I}_2 \times \cdots \times \overline{I}_d$ in $\mathbb{R}^d$, its *embedding number* d is denoted by $\operatorname{emb} \overline{\sigma}$. The dimension of $\overline{\sigma}$ is defined to be the number of nondegenerated intervals in its definition and is denoted by $\dim \overline{\sigma}$. In this way, for the elementary cube $Q \equiv \{(x,y,0)\,|\,0 \leq x,y \leq 1\}$, $\operatorname{emb} Q$ is 3 and $\dim Q$ is 2.

The set of all elementary cubes with dimension p is denoted by $\mathcal{K}_p$. The set of all elementary cubes in $\mathbb{R}^d$ with dimension p is denoted by $\mathcal{K}_p^d$.

The following definition gives sense to the decomposition of elementary cubes into lower-dimensional objects.

Definition 3. *Let $\overline{\delta}$ and $\overline{\sigma}$ be two elementary cubes of any dimension. If $\overline{\delta} \subset \overline{\sigma}$, then $\overline{\delta}$ is a* face *of $\overline{\sigma}$. If $\overline{\delta}$ is a face of $\overline{\sigma}$ and $\overline{\delta} \neq \overline{\sigma}$, then $\overline{\delta}$ is a* proper face *of $\overline{\sigma}$. $\overline{\delta}$ is a* primary face *of $\overline{\sigma}$ if it is a face of $\overline{\sigma}$ and $\dim \overline{\delta} = \dim \overline{\sigma} - 1$. Given an elementary cube $\overline{\sigma} \in \mathcal{K}_p^d$, the set of all primary faces of $\overline{\sigma}$ is called the* border *of $\overline{\sigma}$ and it is denoted by $\partial\,\overline{\sigma}$.*

For example, let us consider the elementary cubes $\overline{\sigma}_1 = \{(x,0,0)\,|\,0 \leq x \leq 1\}$, $\overline{\sigma}_2 = \{(x,y,0)\,|\,0 \leq x,y \leq 1\}$ and $\overline{\sigma}_3 = \{(x,y,z)\,|\,0 \leq x,y,z \leq 1\}$. Notice that $\overline{\sigma}_1 \subseteq \overline{\sigma}_2 \subseteq \overline{\sigma}_3$ holds, and hence $\overline{\sigma}_1$, $\overline{\sigma}_2$ and $\overline{\sigma}_3$ are faces of $\overline{\sigma}_3$; $\overline{\sigma}_1$ and $\overline{\sigma}_2$ are proper faces of $\overline{\sigma}_3$; $\overline{\sigma}_1$ is a primary face of $\overline{\sigma}_2$ and $\overline{\sigma}_2$ is a primary face of $\overline{\sigma}_3$. We also have that $\partial\,\overline{\sigma}_2 = \{\overline{\sigma}_1, \overline{\sigma}_1', \overline{\sigma}_1'', \overline{\sigma}_1'''\}$ with $\overline{\sigma}_1' = \{(x,1,0)\,|\,0 \leq x \leq 1\}$, $\overline{\sigma}_1'' = \{(0,x,0)\,|\,0 \leq x \leq 1\}$, $\overline{\sigma}_1''' = \{(1,x,0)\,|\,0 \leq x \leq 1\}$.

Definition 4. *Let $\overline{I}$ be an elementary interval. The associated* elementary cell *is*

$$I = \begin{cases} (l, l+1) & \text{if } I = [l, l+1], \\ [l] & \text{if } I = [l]. \end{cases}$$

Let $\overline{\sigma} = \overline{I}_1 \times \overline{I}_2 \times \cdots \times \overline{I}_d \subset \mathbb{R}^d$ be an elementary cube, the associated elementary cell *is*

$$\sigma = I_1 \times I_1 \times \cdots \times I_d$$

The *dimension* of an *elementary cell* σ is defined as $\dim \overline{\sigma}$, i.e., the dimension of the associated elementary cube. The border for an elementary cell σ can also be defined as the set $\partial\,\sigma = \{\delta : \overline{\delta} \in \partial\,\overline{\sigma}\}$.

Definition 5. *A* cubical complex *is a set of elementary cells such that, given an elementary cell σ in the complex, all of its principal faces (the cells in $\partial\,\sigma$) are in the complex.*

For the sake of simplicity, hereafter we will say *cells* instead of elementary cells, bearing in mind that we refer to such kind of objects.

For example, Figure 1 (left) shows the cubical complex

$$K = \{ABCD, AC, CD, BD, AB, BE, A, B, C, D, E\}$$

This cubical complex has 1 cell of dimension 2 ($ABCD$), 5 cells of dimension 1 (AC, CD, BD, AB, BE) and 5 cells of dimension 0 (A, B, C, D, E).

When a cell is not a proper face of any cell in a given cell complex, it will be called *isolated cell*. A cell that is a proper face of exactly one cell in the complex is called *free face*. The following proposition links the concepts of free faces, proper faces and dimension. The proof can be found in [22].

Proposition 1. *Let δ be a free face in a cell complex and assume δ is a proper face of σ. Then σ is an isolated cell and* $\dim \delta = \dim \sigma - 1$.

As we are interested in obtaining a simpler representation for a cell complex whilst the topology is preserved. In the following definition, a way to *reduce* the number of cells in a cell complex is presented. This process reduces the number of cells by two and it does not change the topology of the cell complex.

For example, let us consider the cell complex of Figure 1 (left). The cells $ABCD$ and BE are isolated. The cells AC, CD, BD AB and E rare free faces, but A, B, C and D are not free faces, since they are proper faces of more than 1 cell complex.

Definition 6. *Let K be a cubical complex and δ a free cell in K. Let σ be the only cell in K such that δ is a proper face of σ. Let $K' = K \setminus \{\delta, \sigma\}$. K' is obtained from K via a process called* elementary collapse *of σ by δ.*

Let us consider again the cell complex K of Figure 1 (left). The cell E is a free face of BE and, hence, we can consider the elementary collapse of BE by E. The effect of such elementary collapse is the removal of E and BE from the cell complex K. Analogously, AC is a free face of $ABCD$. The elementary collapse of $ABCD$ by AC is the removal of both cells ($ABCD$ and AC) from K. Figure 1 (right) shows the final cubical complex obtained after both collapses.

Definition 7. *Let K be a cubical complex. A pair of cells $\langle \delta, \sigma \rangle$ is said to be a* simple pair *if following conditions hold:*

- *δ is a free cell in K.*
- *σ is the only cell such that $\delta \in \partial\, \sigma$.*

The cell σ is called the facet *of the simple pair.*

As shown in related literature [22, 37], simple pairs removal does not change the topology of the given cell complex.

3 Cell Complex Thinning

Skeletonization is usually considered as a pre-process in pattern recognition algorithms, but its study is also interesting by itself for the analysis of line-based

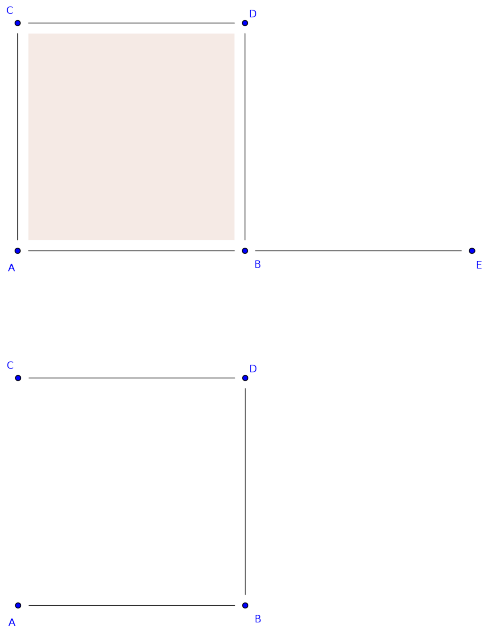


Fig. 1. Elementary collapse example: E collapses onto BE and AC collapses onto $ABDC$ in the image at the left, producing the image at the right.

images as texts, line drawings, human fingerprints or cartography. Skeletonization is a common transformation in Image Analysis. The concept of skeleton was introduced by Blum in [3], under the name of medial axis transform.

Let K be a cubical[2] cell complex and let ∂ be its border operator. As seen in the previous section, if only simple pairs of cells are removed, the topology is kept. For geometry preservation it is necessary to require some additional properties to those cells to be removed.

The basic idea of the algorithm is to define an iterative process where *outer* cells are removed. Here, the idea of *outer* cells makes reference to simple pairs, since in a simple pair $\langle \delta, \sigma \rangle$ the cell δ is a "terminal" cell as it does not lie in the border of any other one rather than σ.

In the process of iterative thinning, given a cell σ, we will denote the later iteration when σ is the facet of a simple pair by $R(\sigma)$. The earlier iteration when σ becomes isolated will be denoted by $I(\sigma)$. Liu *et al.* describe in [25] the relation between $I(\sigma)$ and $R(\sigma)$, and the maximum *isotropic* elongation in $p+1$ and p

[2] In the original work by Liu, [24], the thinning algorithm is designed for cell complexes of any kind, however we restrict to cubical complexes.

directions, respectively, since $\dim \sigma = p$. Thus, if σ is a p-cell in a cell complex, $I(\sigma)$ measures the shortest discrete distance from σ to the object boundary. This gives an idea of the size of the maximum disk centered at σ and inscribed in the object. On the other hand, $R(\sigma)$ measures the longest distance from σ to the object boundary going along the skeleton $(p-1)$-cells.

From the observation of the behaviour of previous measures, Liu defined two *difference measures*. The absolute one, $R(\sigma) - I(\sigma)$, is called *absolute medial persistence* and is denoted by MP_{abs}. On the other hand, *relative medial persistence* is defined as $1 - \frac{I(\sigma)}{R(\sigma)}$ and denoted by MP_{rel}. Both of them measure the duration in which a cell remains isolated during thinning process.

The cell complex thinning algorithm is shown in algorithm 1. It starts by initializing the isolated cells. Next, the thinning iterations start. In each iteration, all simple pairs are selected, all the pairs where the facet cell has one of the medial persistence measures less than given thresholds are chosen. Finally, the cells in selected simple pairs are removed from the cell complex. Otherwise, the cells are removed and the thinning iterations stop, else, the iteration counter increases and the thinning iterations continue. When the algorithm halts, a cell complex representing the skeleton for the initial one is obtained.

Algorithm 1 Cell complex thinning algorithm

Require: K cell complex, $\varepsilon_a, \varepsilon_r > 0$
  for all $\sigma \in K$ isolated **do**
    $I(\sigma) \leftarrow 0$
  end for
  iter $\leftarrow 1$
  repeat
    Let $S = \{\langle \delta, \sigma \rangle : \langle \delta, \sigma \rangle$ is a simple pair$\}$
    for all $\sigma \in \pi_2(S)$ **do**
      $R(\sigma) \leftarrow$ iter
    end for
    Let $S' = \{\langle \delta, \sigma \rangle \in S : MP_{abs}(\sigma) < \varepsilon_a \wedge MP_{rel}(\sigma) < \varepsilon_r\}$
    $K = K \setminus \{\sigma, \delta : \langle \delta, \sigma \rangle \in S'\}$
    for all $\sigma \in K$ new isolated cell **do**
      $I(\sigma) \leftarrow$ iter
    end for
    iter $\leftarrow$ iter $+ 1$
  until $S' = \emptyset$

Here $\pi_2(\langle \delta, \sigma \rangle) = \sigma$ is the second projection for the pair $\langle \delta, \sigma \rangle$.

4 Formal Framework

The chosen P system model for a Membrane Computing implementation of the algorithm is the *tissue-like P systems model* endowed with some extra ingredients.

As it is well-known, the biological inspirations of this model are intercellular communication and cooperation between neurons [26, 27]. The communication among cells is based on symport/antiport rules[3]. Tissue-like P systems have been widely used to solve computational problems in other areas (see e.g. [15, 16]), but recently, they have been also used in the study of digital images (e.g., [4, 8, 10, 17, 28, 29]). In this paper, we use a variant of tissue-like P systems where the application of the rules are regulated by *promoters* and *inhibitors*. These promoters have a clear biological inspiration. The rule is applied if the reactants are present, but it is also necessary the presence of all the promoters and none of the inhibitors in the corresponding cell. The promoters are not *consumed* nor *produced* by the application of the rule, but if they are not in the cell, the rule cannot be applied. In one step, each reactant in a membrane can only be used for one rule, but if several rules need the presence of the same promoter, then the presence of one unique copy of the promoter suffices for the application of the rules. In the general case, if there are several possibilities, the rule is non-deterministically chosen, but sometimes we will consider a priority relation between rules, so we need the concept of *priority* in our P systems. Next, we recall the formal definition of these P systems.

Definition 8. *A* tissue-like P system with promoters, inhibitors and priorities *of degree* $q \geq 1$ *is a tuple of the form*

$$\Pi = (\Gamma, \Sigma, \mathcal{E}, w_1, \ldots, w_q, \mathcal{R}, Pri, i_{in}, i_{out})$$

where q *is the number of cells (or membranes) of the P system and*

1. Γ *is a finite alphabet, whose symbols will be called objects. These objects can be placed in the cells or in the surrounding space (called the environment).*
2. $\Sigma \subseteq \Gamma$ *is the input alphabet. The input of the computation performed by the P system is encoded by using this alphabet.*
3. $\mathcal{E} \subseteq \Gamma$ *is a finite alphabet representing the set of the objects in the environment. Following a biological inspiration, the objects in the environment are available in an arbitrary large amount of copies;*
4. $w_1, \ldots, w_q$ *are strings over* Γ *representing the multisets of objects placed inside the cells at the starting of the computation;*
5. $\mathcal{R}$ *is a finite set of rules of the following form:*

$$(pro\, \neg inh \,|\, i, u/v, j), \quad \text{for } 0 \leq i \neq j \leq q,\ pro, inh, u, v \in \Gamma^*$$

6. Pri *is a finite set of relations* $R_i > R_j$*, where* R_i *and* R_j *are rules from* $\mathcal{R}$*. It means that if* R_i *and* R_j *can be applied, then the application of* R_i *has priority on the application of* R_j*.*
7. $i_{in} \in \{1, 2, \ldots, q\}$ *denotes the input cell, i.e., the cell where the input of the computation will be placed.*
8. $i_{out} \in \{1, 2, \ldots, q\}$ *denotes the output cell, i.e., the cell where the output of the computation will be placed.*

[3] Introduced in Membrane Computing in [30].

Informally, a tissue-like P system with promoters, inhibitors and priorities of degree $q \geq 1$ can be seen as a set of q cells labeled by $1, 2, \ldots, q$. The cells are the nodes of a virtual graph, where the edges connecting the cells are determined by the communication rules of the P system, i.e., as usual in tissue-like P systems, the edges linking cells are not provided explicitly: If a rule $(pro\, \neg inh \,|\, i, u/v, j)$ is given, then cells i and j are considered linked. The application of a rule $(pro\, \neg inh \,|\, i, u/v, j)$ consists of trading the multiset u (initially in the cell i) against the multiset v (initially in j). After the application of the rule, the multiset u disappears from the cell i and it appears in the cell j. Analogously, the multiset v disappears from the cell j and it appears in the cell i. The trade can also be between one cell and the environment, labeled by 0. The rule is applied if in the cell with label i the objects of pro are present in the cell i (*promoters*), while any of the objects in inh do not appear in the cell (*inhibitors*). The promoters or the inhibitors are not modified by the application of the rule. If the promoters and inhibitors are empty, we will write $(i, u/v, j)$ instead of $(\emptyset\, \neg\emptyset |\, i, u/v, j)$. Finally, we write $(pro\,|i, u/v, j)$ or $(\neg inh\,|i, u/v, j)$ when only promoters or inhibitors appear, respectively.

As usual, we also consider that some objects not belonging to $\mathcal{E}$ can arrive to the environment during a computation. So, in a configuration (not initial) we could find two types of objects in the environment: Firstly, those which belong to the environment and appear in an arbitrary large number of copies. Secondly, those which not belong to the environment but are been sent to the environment by the application of a rule.

Rules are used as usual in the framework of membrane computing, that is, in a maximally parallel way (a universal clock is considered). A *configuration* is an instantaneous description of the P system and it is represented as a tuple $(w_0, w_1, \ldots, w_q)$, where 'W_0 is the multiset of objects from $\Gamma - \mathcal{E}$ placed in the environment (initially, $w_0 = \emptyset$). Given a configuration, we can perform a computation step and obtain a new configuration by applying the rules in a parallel manner as it is shown above. A configuration is *halting* when no rules can be applied to it. A *computation* is a sequence of computation steps such that either it is infinite or it is finite and the last step yields a halting configuration (i.e., no rules can be applied to it). Then, a computation halts when the P system reaches a halting configuration. The output of a computation is collected from its halting configuration by reading the objects contained in the output cell.

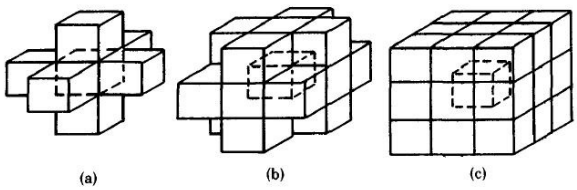
4.1 Image Algebra

Next, we recall some basic definitions from Image Algebra used in thi paper[4].

For a point set $X \subset \mathbb{Z}^2$, a *neighborhood function* is a function $N : X \to 2^{\mathbb{Z}^2}$. For each point $x \in X$, $N(x) \subset \mathbb{Z}^2$. The set $N(x)$ is called a *neighborhood* for x. There are two neighborhood function on subsets of $\mathbb{Z}^2$ which are of particular importance in image processing, the *von Neumann* neighborhood and the *Moore*

[4] A detailed introduction can be found in [32].

neighborhood. The first one, $N : X \to 2^{\mathbf{Z}^2}$, is defined by $N(x) = \{y : y = (x_1 \pm j, x_2)$ *or* $y = (x_1, x_2 \pm k),\ j, k \in \{0, 1\}\}$, where $x = (x_1, x_2) \in X \subset \mathbf{Z}^2$. While the Moore neighborhood $M : X \to 2^{\mathbf{Z}^2}$ is defined by $M(x) = \{y : y = (x_1 \pm j, x_2 \pm k),\ j, k \in \{0, 1\}\}$, where $x = (x_1, x_2) \in X \subset \mathbf{Z}^2$. The von Neumann and Moore neighborhood are also called the *four neighborhood* (4-adjacency) and *eight neighborhood* (8-adjacency), respectively.



(a)6-Neighbors (b) 18-Neighbors (c) 26-Neighbors

Fig. 2. Neighbors of a voxel in a cube

In $\mathbb{Z}^3$ two voxels are said to be 26-adjacent if they are distinct and each coordinate of one differs from the corresponding coordinate of the other by at most 1. Two voxels are 18-adjacent if they are 26-adjacent and differ in at most two of their coordinates; and two voxels are 6-adjacent if they are 26-adjacent and differ in at most one coordinate. That is to say each voxel has three kinds of *neighbors*: 6-neighbors which are also called face neighbors, 18-neighbors which are face and edge neighbors and 26-neighbors which are face, edge, and vertex neighbors, as they are shown in Figure 2. For $n = 4; 8; 6; 18$ or 26 an *n-neighbor* of a voxel p is a point that is n-adjacent to p.

The point sets with the usual operations has an algebra structure (see [32]).

A *Z-valued image* on X is any element of Z^X. Given a Z-valued image $I : X \to Z$, we will refer to Z as the set of possible range values of I, and to X as the spatial domain of I. The graph of an image is also referred to as the *data structure representation* of the image. Given the data structure representation $I = \{(x, I(x)) : x \in X\}$, then an element $(x, I(x))$ is called a *picture element* or *resel*[5]. The first coordinate x of a resel is called the *resel location* or *image point*, and the second coordinate $I(x)$ is called the *resel value* of I at location x.

For example, X could be a subset of $\mathbb{Z}^2$ where $x = (i, j)$ denotes spatial location, and Z could be a subset of $\mathbb{N}$, $\mathbb{N}^3$, etc. We call to the image set of the function I with domain X the *set of colors* or *alphabet of colors* and the image point of each resel is called *associated color*.

[5] The elements of a two-dimensional image are usually called *pixels*; the elements of a three dimensional image are usually called *voxels*, and the elements of a four-dimensional image are usually called *doxels* (*resel* in general).

5 Description of the Algorithm

In previous sections two kinds of objects has been reviewed. On one side, cell complexes achieves an useful link between continuous spaces and discrete structures where combinatorial algorithms may be developed using well-established properties and results by continuous topology. On the other hand, it has been settled a theoretical framework for working with images, considering them as a function from a topological discrete space to a set of "colors".

Our main goal is, starting from a k-dimensional binary image, build another image which represents a skeleton for the original one. In this process we will get a cell complex from the original image, skeletonize it and build back an image from the last skeleton. In this process no topological or shape information will be lost.

The set of points for our source images will be the set $[0,n)^k = \{0,1,\ldots,n-1\}^k \subset \mathbb{Z}^k$ equipped with a *cubic* neighbourhood function, described as follows: Two resels $\mathbf{i} = (i_1,\ldots,i_p,\ldots,i_k)$ and $\mathbf{j} = (j_1,\ldots,j_p,\ldots,j_k)$ are to be said $2k$-adjacent if $i_l = j_l$ for $l \neq p$ and $|i_p - j_p| = 1$. More formally, the neighbourhood function is given by

$$N(i_1,\ldots,i_k) = \left\{ (j_1,\ldots,j_k) \in [0,n)^k : j_l = \begin{cases} i_l & \text{if } l \neq p \\ i_l \pm 1 & \text{if } l = p \end{cases} ; 1 \leq p \leq k \right\}$$

This neighborhood function, when restricted to $k = 2$, gives the 4-adjacency, and 8-adjacency when $k = 3$.

Let $I : [0,n)^k \to \{0,1\}$ be a k-D binary image of size n^k, where the set of points in the object (or black points) is $I^{-1}(1)$. Let $K = K(I)$ be the cubic cell complex built from I. In K, the 0-cells represent points in the object, the 1-cells represent pairs of $2k$-adjacent points, the 2-cells represent unit squares where its edges are pairs of $2k$-adjacent points, and so on. In general, each p-cell is a p-dimensional unit hypercube whose edges are pairs of $2k$-adjacent points.

The cubical complex K built from an image I can be encoded using tuples in $[0,2n-1)^k$. The 0-cell $(i_1,\ldots,i_p)$ is encoded using the tuple $(2i_1,\ldots,2i_p)$. Higher dimension cells are encoded using tuples in $[0,2n-1)^k$ with many odd coordinates as the dimension of the cell. The way a p-cell is encoded using only one tuple is based in the idea of barycenter. Exactly, let σ be a p-cell with vertices given by $\mathbf{i}_1,\ldots,\mathbf{i}_{2p}$, and let us suppose that the vertices are sorted by lexicographic order. The set $\{\mathbf{v}_j = \mathbf{i}_j - \mathbf{i}_1 : 2 \leq j \leq 2p\}$ can be thought as a set of vectors in $\mathbb{R}^k$. From this set, we can extract a basis formed by vectors from the canonical one. Let $\{\mathbf{u}_1,\ldots,\mathbf{u}_p\}$ be that basis. In such situation, the cell σ is encoded by the tuple

$$2\mathbf{i}_1 + \sum_{j=1}^{p} \mathbf{u}_p$$

As the vectors $\mathbf{u}_j$ have all the coordinates 0, except one of them with value 1, and all of them are linearly independent, the dimension of cell σ is the number of odd coordinates in its encoded tuple, as we have said before.

The operator $\partial : [0, 2n-1)^k \to 2^{[0,2n-1)^k}$ given by

$$\partial(i_1, \ldots, i_k) = \\ = \left\{ (j_1, \ldots, j_k) \in [0, 2n-1)^k : j_l = \begin{cases} i_l \pm 1 \text{ if } i_l \equiv 1 (mod\ 2) \wedge l = p \\ i_l \qquad \text{in other case} \end{cases} ; 1 \leq p \leq k \right\}$$

gives all the possible cells in the border of the one represented by $(i_1, \ldots, i_k)$. When we would like to find the border cells for one in a complex K, we may use the *restricted border operator* given by

$$\partial|_K \, \mathbf{i} = \partial \, \mathbf{i} \cap K$$

In the definition of the rules for the family of tissue-like P systems which solves the proposed skeletonization problem, the use of the *inverse border operator* will be useful. It is defined as follows.

$$\partial^{-1}(i_1, \ldots, i_k) = \\ = \left\{ (j_1, \ldots, j_k) \in [0, 2n)^k : j_l = \begin{cases} i_l \pm 1 \text{ if } i_l \equiv 0 (mod\ 2) \wedge l = p \\ i_l \qquad \text{in other case} \end{cases} ; 1 \leq p \leq k \right\}$$

There is no difficult in observing that, for any $\mathbf{j} \in \partial \mathbf{i}$ is $\mathbf{i} \in \partial^{-1} \mathbf{j}$. So the use of the name "inverse border operator" is plenty justified.

The tissue-like P systems presented in this paper have six membranes. The first membrane is used as input and for marking the isolated cells before starting the thinning iterations. The second membrane is used to mark simple pairs. The third membrane selects the cells to be removed. The fourth membrane marks new isolated cells and update the counter I. The fifth membrane updates counter R and the sixth one is used as output membrane. Next, the P system is formally described.

Let I be a k-D binary image of size n^k, let K be the cubical cell built from I, let $\varepsilon_{abs} \in \{1, 2, \ldots, n\}$ and $\varepsilon_{rel} \in \{\tau_1, \ldots, \tau_m\} \subset (0, 1) \cap \mathbb{Q}$, where $\tau_j < \tau_{j+1}$ for $1 \leq j < m$. For every tuple $\langle n, \varepsilon_{abs}, \varepsilon_{rel} \rangle$ we will define a tissue-like P system with promoters, inhibitors, priorities and input, denoted by $\Pi(n, \varepsilon_{abs}, \varepsilon_{rel})$ and defined as follows:

$$\Pi(n, \varepsilon_{abs}, \varepsilon_{rel}) = (\Gamma, \Sigma, \mathcal{E}, w_1, \ldots, w_6, \mathcal{R}, Pri, i_{in}, i_o)$$

where:

- $\Gamma = \{\mathbf{i} : \mathbf{i} \in [0, 2n-1)^k\} \cup \{(I, \mathbf{i}) : \mathbf{i} \in [0, 2n-1)^k\} \cup$
  $\{(R, \mathbf{i}, d) : \mathbf{i} \in [0, 2n-1)^k, 1 \leq d \leq n\} \cup \{(I, \mathbf{i}, D) : \mathbf{i} \in [0, 2n-1)^k, 0 \leq D \leq n\} \cup$
  $\{S_{\mathbf{i}} : \mathbf{i} \in [0, 2n-1)^k\} \cup \{U_{\mathbf{i}} : \mathbf{i} \in [0, 2n-1)^k\} \cup \{R\}$
- $\Sigma = \{\mathbf{i} \in [0, 2n-1)^k : \mathbf{i} \in K\}$
- $w_1 = \{(R, \mathbf{i}, 1) : \mathbf{i} \in K\} \cup \{(I, \mathbf{i}, 0) : \mathbf{i} \in K\}$
- $w_2 = \ldots = w_6 = \emptyset$
- $\mathcal{E} = \Gamma \setminus \Sigma$
- $\mathcal{R}$ is the set of rules:

- $R_1 \equiv \big(\{\mathbf{i}\}\, \neg\partial^{-1}\mathbf{i}|1, \lambda/(I, \mathbf{i}), 0\big)$
  for $\mathbf{i} \in [0, 2n-1)^k$
  These rules mark isolated cells before starting thinning iterations.
- $R_2 \equiv (1, \mathbf{i}\,(R, \mathbf{i}, 1)\,(I, \mathbf{i}, 0)/\lambda, 2)$
  for $\mathbf{i} \in [0, 2n-1)^k$
- $R_3 \equiv (1, (I, \mathbf{i})/\lambda, 2)$
  for $\mathbf{i} \in [0, 2n-1)^k$
  These rules move objects to the second membrane for starting the thinning iterations.
- $R_4 \equiv \big(\{\mathbf{i}, \mathbf{j}\}\, \neg\left(\partial^{-1}\mathbf{j} \setminus \{\mathbf{i}\} \cup \{S_\mathbf{i}, S_\mathbf{j}\}\right)|2, \lambda/S_\mathbf{i}\, S_\mathbf{j}, 0\big)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $\mathbf{j} \in \partial\mathbf{i}$.
  These rules mark simple pairs.
- $R_5 \equiv (2, \mathbf{i}\,(R, \mathbf{i}, d)\,(I, \mathbf{i}, D)/\lambda, 3)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $0 \leq d, D \leq n$.
- $R_6 \equiv (2, (I, \mathbf{i})/\lambda, 3)$
  for $\mathbf{i} \in [0, 2n-1)^k$.
- $R_7 \equiv (2, S_\mathbf{i}\, S_\mathbf{j}/\lambda, 3)$
  for $\mathbf{i}, \mathbf{j} \in [0, 2n-1)^k$.
  These rules move objects to the third membrane for marking cells to be removed.
- $R_8 \equiv (\{S_\mathbf{i}, S_\mathbf{j}, (R, \mathbf{i}, d), (I, \mathbf{i}, D)\}\, \neg\{R_\mathbf{i}, R_\mathbf{j}\}|3, \lambda/RR_\mathbf{i}\, R_\mathbf{j}, 0)$
  for $\mathbf{i} \in [0, 2n-1)^k$, $\mathbf{j} \in \partial\mathbf{i}$,
  $0 \leq d, D \leq n$, $d \neq 0$,
  $d - D < \varepsilon_{abs}$ and $1 - \frac{D}{d} < \varepsilon_{rel}$
  These rules will mark for removal only those simple pairs whose higher dimension cell has not enough shape signification. Shape signification is calculated using medial persistence measures from [24, 25] . A cell is significant enough if both medial persistence measures are greater than some thresholds, given by ε_{abs} and ε_{rel} for MP_{abs} and MP_{rel}, respectively.
- $R_9 \equiv (\{R_\mathbf{i}\}|3, \mathbf{i}\,(R, \mathbf{i}, d)\,(I, \mathbf{i}, D)/\lambda, 0)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $0 \leq d, D \leq n$.
- $R_{10} \equiv (\{R_\mathbf{i}\}|3, (I, \mathbf{i})/\lambda, 0)$
  for $\mathbf{i} \in [0, 2n-1)^k$.
- $R_{11} \equiv (\{R_\mathbf{i}, R_\mathbf{j}\}|3, S\, S_\mathbf{i}\, S_\mathbf{j}/\lambda, 0)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $\mathbf{j} \in \partial\mathbf{i}$
  These rules remove those simple pairs which are not significant enough.
- $R_{12} \equiv (\neg\{R_\mathbf{i}\}|3, \mathbf{i}\,(R, \mathbf{i}, d)\,(I, \mathbf{i}, D)/\lambda, 4)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $0 \leq d, D \leq n$.
- $R_{13} \equiv (\neg\{R_\mathbf{i}\}|3, (I, \mathbf{i})/\lambda, 4)$
  for $\mathbf{i} \in [0, 2n-1)^k$.
- $R_{14} \equiv (\neg\{R_\mathbf{i}, R_\mathbf{j}\}|3, S_\mathbf{i}\, S_\mathbf{j}/\lambda, 4)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $\mathbf{j} \in \partial\mathbf{i}$
  These rules send objects to the fourth membrane for marking new isolated cells.

- $R_{15} \equiv \big(\{\mathbf{i}, (R, \mathbf{i}, d)\}\, \neg(\partial^{-1}\mathbf{i} \cup \{(I, \mathbf{i})\}|4, (I, \mathbf{i}, D)/(I, \mathbf{i})\, (I, \mathbf{i}, d), 0\big)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $0 \leq d, D \leq n$.
  These rules mark new isolated cells and update counter I.
- $R_{16} \equiv (4, \mathbf{i}\, (R, \mathbf{i}, d)\, (I, \mathbf{i}, D)/\lambda, 5)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $0 \leq d, D \leq n$.
- $R_{17} \equiv (4, (I, \mathbf{i})/\lambda, 5)$
  for $\mathbf{i} \in [0, 2n-1)^k$.
- $R_{18} \equiv (4, S\, S_{\mathbf{i}}\, S_{\mathbf{j}}/\lambda, 5)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $\mathbf{j} \in \partial\mathbf{i}$
  These rules send objects to the fifth membrane for updating counter R.
- $R_{19} \equiv (\{R\}\, \neg\{U_{\mathbf{i}}\}|5, (R, \mathbf{i}, d)/(R, \mathbf{i}, d+1)\, U_{\mathbf{i}}, 0)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $1 \leq d \leq n$
  These rules update counter R.
- $R_{20} \equiv (\{U_{\mathbf{i}}\}|5, \mathbf{i}\, (R, \mathbf{i}, d)\, (I, \mathbf{i}, D)/\lambda, 2)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $0 \leq d, D \leq n$.
- $R_{21} \equiv (\{U_{\mathbf{i}}\}|5, (I, \mathbf{i})/\lambda, 2)$
  for $\mathbf{i} \in [0, 2n-1)^k$.
  These rules move objects back to the second membrane for starting the next thinning iteration.
- $R_{22} \equiv (\{U_{\mathbf{i}}, U_{\mathbf{j}}\}|5, R\, S_{\mathbf{i}}\, S_{\mathbf{j}}/\lambda, 0)$
  for $\mathbf{i} \in [0, 2n-1)^k$ and $\mathbf{j} \in \partial\mathbf{i}$.
- $R_{23} \equiv (5, U_{\mathbf{i}}/\lambda, 0)$
  for $\mathbf{i} \in [0, 2n-1)^k$.
  These rules remove unnecessary objects.
- $R_{24} \equiv (\neg\{R\}|5, \mathbf{i}/\lambda, 6)$
  for $\mathbf{i} \in [0, 2n-1)^k$.
  These rules send the skeletonized cell complex to the output membrane, when no cell has been removed in prior steps.

- $Pri = \{R_1 > R_p : p = 2, 3\} \cup \{R_4 > R_p : 5 \leq p \leq 7\} \cup$
  $\{R_{15} > R_p : 16 \leq p \leq 18\} \cup \{R_p > R_{23} : 19 \leq p \leq 22\} \cup$
  $\{R_8 > R_p : 12 \leq p \leq 14\}$
- $i_{in} = 1$ is the input cell.
- $i_{out} = 6$ is the output cell.

6 Overview of the Computation

Let $K \subset [0, 2n-1)^k$ be a cubical cell complex encoded as described above. Next, we will describe the behaviour of the P systems in the family Π when the input is set to K with thresholds set to ε_{abs} and ε_{rel} respectively. From now, $\mathcal{C}_c$ will denote the c-th configuration for the P system.

In order to make this overview more understandable, the process will be illustrated by the thinning process of image shown in figure 3.

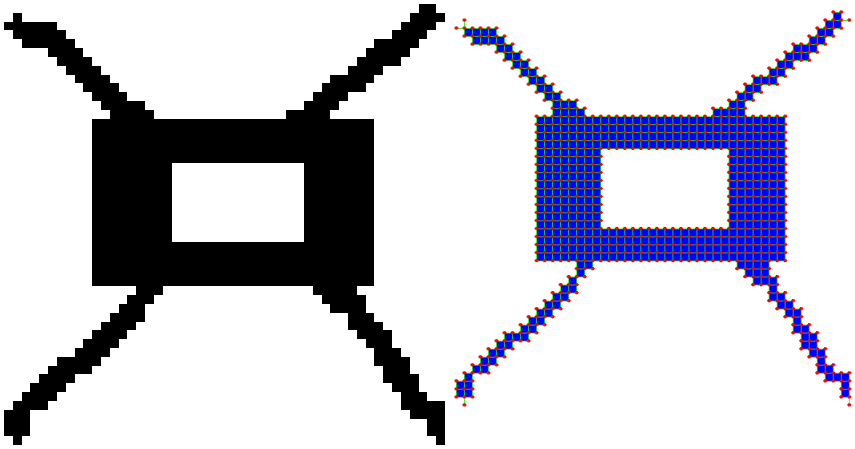


Fig. 3. Example image to show the thinning process, on the left. On the right is the cell complex representation for the image. Blue squares represent 2-cells, green lines represent 1-cells and red dots represent 0-cells.

In the initial state $\mathcal{C}_0$, the first membrane stores one object $\mathbf{i}$ for each cell in K. The initial values for counters R and I, given by objects $(R, \mathbf{i}, 1)$ and $(I, \mathbf{i}, 0)$, are also stored in the first membrane. In this situation, only rules R_1, R_2 or R_3 can be applied. For priority reasons, the rules R_1 are the only one that can be selected. After apply the selected rules from R_1, in $\mathcal{C}_1$, the first membrane contains objects $\mathbf{i}$ (for cells in K), counters R and I, and *isolation marks* $(I, \mathbf{i})$ for each isolated cell $\mathbf{i}$.

In the configuration $\mathcal{C}_1$, only the rules R_2 and R_3 can be selected, moving the cell objects $\mathbf{i}$, along with the isolation marks $(I, \mathbf{i})$ and counters $(R, \mathbf{i}, 1)$ and $(I, \mathbf{i}, 0)$, to the second membrane. The application of these rules gives as result the next configuration, $\mathcal{C}_2$. In this situation, only the rules establishing communications with the second membrane can be selected. Hence, the P system must select rules from $\{R_4, R_5, R_6, R_7\}$. However, for priority reasons, only the rules R_4 can be selected and applied, arising to the next configuration, where simple pairs $\langle j, i \rangle$ are marked by the presence of objects $S_{\mathbf{j}}$ and $S_{\mathbf{i}}$ in the second membrane.

In the current configuration, $\mathcal{C}_3$, only rules R_5, R_6 and R_7 can be selected. The application of them gives as result the configuration $\mathcal{C}_4$, where objects have been moved from the second to the third membrane. In the third membrane the simple pairs are going to be examined in order to detect those to be marked for removal, when they were not significant enough. In this situation, only rules R_8 can be selected and their application arises to the next configuration, $\mathcal{C}_5$, where those simple pairs $\langle \mathbf{j}, \mathbf{i} \rangle$ that can be removed are marked by $R_{\mathbf{j}}$ and $R_{\mathbf{i}}$.

In the previous configuration, only rules R_p, for $9 \leq p \leq 14$, can be selected. The application of these rules makes the P system evolve to the configuration $\mathcal{C}_6$,

where selected simple pairs have been removed, along with the auxiliary objects, and the remaining objects have been moved from the third to the fourth membrane.

In the configuration $\mathcal{C}_6$, for priority reasons again, only the rules R_{15} can be selected, and their application marks the new isolated cells and updates the counter $(I, \mathbf{i}, D)$. Now, all available objects are updated in the fourth membrane, in the configuration $\mathcal{C}_7$. Then, only rules R_{16}, R_{17} and R_{18} can be applied, resulting in the configuration $\mathcal{C}_8$ where all the objects in the fourth membrane are moved to the fifth one.

If no simple pairs have been marked for removal in configuration $\mathcal{C}_9$, there is no marker R in the fifth membrane. In this situation, the only rules that can be applied are those in R_{24}. The application of these rules leaves the P system in the configuration $\mathcal{C}_{10}$ which also is a halting configuration.

Let us Ssppose there have been some simple pairs marked for removal in configuration $\mathcal{C}_8$, which ensures the presence of marker R in the P system. Then, the application of rules in R_{19} updates the counter R, leaving the P system in the configuration C_9. In this situation, only rules in R_{20}, R_{21} and R_{22} can be applied. The former move objects to the second membrane, where the thinning iterations restart, the latter removes auxiliary objects from the fifth membrane. In this situation, the P system is in the configuration $\mathcal{C}_{10}$. The result for the example image is shown in Figure 4 (Right).

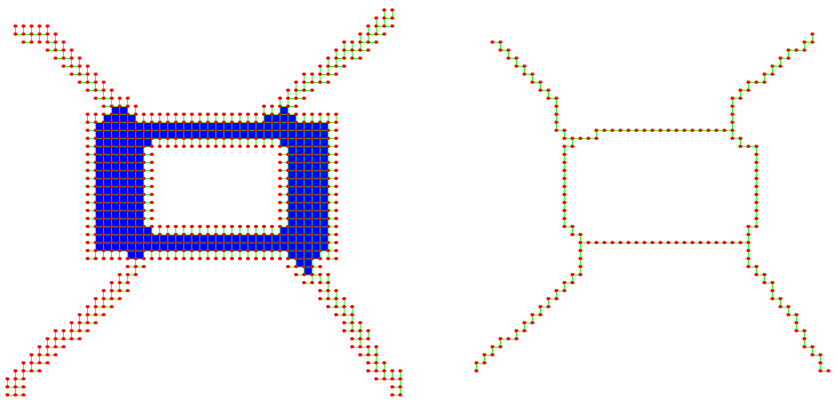


Fig. 4. (Left) Cell complex representation for cells in membrane 2 after the first thinning iteration. (Right) The thinned cell complex.

In previous situation, only the rules in R_4 and R_{23} can be applied. The former marks simple pairs in the second membrane, while the latter remove auxiliary remaining objects in the fifth membrane, leaving the P system in the configuration $\mathcal{C}_{11}$. From this point, the P system will evolve as above until it reaches the configuration $\mathcal{C}_{16}$ whether the halting condition may be reached in next configuration

$\mathcal{C}_{17}$, or not, depending on the presence of marker R. In the first case, the P system will start a new thinning iteration. In the second situation, the P system sends out the skeleton to the output membrane.

In any case, the P system will reach the halting configuration in $7t + 3$ steps, where t stands for the thinning iterations performed. If we start from a k-D binary image of size n^k where all the resels are black, and we do not pay attention to the shape significance, we perform a full thinning in a number of thinning iterations which, in addition, is the maximum. We have found that, in situation above, the greater number of thinning iterations is given by $k(n + 1)$. Hence, we can ensure that the P system halts in, at most, $7k(n + 1) + 3$ computation steps.

In Figure 4 (Right), the resulting image, representing the cell complex in the sixth membrane when the halting condition is reached, is shown.

The required computational resources for the family of tissue-like P systems defined in this paper is given in the table 1.

k-D binary image thinning problem	
Complexity	
Number of steps of computation	$\leq 7k(n+1)+3$
Resources needed	
Size of the alphabet	$O(n^{k+1})$
Initial number of cells	6
Initial number of objects	$3\|K\|$
Number of rules	$O(n^{k+2})$
Upper bound for the length of the rules	3

Table 1. Complexity aspects, where the size of the input data is $O(n^k)$, $|K|$ is the number of cells in the input cell complex K.

7 Conclusions and Future Work

In this paper, we bring together Membrane Computing and Cell Complexes. Both disciplines deal with compartments of the Euclidean space on their foundations, but their inspiration and motivation are quite different. The former is a computation model inspired in the functioning of living cells and tissues and the latter is born as a tool for handle concepts of Algebraic Topology.

In this paper, we use Membrane Computing techniques to implement a cell complex based algorithm for thinning images and show a new proof that the Membrane Computing framework is flexible enough to adapt to unexpected situations. In this way, this is a pioneer work that open a new research line that can be followed at different levels.

Firstly, we can study if other P system models (cell-like P systems, SN P systems, a most restrictive model of tissue-like P systems, ...) are *better* than

the one used in this paper to implement the Liu's algorithm in the Membrane Computing framework. *Better* should be considered here in a broad sense, since it can mean with a lower amount of resources, with less *ingredients* in the P system model o more efficient in some sense.

Another line to follow is to study if other problems in Algebraic Topology already studied with Cells Complexes can be considered in the framework of Membrane Computing. This research line can open a flow of inquiries and solutions in both directions enriching both disciplines with new points of view.

Finally, a more general question is the study of links on the foundations of Membrane Computing and Cell Complexes. As pointed out above, both disciplines shares a compartmental view of the Euclidean space and this can be a starting point for a deeper study of their common properties.

Acknowledgements

DDP and MAGN acknowledge the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. Adeoye, O.S.: A survey of emerging biometric technologies. International Journal of Computer Applications 9(10), 1–5 (November 2010)
2. Ayache, N.: Medical image analysis and simulation. In: Shyamasundar, R.K., Ueda, K. (eds.) ASIAN. Lecture Notes in Computer Science, vol. 1345, pp. 4–17. Springer (1997)
3. Blum, H.: An associative machine for dealing with the visual field and some of its biological implications. In: Bernard, E.E., Kare, M.R. (eds.) Biological Prototypes and Synthetic Systems. vol. 1, pp. 244–260. Plenum Press, New York (1962)
4. Carnero, J., Díaz-Pernil, D., Gutiérrez-Naranjo, M.A.: Designing tissue-like P systems for image segmentation on parallel architectures. In: del Amor, M.A.M., Păun, Gh., de Mendoza, I.P.H., Romero-Campero, F.J., Cabrera, L.V. (eds.) Ninth Brainstorming Week on Membrane Computing. pp. 43–62. Fénix Editora, Sevilla, Spain (2011)
5. Ceterchi, R., Gramatovici, R., Jonoska, N., Subramanian, K.G.: Tissue-like P systems with active membranes for picture generation. Fundamenta Informaticae 56(4), 311–328 (2003)
6. Ceterchi, R., Mutyam, M., Păun, Gh., Subramanian, K.G.: Array-rewriting P systems. Natural Computing 2(3), 229–249 (2003)
7. Christinal, H.A., Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Thresholding of 2D images with cell-like P systems. Romanian Journal of Information Science and Technology 13(2), 131–140 (2010)

8. Christinal, H.A., Díaz-Pernil, D., Real, P.: Segmentation in 2D and 3D image using tissue-like P system. In: Bayro-Corrochano, E., Eklundh, J.O. (eds.) Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. Lecture Notes in Computer Science, vol. 5856, pp. 169–176. Springer (2009)
9. Christinal, H.A., Díaz-Pernil, D., Real, P.: P systems and computational algebraic topology. Mathematical and Computer Modelling 52(11-12), 1982 – 1996 (2010)
10. Christinal, H.A., Díaz-Pernil, D., Real, P.: Region-based segmentation of 2D and 3D images with tissue-like P systems. Pattern Recognition Letters 32(16), 2206 – 2212 (2011)
11. Collins, R., Lipton, A., Kanade, T.: Introduction to the special section on video surveillance. Pattern Analysis and Machine Intelligence, IEEE Transactions on 22(8), 745 –746 (2000)
12. Dersanambika, K.S., Krithivasan, K.: Contextual array P systems. International Journal of Computer Mathematics 81(8), 955–969 (2004)
13. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Molina-Abril, H., Real, P.: A bio-inspired software for segmenting digital images. In: Nagar, A.K., Thamburaj, R., Li, K., Tang, Z., Li, R. (eds.) Proceedings of the 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications BIC-TA. vol. 2, pp. 1377 – 1381. IEEE Computer Society, Beijing, China (2010)
14. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Molina-Abril, H., Real, P.: Designing a new software tool for digital imagery based on P systems. Natural Computing pp. 1–6 (2011)
15. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A linear-time tissue P system based solution for the 3-coloring problem. Electronic Notes in Theoretical Computer Science 171(2), 81–93 (2007)
16. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Solving subset sum in linear time by using tissue P systems with cell division. In: Mira, J., Álvarez, J.R. (eds.) IWINAC (1). Lecture Notes in Computer Science, vol. 4527, pp. 170–179. Springer (2007)
17. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Real, P., Sánchez-Canales, V.: Computing homology groups in binary 2D imagery by tissue-like P systems. Romanian Journal of Information Science and Technology 13(2), 141–152 (2010)
18. Egmont-Petersen, M., de Ridder, D., Handels, H.: Image processing with neural networks - a review. Pattern Recognition 35(10), 2279–2301 (2002)
19. Ferretti, C., Mauri, G., Zandron, C.: P systems with string objects. In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 168 – 197. Oxford University Press, Oxford, England (2010)
20. Gimel'farb, G., Nicolescu, R., Ragavan, S.: P systems in stereo matching. In: Real, P., Díaz-Pernil, D., Molina-Abril, H., Berciano, A., Kropatsch, W. (eds.) Computer Analysis of Images and Patterns, Lecture Notes in Computer Science, vol. 6855, pp. 285–292. Springer (2011)
21. Gimel'farb, G.L.: Probabilistic regularisation and symmetry in binocular dynamic programming stereo. Pattern Recognition Letters 23(4), 431–442 (2002)
22. Kaczyński, T., Mischaikow, K., Mrozek, M.: Computational homology. Applied mathematical sciences, Springer (2004)
23. Krishna, S.N., Rama, R., Krithivasan, K.: P systems with picture objects. Acta Cybernetica 15(1), 53–74 (2001)
24. Liu, L.: 3D thinning on cell complexes for computing curve and surface skeletons. Washington University (2009)

25. Liu, L., Chambers, E.W., Letscher, D., Ju, T.: A simple and robust thinning algorithm on cell complexes. Computer Graphics Forum 29(7), 2253–2260 (2010)
26. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: Tissue P systems. In: Ibarra, O.H., Zhang, L. (eds.) COCOON. Lecture Notes in Computer Science, vol. 2387, pp. 290–299. Springer (2002)
27. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science 296(2), 295–326 (2003)
28. Peña-Cantillana, F., Díaz-Pernil, D., Berciano, A., Gutiérrez-Naranjo, M.A.: A parallel implementation of the thresholding problem by using tissue-like P systems. In: Real, P., Díaz-Pernil, D., Molina-Abril, H., Berciano, A., Kropatsch, W.G. (eds.) CAIP (2). Lecture Notes in Computer Science, vol. 6855, pp. 277–284. Springer (2011)
29. Peña-Cantillana, F., Díaz-Pernil, D., Christinal, H.A., Gutiérrez-Naranjo, M.A.: Implementation on CUDA of the smoothing problem with tissue-like P systems. International Journal of Natural Computing Research 2(3), 25–34 (2011)
30. Păun, A., Păun, Gh.: The power of communication: P systems with symport/antiport. New Generation Computing 20(3), 295–306 (2002)
31. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
32. Ritter, G.X., Wilson, J.N., Davidson, J.L.: Image algebra: An overview. Computer Vision, Graphics, and Image Processing 49(3), 297–331 (1990)
33. Rosin, P.L.: Training cellular automata for image processing. IEEE Transactions on Image Processing 15(7), 2076–2087 (2006)
34. Saeed, K., Tabedzki, M., Rybnik, M., Adamski, M.: K3M: A universal algorithm for image skeletonization and a review of thinning techniques. Applied Mathematics and Computer Science 20(2), 317–335 (2010)
35. Selvapeter, P.J., Hordijk, W.: Cellular automata for image noise filtering. In: NaBIC. pp. 193–197. IEEE (2009)
36. Shapiro, L.G., Stockman, G.C.: Computer Vision. Prentice Hall PTR, Upper Saddle River, NJ, USA (2001)
37. Zhou, Q.Y., Ju, T., Hu, S.M.: Topology repair of solid models using skeletons. IEEE Transactions on Visualization and Computer Graphics 13(4), 675–685 (2007)
38. Zhou, Y., Chellappa, R.: Artificial neural networks for computer vision. Research notes in neural computing, Springer-Verlag (1992)

Asynchronous Spiking Neural P Systems with Local Synchronization

Tao Song[1], Linqiang Pan[1], Gheorghe Păun[2]

[1] Key Laboratory of Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
lqpan@mail.hust.edu.cn

[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania, and
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es

Summary. Spiking neural P systems (SN P systems, for short) are a class of distributed parallel computing devices inspired from the way neurons communicate by means of spikes. Asynchronous SN P systems are non-synchronized systems, where the use of spiking rules (even if they are enabled by the contents of neurons) is not obligatory. In this paper, with a biological inspiration (in order to achieve some specific biological functioning, neurons from the same functioning motif or community work synchronously to cooperate with each other), we introduce the notion of local synchronization into asynchronous SN P systems. The computation power of asynchronous SN P systems with local synchronization is investigated. Such systems consisting of general neurons (resp. unbounded neurons) and using standard spiking rules are proved to be universal. Asynchronous SN P systems with local synchronization consisting of bounded neurons and using standard spiking rules characterize the semilinear sets of natural numbers. These results show that the local synchronization is useful, it provides some "programming capacity" useful for achieving a desired computational power.

1 Introduction

Membrane computing is one of the recent branches of natural computing. It was initiated in [9] and has developed rapidly (already in 2003, ISI considered membrane computing as a "fast emerging research area in computer science", see http://esi-topics.com). The aim is to abstract computing ideas (data structures, operations with data, ways to control operations, computing models, etc.) from the structure and the functioning of a single cell and from complexes of

cells, such as tissues and organs, including the brain. The obtained models are distributed and parallel computing devices, usually called *P systems*. There are three main classes of P systems investigated: cell-like P systems (based on a cell-like (hence hierarchical) arrangement of membranes delimiting compartments where multisets of chemicals evolve according to given evolution rules) [9], tissue-like P systems (instead of hierarchical arrangement of membranes, one considers arbitrary graphs as underlying structures, with membranes placed in the nodes, and with the edges corresponding to communication channels) [7], and neural-like P systems. Many variants of all these systems have been considered; an overview of the field can be found in [10] and [11], with up-to-date information available at the membrane computing website (`http://ppage.psystems.eu`). For an introduction to membrane computing, one may consult [10] and [11]. The present work deals with a class of neural-like P systems, called *spiking neural P systems* (SN P systems, for short), introduced in [5].

SN P systems are a class of distributed and parallel computing models inspired by spiking neurons. As we know, neurons are one of the most interesting cell-types in the human body. A large number of neurons working in a cooperative manner are able to perform tasks (such as thought, self-awareness, intuition) that are not yet matched by the tools we can build with our current technology. However, we believe that the distributed manner in which the brain processes information is important in obtaining better performance for electronic computers, that is why we are interested in SN P systems defined as a *computation model*. We stress that in this work SN P systems are a subject of a theoretical computer science investigation, without any intention to propose a platform for modeling biological processes.

Briefly, an SN P system consists of a set of *neurons* placed in the nodes of a directed graph, where neurons send signals (called *spikes* and denoted by the symbol a in what follows) along *synapses* (arcs of the graph). Spikes evolve by means of *standard spiking rules*, which are of the form $E/a^c \to a; d$, where E is a regular expression over $\{\mathrm{a}\}$ and c, d are natural numbers, $c \geq 1$, $d \geq 0$. The meaning is that if a neuron contains k spikes such that $a^k \in L(E)$ and $k \geq c$, then it can consume c spikes and produce one spike after a delay of d steps. This spike is sent to all neurons connected by an outgoing synapse starting in the neuron where the rule was applied. There are also *standard forgetting rules*, of the form $a^s \to \lambda$, with the meaning that $s \geq 1$ spikes are forgotten if the neuron contains exactly s spikes. *Extended rules* were considered in [3]: these rules are of the form $E/a^c \to a^p; d$, with the meaning that when using this rule, c spikes are consumed and p spikes are produced. Because p can be 0 or greater than 0, we obtain a generalization of both standard spiking and forgetting rules.

A rule is *bounded* if it is of the form $a^i/a^c \to a^p; d$, where $0 < c \leq i$, $0 < p \leq c$, $d \geq 0$. A neuron is *bounded* if it contains only bounded rules. A rule is called *unbounded* if it is of the form $a^i(a^j)^*/a^c \to a^p; d$, with $i \geq 0$, $j \geq 1$, $c \geq p > 0$, $d \geq 0$. (Note that also rules of the form $a^i(a^j)^+/a^c \to a^p; d$ are covered, as they can be rewritten in the form $a^{i+j}(a^j)^*/a^c \to a^p; d$.) A neuron is *unbounded* if it

contains only unbounded rules. A neuron is *general* if it contains both bounded and unbounded rules. An SN P system is *bounded* if all neurons in the system are bounded. It is *unbounded* if it has both bounded and unbounded neurons. An SN P system is *general* if it contains at least one general neuron (i.e., a neuron containing both bounded and unbounded rules).

An SN P system works in a synchronized manner. A global clock is assumed, and in each time unit, the rule to be applied in each neuron is non-deterministically chosen; one rule must be applied in each neuron with applicable rules. The work of the system is sequential in each neuron: only (at most) one rule is applied in each neuron. One of the neurons is considered to be the output one, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, and the other moments are marked with 0. This binary sequence is called the *spike train* of the system; it might be infinite if the computation does not stop. Various numbers can be associated with a spike train, which can be considered as computed (or generated) by an SN P system.

Synchronized SN P systems using standard rules were proved to be computationally complete both in the generating and the accepting case [5]. In the proof of these results, the synchronization plays a crucial role. However, both from a mathematical point of view and from a neuro-biological point of view, it is rather natural to consider non-synchronized systems, where the use of rules is not obligatory. Even if a neuron has a rule enabled in a given time unit, this rule is not obligatorily used. The neuron may remain unfired, maybe receiving spikes from the neighboring neurons. If the unused rule may be used later, it is used later, without any restriction on the interval when it has remained unused. If further spikes made the rule non-applicable, then the computation continues in the new circumstances (maybe other rules are enabled now). With this motivation, asynchronous SN P systems were introduced in [2]. It was proved that asynchronous general SN P systems with extended rules are equivalent with Turing machines; asynchronous unbounded SN P systems with extended rules are not universal. However, it remains open whether asynchronous general SN P systems with standard rules are universal.

In a biological neural system, motifs with 4-5 neurons and communities with 12-15 neurons, associated with some specific functioning, are rather common [1]. The neurons from the same motif or community will work synchronously to cooperate with each other. That is, in a biological neural system, neurons work asynchronously at the global level, but neurons from the same functioning motif or community work synchronously at the local level. With this biological inspiration, we introduce asynchronous SN P systems with *local synchronization*, where a family of sets of neurons (we call them *ls-sets*) is specified; if one of the neurons from an ls-set fires, then all neurons from this set should fire, provided that they have enabled rules. Of course, it is possible that all neurons from an ls-set remain unfired even if they have enabled rules, because of the global asynchronous mode.

In this work, we prove that asynchronous general or unbounded SN P systems with local synchronization using standard rules are universal; in the bounded case a characterization of semilinear sets of numbers is obtained.

In the asynchronous SN P systems constructed in [2], the non-determinism comes from two resources: (1) the asynchronous mode; (2) the non-deterministic choice of enabled rules to be applied. Each neuron of a system constructed in the present paper works in a deterministic way, in the sense that at each step each neuron has at most one enabled rule; however, the whole system works in a non-deterministic way because of the asynchronous mode.

In the proofs of the universality results in this work, the feature of local synchronization plays a crucial role. An asynchronous general SN P system with standard rules loses this "programming capacity" ensured by the extended rules and the local synchronization. So, our research gives some hint to support the conjecture that an asynchronous general SN P system with standard rules is non-universal [2].

2 Preliminaries

It is useful for readers to have some familiarity with basic elements of language theory, e.g., from [13], as well as basic membrane computing [10]. We here only introduce the necessary prerequisites.

The set of natural numbers is denoted by $\mathbb{N}$.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V; the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$.

A regular expression over an alphabet V is defined as follows: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. With each regular expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, (ii) $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = (L(E_1))^+$, for all regular expressions E_1, E_2 over V. Unnecessary parentheses can be omitted when writing a regular expression, and $(E)^+ \cup \{\lambda\}$ can also be written as E^*.

By $SLIN$, NRE we denote the families of semilinear and of Turing computable sets of numbers. ($SLIN$ is the family of length sets of regular languages – languages characterized by regular expressions; NRE is the family of length sets of recursively enumerable languages – those recognized by Turing machines.)

In the university proofs, the notion of a *register machine* is used. A register machine is a construct $M = (m, H, l_0, l_h, R)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to instruction `HALT`), and R is the set of instructions; each element of H labels

only one instruction from R, thus precisely identifying it. The instructions are of the following forms:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k),
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ (if register r is non-zero, then subtract 1 from it, and go to the instruction with label l_j; otherwise, go to the instruction with label l_k),
- $l_h : \texttt{HALT}$ (the halt instruction).

A register machine M computes (generates) a number n in the following way. The register machine starts with all registers empty (i.e., storing the number zero). It applies the instruction with label l_0 and proceeds to apply instructions as indicated by labels (and, in the case of SUB instructions, by the contents of registers). If the register machine reaches the halt instruction, then the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$. It is known that register machines compute all sets of numbers which are Turing computable, hence they characterize NRE [8].

Without loss of generality, it can be assumed that l_0 labels an ADD instruction, that in the halting configuration all registers different from the first one are empty, and that the output register is never decremented during the computation (its content is only added to).

A strongly monotonic register machine is a non-deterministic machine with only one register (this is also the output register). This register is initially zero and can only be incremented by 1 at each computation step (that is why we call such a machine strongly monotonic). When the machine halts, the value stored in the register is said to be generated. It is known that a set of natural numbers is semilinear if and only if it can be generated by a strongly monotonic register machine.

A register machine can also be used in the accepting mode: one starts with all registers empty, except one specified register, the input one, where a number x is introduced; the computation starts (with instruction with label l_0) and, if it halts, then the number x is accepted. In this way, again all sets of numbers from NRE are characterized. Furthermore, a register machine can be used for computing functions $\varphi : \mathbb{N} \longrightarrow \mathbb{N}$: certain registers are designated as input registers and a specific one as the output register; numbers $x_1, \ldots, x_k$ are introduced in the input registers and the value of a function $\varphi(x_1, \ldots, m_k)$ is obtained in the output register – providing that φ is defined for $x_1, x_2, \ldots, x_k$, otherwise the computation never halts. Turing computable functions can be computed in this way.

We use the following convention. When the power of two number generating/accepting devices D_1 and D_2 are compared, number zero is ignored, that is, $N(D_1) = N(D_2)$ if and only if $N(D_1) - \{0\} = N(D_2) - \{0\}$ (this corresponds to the usual practice of ignoring the empty string in language and automata theory).

3 Asynchronous Spiking Neural P Systems with Local Synchronization

In this section, we introduce the variant of SN P systems investigated in this work – asynchronous spiking neural P systems with local synchronization. The definition is complete, but familiarity with the basic elements of classic SN P systems (e.g., from [11], [12]) is helpful.

An *asynchronous spiking neural P system (without delay) with local synchronization* is a construct of the form:

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, Loc, syn, out), \text{ where}$$

- $m \geq 1$ is the *degree* of the system;
- $O = \{a\}$ is the singleton alphabet (a is called *spike*);
- $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons* of the form $\sigma_i = (n_i, R_i)$ with $1 \leq i \leq m$, where
  (1) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
  (2) R_i is a finite set of *extended rules* of the following form: $E/a^c \to a^p$, where E is a regular expression over O, $c \geq 1$ and $c \geq p \geq 0$;
- $Loc = \{loc_1, loc_2, \ldots, loc_l\} \subseteq \mathcal{P}(\{\sigma_1, \sigma_2, \ldots, \sigma_m\})$ is the *family of sets of locally synchronous neurons* (we call these sets ls-sets), where $\mathcal{P}(\{\sigma_1, \sigma_2, \ldots, \sigma_m\})$ is the power set of $\{\sigma_1, \sigma_2, \ldots, \sigma_m\}$; the number $\max\{|loc_1|, |loc_2|, \ldots, |loc_l|\}$ is the *local synchronization degree* of the system;
- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ is the set of *synapses* between neurons;
- $out \in \{1, 2, \ldots, m\}$ indicates the *output* neuron.

A rule $E/a^c \to a^p$ with $p \geq 1$ is called *extended firing* (we also say *spiking*) *rule*; a rule $E/a^c \to a^p$ with $p = 0$ is written in the form $E/a^c \to \lambda$ and is called a *forgetting rule*. If $L(E) = \{a^c\}$, then the rules are written in the simplified forms $a^c \to a^p$ and $a^c \to \lambda$. A rule of the type $E/a^c \to a$ and $a^c \to \lambda$ is said to be *standard*.

The rules are applied as follows. If neuron σ_i contains k spikes and $a^k \in L(E)$, $k \geq c$, then the rule $E/a^c \to a^p \in R_i$ is enabled and can be applied. This means that c spikes are consumed (thus $k - c$ spikes remain in neuron σ_i), the neuron is fired, and it produces p spikes. The p spikes emitted by a neuron σ_i are replicated and they go to all neurons σ_j such that $(i, j) \in syn$ (each neuron σ_j receives p spikes). Every neuron can contain several rules. Because two firing rules, $E_1/a^{c_1} \to a^{p_1}$ and $E_2/a^{c_2} \to a^{p_2}$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more spiking rules are enabled in a neuron at some moment, and then one of them is chosen non-deterministically.

If the rule is a forgetting one of the form $E/a^c \to \lambda$, then, when it is applied, $c \geq 1$ spikes are removed.

A global clock is assumed, marking the time for all neurons. In each time unit, any neuron is free to use a rule or not, i.e., a neuron can remain still in spite of the fact that it contains rules which are enabled by its contents. If the content of the

neuron is not changed, a rule which is enabled in a given step can fire later. If new spikes are received, then it is possible that other rules will be enabled and applied or not. Furthermore, for neurons in the same ls-set loc_j, if one of these neurons fires, then all neurons in loc_j that have enabled rules should fire. Of course, it is possible that all neurons from loc_j remain unfired even if they have enabled rules. That is, all neurons from loc_j may remain still, or all neurons from loc_j with enabled rules fire at a same step (of course, neurons without enabled rules cannot fire). Hence, neurons work asynchronously at the global level, but neurons in each ls-set work synchronously.

The "state" of the system at a given time is described by the number of spikes present in each neuron. That is, the *configuration* of the system is of the form $\langle r_1, r_2, \ldots, r_m \rangle$ for $r_i \geq 0$, which indicates that neuron σ_i contains r_i spikes. With this notation, the *initial configuration* of the system is $\langle n_1, n_2, \ldots, n_m \rangle$. By using the rules as described above, one can define *transitions* among configurations. Any series of transitions starting from the initial configuration is called a *computation*. A computation is *successful* if it reaches a configuration where no rule can be applied in any neuron (i.e., the SN P system has halted). The *result of a computation* is defined here as the total number of spikes sent into the environment by the output neuron. (Because of the asynchronous mode, now "the time does not matter". The output neuron can remain still for any number of steps between two consecutive spikes, therefore the result of a computation can no longer be defined in terms of the steps between two consecutive spikes as in the standard SN P systems definition.)

Specifically, a number x is generated by an SN P system if there is a halting computation of the system where the output neuron emits exactly x spikes (if several spikes are emitted at the same time, all of them are counted). Because of the non-determinism in using the rules, a given system computes in this way a set of numbers.

The rules, the neurons, and the SN P systems are called *bounded, unbounded*, or *general* as defined in the Introduction – the definitions are obvious, so we do not recall them here.

Asynchronous SN P systems with local synchronization can be used as computing devices in various ways, but here we consider them only as generators of numbers. We denote by $N(\Pi)$ the set of numbers generated by an asynchronous SN P system with local synchronization Π, and by $NSpik_{out}P_m^{locsyn}(\alpha, ls)$ with $\alpha \in \{gen, boun, unb\}$ and $ls \geq 0$, the family of such sets of numbers generated by systems of type α (*gen* stands for general, *boun* for bounded, *unb* for unbounded), with at most m neurons, and local synchronization degree at most ls. If one of the parameters m and ls is not bounded, then it is replaced with $*$. The subscript *out* reminds us of the fact that we count all spikes sent into the environment as computation results.

4 Asynchronous General SN P Systems with Local Synchronization

In this section, we investigate the computation power of asynchronous general SN P systems with local synchronization, and we prove that such systems using standard rules are universal. It was formulated as an open problem whether asynchronous SN P systems with standard rules are universal and it was conjectured that the answer is negative in [2]. So, the feature of local synchronization provides a useful "programming capacity".

As it is usual in the area of spiking neural P systems, asynchronous SN P systems with local synchronization are represented graphically, which may be easier to understand than in a symbolic way. We use an oval with rules inside to represent a neuron, and a directed graph to represent the structure of the system: the neurons are placed in the nodes of the graph and the edges represent the synapses; the output neuron has an outgoing arrow, suggesting its communication with the environment.

Theorem 1. $NRE = NSpik_{out}P_*^{locsyn}(gen, *)$.

We only have to prove that $NRE \subseteq NSpik_{out}P_*^{locsyn}(gen, *)$, since the converse inclusion is straightforward (or we can invoke for it the Turing-Church thesis). To this aim, we use the characterization of NRE by means of generating register machines. Let us consider a register machine $M = (m, H, l_0, l_h, I)$. As mentioned in Section 2, without any loss of generality, we may assume that in the halting configuration, all registers different from register 1 are empty, and that the output register is never decremented during a computation. For each register r of M, let s_r be the number of SUB instructions acting on register r. If there is no such SUB instruction, then $s_r = 0$, which is the case for the first register $r = 1$. In what follows, a specific asynchronous SN P system with local synchronization Π will be constructed to simulate the register machine M, where each neuron in system Π has only standard rules.

The system Π consists of three types of modules – ADD modules, SUB modules, and a FIN module. ADD modules and SUB modules are used to simulate the ADD and SUB instructions of M, respectively; the FIN module is used to output a computation result.

In general, a neuron σ_r is associated with each register r of M; the number stored in register r is encoded by the number of spikes in neuron σ_r. Specifically, if register r holds the number $n \geq 0$, then neuron σ_r contains $2n$ spikes. With each label l_i of an instruction in M, a neuron σ_{l_i} is associated. In the initial configuration, all neurons are empty, with the exception of neuron σ_{l_0} associated with the initial instruction l_0 of M, which contains one spike. During a computation, a neuron σ_{l_i} having one spike inside will become active and starts to simulate an instruction $l_i : (\mathtt{OP}(r), l_j, l_k)$ of M: starting with neuron σ_{l_i} activated, one changes neuron σ_r as requested by OP, then one introduces one spike into neuron σ_{l_j} or neuron σ_{l_k}, which becomes active in this way. When neuron σ_{l_h} (associated with the

label l_h of the halting instruction of M) is activated, a computation in M is completely simulated in Π; the FIN module starts to output the computation result (the number of spikes sent into the environment by the output neuron corresponds to the number stored in register 1 of M).

In what follows, the modules ADD, SUB, and FIN are given in the standard graphical way, also specifying their ls-sets, and their work is briefly analyzed.

Module ADD (shown in Figure 1) – simulating an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$.

The initial instruction of M, the one with label l_0, is an ADD instruction. Let us assume that at step t, an instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ has to be simulated, with one spike present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and no spike in any other neurons, except in those neurons associated with registers. Having one spike inside, the rule $a \to a$ is enabled, neuron σ_{l_i} can fire, and at some time it will do it (otherwise, the computation does not halt), sending one spike to neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$, respectively. Having one spike inside, neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(3)}}$ can fire; neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ will fire at the same step (because they are in the same ls-set $\{\sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}\}$) sending two spikes to neurons $\sigma_{l_i^{(3)}}$, $\sigma_{l_i^{(4)}}$ and σ_r, respectively. In this way, the number of spikes in neuron σ_r is increased by two, which corresponds to the fact that the number stored in register r is increased by one. For neuron $\sigma_{l_i^{(3)}}$, we have two possible cases.

Proof. (1) Neuron $\sigma_{l_i^{(3)}}$ fires before neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire. Note that, in this case, when neuron $\sigma_{l_i^{(3)}}$ fires, neuron $\sigma_{l_i^{(4)}}$ does not fire (it has no enabled rule), although neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ are in the same ls-set $\{\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}\}$. Neurons $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ receive one spike from neuron $\sigma_{l_i^{(3)}}$ and they keep inactive. After neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ receive two spikes from neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$, both of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ can fire, and they will fire at a same step, sending two spikes to neurons $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$. In this way, both of neurons $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ accumulate 3 spikes. With 3 spikes inside, neuron $\sigma_{l_i^{(5)}}$ can fire at any step by the rule $a^3 \to a$, and send one spike to neuron σ_{l_j}. After neuron σ_{l_j} receives one spike, it becomes active, starting to simulate the instruction l_j of M.

When neuron σ_{l_j} fires, it is possible that neuron $\sigma_{l_i^{(6)}}$ still contains three spikes because of non-synchronization. In this case, neuron $\sigma_{l_i^{(6)}}$ should also fire (because they are in the same ls-set) at the same step removing all three spikes, which insures that no spike remains in the module (except those in σ_r).

(2) Neuron $\sigma_{l_i^{(3)}}$ fires after neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire. In this case, neuron $\sigma_{l_i^{(3)}}$ accumulates three spikes and neuron $\sigma_{l_i^{(4)}}$ has two spikes. Neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ will fire at a same step. Similar to case (1), neuron σ_{l_k} will become active, starting to simulate the instruction l_k of M (at the same step "cleaning" neuron $\sigma_{l_i^{(5)}}$).

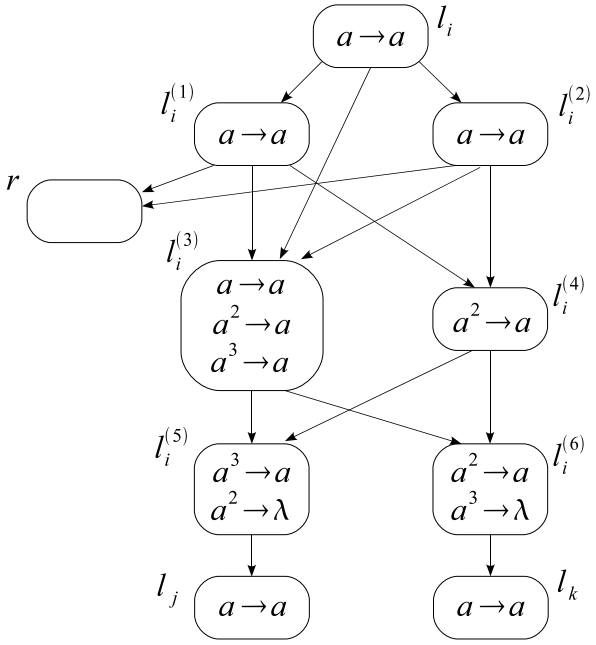


Fig. 1. Module ADD with four ls-sets $\{\sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}\}$, $\{\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}\}$, $\{\sigma_{l_i^{(5)}}, \sigma_{l_k}\}$ and $\{\sigma_{l_i^{(6)}}, \sigma_{l_j}\}$ for simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$

Therefore, after firing neuron σ_{l_i}, the system adds two spikes to neuron σ_r and non-deterministically fires one of neurons σ_{l_j} and σ_{l_k}, which correctly simulates the ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$.

Module SUB (shown in Figure 2) – simulating a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$.

A SUB instruction l_i is simulated in Π in the following way. Initially, neuron σ_{l_i} has one spike, and other neurons are empty, except the neurons associated with registers. With one spike inside, the rule $a \to a$ in neuron σ_{l_i} is enabled, and it will fire at some step sending one spike to neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and σ_r. These neurons will fire simultaneously, because they are in the same ls-set. For neuron σ_r, there are two cases.

(1) Before receiving one spike from neuron σ_{l_i}, neuron σ_r contains $2n$ $(n > 0)$ spikes (corresponding to the fact that the number stored in register r is n, and $n > 0$). In this case, neuron σ_r gets $2n + 1$ spikes and the rule $a(a^2)^+/a^3 \to a$ is enabled. So, when neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and σ_r fire at some step, they send three spikes to each of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$. In neuron σ_r, three spike are consumed, ending with $2n + 1 - 3 = 2(n - 1)$ spikes, which simulates the fact that the number stored in register r is decreased by one. With three spikes

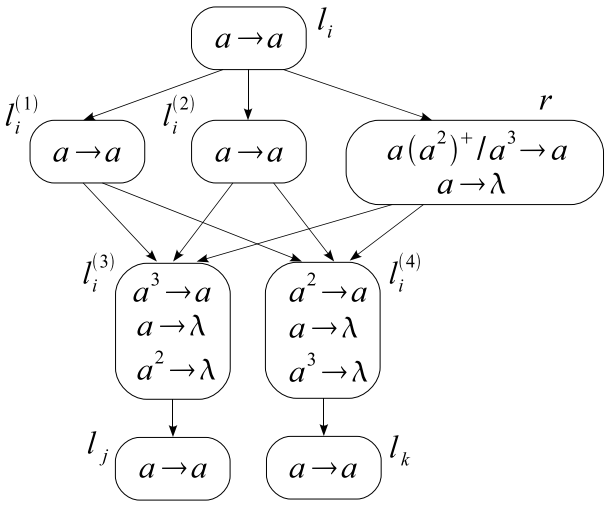


Fig. 2. Module SUB with three ls-sets $\{\sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}, \sigma_r\}$, $\{\sigma_{l_1^{(3)}}, \sigma_{l_2^{(3)}}, \ldots, \sigma_{l_{s_r}^{(3)}}, \sigma_{l_k}\}$ and $\{\sigma_{l_1^{(4)}}, \sigma_{l_2^{(4)}}, \ldots, \sigma_{l_{s_r}^{(4)}}, \sigma_{l_j}\}$ for simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$

inside, neuron $\sigma_{l_i^{(3)}}$ can fire, sending one spike to neuron σ_{l_j}, hence neuron σ_{l_j} will become active, and the system Π starts to simulate instruction l_j of M.

When neuron σ_{l_j} fires, it is possible that neuron $\sigma_{l_i^{(4)}}$ also contains three spikes because of non-synchronization. In this case, all neurons $\sigma_{l_s^{(4)}}$ should also fire (because they are in the same ls-set $\{\sigma_{l_1^{(4)}}, \sigma_{l_2^{(4)}}, \ldots, \sigma_{l_{s_r}^{(4)}}, \sigma_{l_j}\}$) removing simultaneously the three spikes, hence these neurons return to the initial state, with no spike inside.

(2) When receiving one spike from neuron σ_{l_i}, neuron σ_r has no spike inside (corresponding to the fact that the number stored in register r is 0). In this case, after neuron σ_r receives one spike from neuron σ_{l_i}, the rule $a \to \lambda$ in neuron σ_r is enabled. When neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and σ_r fire at some step, they send two spikes to each of neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$. In neuron σ_r, one spike is consumed, ending with 0 spikes, which means that the number stored in register r of M is zero. With two spikes inside, neuron $\sigma_{l_i^{(4)}}$ can fire, sending one spike to neuron σ_{l_k}, hence neuron σ_{l_k} will become active, and the system Π starts to simulate instruction l_k of M.

Similar to case (1), the ls-set $\{\sigma_{l_1^{(3)}}, \sigma_{l_2^{(3)}}, \ldots, \sigma_{l_{s_r}^{(3)}}, \sigma_{l_k}\}$ ensures that no "wrong" step is done in system Π.

The simulation of SUB instruction is correct: system Π starts from spiking neuron σ_{l_i} and ends in firing neuron σ_{l_j} (if the number stored in register r is greater than 0 and it was decreased by one), or in firing neuron σ_{l_k} (if the number stored in register r is 0).

Note that there is no interference between the ADD modules and the SUB modules, other than correctly firing the neurons σ_{l_j} or σ_{l_k}, which may label instructions of the other kind. However, it is possible to have interferences between two SUB modules. Specifically, if there are several SUB instructions l_v that act on the same register r, then neuron σ_r has synapses to all neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$. When a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is simulated, in the SUB module associated with l_v ($l_v \neq l_i$) all neurons receive no spike except for neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$, each of them having one spike inside. Because we have the ls-sets $\{\sigma_{l_1^{(3)}}, \sigma_{l_2^{(3)}}, \ldots, \sigma_{l_{s_r}^{(3)}}, \sigma_{l_k}\}$ and $\{\sigma_{l_1^{(4)}}, \sigma_{l_2^{(4)}}, \ldots, \sigma_{l_{s_r}^{(4)}}, \sigma_{l_j}\}$, when neuron $\sigma_{l_i^{(3)}}$ (resp. $\sigma_{l_i^{(4)}}$) fires, each of neurons $\sigma_{l_v^{(3)}}$ (resp. $\sigma_{l_v^{(4)}}$) ($l_v \neq l_i$) should also fire at the same step removing its spike. Consequently, the interference among SUB modules will not cause undesired steps in Π (i.e., steps that do not correspond to correct simulations of instructions of M).

Module FIN (shown in Figure 3) – outputting the result of computation.

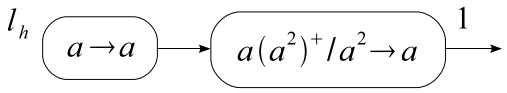


Fig. 3. The FIN module of Π

The functioning of this module is obvious: after activating the neuron σ_{l_h}, neuron σ_1 outputs one spike for each two spikes present inside; the last spike remains idle in the system.

From the above description of the modules and of their work, it is clear that the register machine M is correctly simulated by the system Π. Therefore, $N(\Pi) = N(M)$, and this completes the proof. □

In Theorem 1, the number m of neurons and the local synchronization degree ls are not bounded (thus, denoted by $*$). As expected, these parameters can be bounded by making use of the fact that there are (small) universal register machines. Such machines are given, e.g., in [6], but they are used in the accepting mode: the code of a particular register machine is introduced in register 1, an input for the particular machine is introduced in register 2, and the result is obtained in register 0. The universal machine halts if and only if the particular machine halts for the given input. The problem which remains to be solved is to pass from such an universal register machine to a generative SN P system.

Corollary 1. $NRE = NSpik_{out}P_{152}^{locsyn}(gen, 5)$.

Consider the universal register machine M_u from [6] as shown in Figure 4. It takes the code $code(M)$ of a particular register machine M which computes a function φ in register 1 and a number x in register 2, and outputs the value of $\varphi(x)$ in register 0.

Take an arbitrary set K in NRE and consider its membership function $\varphi_K : \mathbb{N} \longrightarrow \{0,1\}$. There is a register machine M_K computing this function. Starting with a number x in its input register, M_K halts if and only if $x \in K$, hence $\varphi_K(x) = 1$. Let $code(M_K)$ be its code; we introduce $code(M_K)$ in register 1 of M_u, thus obtaining a register machine $M_u(K)$. It takes any natural number n (in register 2) and halts (with 1 in register 0) if and only if $n \in K$. We modify $M_u(K)$ as follows. A further register is added, labeled with 8; consider two further labels, l_{-1}, l_{-2}, with l_{-1} being the initial label of the machine we want to obtain. Consider also the instructions $l_{-1} : (\texttt{ADD}(2), l_{-2}, l_{-2})$, $l_{-2} : (\texttt{ADD}(8), l_{-1}, l_0)$, where l_0 is the start label of M_u. The register machine M' obtained in this way has 9 registers, 11 ADD and 13 SUB instructions, and 25 labels; the result of a computation is stored in register 8, which is never decremented during a computation. Clearly, $n \in N(M')$ if and only if $\varphi_K(n) = 1$, hence $n \in K$, therefore $N(M') = K$: M' first "proposes" a number x to machine $M_u(K)$, by introducing it both in register 2, as needed for M_u and in register 8, the output one of M'; when the computation of $M_u(K)$ halts, also the computation of M' halts. Therefore, the number x is accepted and the computation of M' halts if and only if $x \in K$.

$l_0 : (\texttt{SUB}(1), l_1, l_2)$, $l_1 : (\texttt{ADD}(7), l_0)$,
$l_2 : (\texttt{ADD}(6), l_3)$, $l_3 : (\texttt{SUB}(5), l_2, l_4)$,
$l_4 : (\texttt{SUB}(6), l_5, l_3)$, $l_5 : (\texttt{ADD}(5), l_6)$,
$l_6 : (\texttt{SUB}(7), l_7, l_8)$, $l_7 : (\texttt{ADD}(1), l_4)$,
$l_8 : (\texttt{SUB}(6), l_9, l_0)$, $l_9 : (\texttt{ADD}(6), l_{10})$,
$l_{10} : (\texttt{SUB}(4), l_0, l_{11})$, $l_{11} : (\texttt{SUB}(5), l_{12}, l_{13})$,
$l_{12} : (\texttt{SUB}(5), l_{14}, l_{15})$, $l_{13} : (\texttt{SUB}(2), l_{18}, l_{19})$,
$l_{14} : (\texttt{SUB}(5), l_{16}, l_{17})$, $l_{15} : (\texttt{SUB}(3), l_{18}, l_{20})$,
$l_{16} : (\texttt{ADD}(4), l_{11})$, $l_{17} : (\texttt{ADD}(2), l_{21})$,
$l_{18} : (\texttt{SUB}(4), l_0, l_h)$, $l_{19} : (\texttt{SUB}(0), l_0, l_{18})$,
$l_{20} : (\texttt{ADD}(0), l_0)$, $l_{21} : (\texttt{ADD}(3), l_{18})$,
$l_h : \texttt{HALT}$

Fig. 4. A universal register machine M_u from Korec [6]

As in the proof of Theorem 1, we can construct an asynchronous SN P system with local synchronization $\Pi_{M'}$ to simulate the register machine M'. The system $\Pi_{M'}$ has

Proof.
- 9 neurons for the 9 registers,
- 25 neurons for the 25 labels,

- 6×11 neurons for the 11 ADD instructions,
- 4×13 neurons for the 13 SUB instructions,

which gives a total of 152 neurons.

The maximal size of ls-sets in the ADD module shown in Figure 1 is 2; the maximal size of ls-sets in the SUB module shown in Figure 2 is $\max\{3,\ s_r + 1\}$, where s_r is the number of SUB instructions acting on register r; the maximal size of ls-sets in the FIN module shown in Figure 3 is 0. So, the local synchronization degree is not more than $\max\{3, s_r + 1\}$. We can check that register 5 in M'_u has 4 SUB instructions, which is maximal, hence $\max\{3, s_r + 1\} = 5$. Therefore, our corollary holds. □

The above way of passing from computing universal register machines to register machines used in the generative way can be used in all results in membrane computing which show the computational completeness of a class of P systems by means of proofs based on the simulation of register machines. In this way, bounds on many parameters of the resulting P systems can be obtained. Note that many such bounds were reported in membrane computing (obtained by directly simulating a generating register machine), but no such result was reported for the number of neurons in SN P systems which characterize NRE. For instance, all universality results given in Chapter 13 of [11] are stated for SN P systems with arbitrarily many neurons. All these results can be improved from this point of view by using the technique from the proof of the previous corollary. Actually, also the number of rules in an SN P system should be considered (neurons can be saved by putting in the same neuron several rules), hence a double hierarchy should be discussed: number of neurons and number of rules.

5 Asynchronous Unbounded SN P System with Local Synchronization

In [2] it was proved that asynchronous unbounded SN P systems with extended rules can only characterize $SLIN$. In this section, making use of the "programming capability" of local synchronization, we prove that asynchronous unbounded SN P systems with local synchronization can achieve the Turing completeness.

Theorem 2. $NRE = NSpik_{out}P_*^{locsyn}(unb, *)$.

We only have to prove that $NRE \subseteq NSpik_{out}P_*^{locsyn}(unb, *)$, since the converse inclusion is straightforward (or we can invoke for it the Turing-Church thesis). As in the proof of Theorem 1, let us consider a register machine $M = (m, H, l_0, l_h, I)$, with the properties specified in Section 2. For each register r of M, let s_r be the number of instructions of the form $l_i : (\texttt{SUB}(r), l_j, l_k)$.

As in the proof of Theorem 1, we construct an SN P system Π' consististing of three types of modules – ADD, SUB, and FIN. The first and the third types of modules can be easily obtained by modifying the ADD and FIN modules from Figures 1 and 3, respectively, while the SUB module is given in Figure 5.

Specifically, we observe that in the ADD and FIN modules from the proof of Theorem 3, all bounded neurons never contain more than three spikes. Then, each bounded rule $a^c \to a$, $a^c \to \lambda$ can be replaced with the unbounded rule $a^c(a^4)^*/a^c \to a$, $a^c(a^4)^*/a^c \to \lambda$, respectively, and the functioning of modules ADD and FIN is not changed.

The difficulty with the module SUB from Figure 2 is that the neurons σ_r contain both unbounded rules and the bounded rule $a \to \lambda$, which we do not see how can be turned to an unbounded rule. That is why we present a different module SUB, more complex than that from Figure 2.

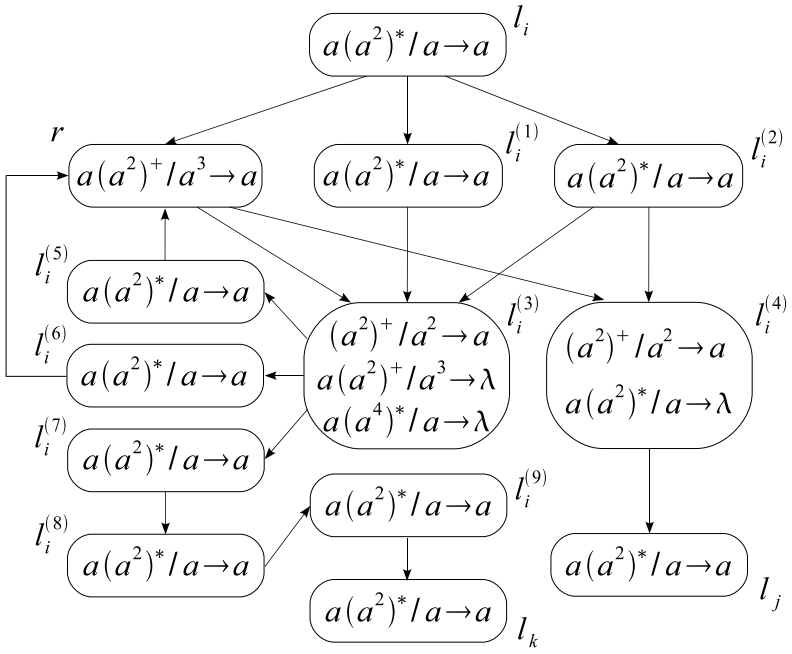


Fig. 5. The SUB module of Π' with ls-sets $\{\sigma_r, \sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}, \sigma_{l_i^{(8)}}\}$, $\{\sigma_{l_i^{(5)}}, \sigma_{l_i^{(6)}}, \sigma_{l_i^{(7)}}\}$, $\{\sigma_{l_1^{(3)}}, \sigma_{l_2^{(3)}}, \ldots, \sigma_{l_{s_r}^{(3)}}, \sigma_{l_1^{(4)}}, \sigma_{l_2^{(4)}}, \ldots, \sigma_{l_{s_r}^{(4)}}, \sigma_{l_1^{(9)}}, \sigma_{l_2^{(9)}}, \ldots, \sigma_{l_{s_r}^{(9)}}\}$

It is given in Figure 5 and it works as follows.

The simulation of a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is started with neuron σ_{l_i} having one spike. The rule $a(a^2)^*/a \to a$ is enabled and neuron σ_{l_i} fires at some step, sending a spike to neurons $\sigma_{l_i^{(1)}}, \sigma_{l_i^{(2)}}$ and σ_r, respectively. For neuron σ_r, there are two cases.

Proof. (1) Neuron σ_r has $2n$ ($n > 0$) spikes (corresponding to the fact that the number stored in register r is n, and $n > 0$) before it receives one spike from neuron σ_{l_i}. In this case, neuron σ_r gets $2n+1$ spikes and the rule $a(a^2)^+/a^3 \to a$ is enabled. When neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and σ_r fire at some step, they send three

spikes to neuron $\sigma_{l_i^{(3)}}$, as well as two spikes to neuron $\sigma_{l_i^{(4)}}$. (Note that neuron $\sigma_{l_i^{(8)}}$ is in the same ls-set with neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and σ_r, but it has no spike inside, so it cannot fire.) In neuron σ_r, three spikes are consumed, ending with $2n+1-3 = 2(n-1)$ spikes, which simulates the fact that the number stored in register r is decreased by one. Neuron $\sigma_{l_i^{(3)}}$ removes the three spikes inside by the forgetting rule $a(a^2)^+/a^3 \to \lambda$, meanwhile neuron $\sigma_{l_i^{(4)}}$ fires by the rule $(a^2)^+/a^2 \to a$, sending one spike to neuron σ_{l_j}, hence neuron σ_{l_j} will become active, and the system Π starts to simulate instruction l_j of M.

(2) Neuron σ_r has no spike inside (corresponding to the fact that the number stored in register r is 0) before it receives one spike from neuron σ_{l_i}. In this case, after neuron σ_r receives one spike from neuron σ_{l_i}, it keeps inactive for no rule is enabled. Neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ fire at some step, sending two spikes to neuron $\sigma_{l_i^{(3)}}$, as well as one spike to neuron $\sigma_{l_i^{(4)}}$. With two spikes inside, neuron $\sigma_{l_i^{(3)}}$ can fire at some step, sending one spike to neurons $\sigma_{l_i^{(5)}}, \sigma_{l_i^{(6)}}$, and $\sigma_{l_i^{(7)}}$, respectively. At the same moment, neuron $\sigma_{l_i^{(4)}}$ removes the spike inside by the forgetting rule $a(a^2)^*/a \to \lambda$. Neurons $\sigma_{l_i^{(5)}}, \sigma_{l_i^{(6)}}$, and $\sigma_{l_i^{(7)}}$ fire at some step sending two spikes to neuron σ_r and one spike to neuron $\sigma_{l_i^{(8)}}$. After neuron σ_r receives the two spikes, it has three spikes inside and the rule $a(a^2)^+/a^3 \to a$ is enabled. Neurons σ_r and $\sigma_{l_i^{(8)}}$ fire at some moment, sending one spike to each of the neurons $\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}$, and $\sigma_{l_i^{(9)}}$. Neurons $\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}$, and $\sigma_{l_i^{(9)}}$ fire at some step, removing the spikes in neurons $\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}$ and sending one spike to neuron σ_{l_k} from $\sigma_{l_i^{(9)}}$. Therefore, neuron σ_{l_k} becomes active, and the system Π starts to simulate instruction l_k of M. In neuron σ_r, there is no spike inside, which means that the number stored in register r of M is zero.

Note that it is possible to have interferences between two SUB modules. Specifically, if there are several SUB instructions l_v that act on the same register r, then neuron σ_r has synapses to all neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$. When a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is simulated, in the SUB module associated with l_v ($l_v \neq l_i$) all neurons receive no spike except for neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$. Each of neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$ has one spike inside. Because we have the ls-set $\{\sigma_{l_1^{(3)}}, \sigma_{l_2^{(3)}}, \ldots, \sigma_{l_{s_r}^{(3)}}, \sigma_{l_1^{(4)}}, \sigma_{l_2^{(4)}}, \ldots, \sigma_{l_{s_r}^{(4)}}, \sigma_{l_1^{(9)}}, \sigma_{l_2^{(9)}}, \ldots, \sigma_{l_{s_r}^{(9)}}\}$, when neuron $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ fire, each of neurons $\sigma_{l_v^{(3)}}$ and $\sigma_{l_v^{(4)}}$ ($l_v \neq l_i$) should also fire at the same step removing its spike. Consequently, the interference among SUB modules will not cause undesired steps in Π (i.e., steps that do not correspond to correct simulations of instructions of M). Therefore, the simulation of SUB instruction is correct.

With the above description, we can see that the unbounded SN P system Π' can correctly simulate register machine M, hence $N(M) = N(\Pi')$. □

Similar with Corollary 1, we can also have the following corollary.

Corollary 2. $NRE = NSpik_{out}P_{217}^{locsyn}(unb, 12)$.

Indeed, the system Π' has

- 9 neurons for the 9 registers,
- 25 neurons for the 25 labels,
- 6×11 neurons for the 11 ADD instructions,
- 9×13 neurons for the 13 SUB instructions,

which gives a total of 217 neurons. The reader can easily check that the local synchronization degree is 12.

6 Asynchronous Bounded SN P Systems with Local Synchronization

In this section, we investigate the computation power of asynchronous bounded SN P systems with local synchronization. It was shown that asynchronous bounded SN P systems with extended rules can characterize semilinear sets of numbers [4], but it is open whether this result holds when the systems are restricted to use only standard rules. In the following, we prove that a set of natural numbers is semilinear if only if it can be generated by asynchronous bounded SN P systems with local synchronization using standard rules.

Lemma 1. $NSpik_{out}P_*^{locsyn}(boun, *) \subseteq SLIN$.

Take an asynchronous bounded SN P system with local synchronization using standard rules, Π. The number of neurons is fixed, and the number of spikes in each neuron is bounded, hence the number of configurations reached by Π is finite. Let $\mathcal{C}$ be the set of configurations of Π, and C_0 be the initial configuration of Π.

We construct the right-linear grammar $G = (\mathcal{C}, \{a\}, C_0, P)$, where P contains the following productions:

Proof. (1) $C \to C'$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Rightarrow C'$ in Π during which the output neuron does not spike;

(2) $C \to aC'$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Rightarrow C'$ in Π during which the output neuron spikes;

(3) $C \to \lambda$, for any $C \in \mathcal{C}$ in which all neurons have no enabled rules.

Clearly, the construction of G ensures the fact that $N(\Pi)$ is the length set of the regular language $L(G)$, hence it is semilinear. Therefore, $NSpik_{out}P_*^{locsyn}(boun, *) \subseteq SLIN$. □

Lemma 2. $SLIN \subseteq NSpik_{out}P_*^{locsyn}(boun, *)$.

Since a set of natural numbers is semilinear if and only if it can be generated by a strongly monotonic register machine, it is enough to prove that any strongly monotonic register machine can be simulated by an asynchronous bounded SN P system with local synchronization using standard rules. Let M be a strongly monotonic register machine. Clearly, the machine M has only register 1 and the ADD instructions of the forms $l_i : (\texttt{ADD}(1), l_j, l_k)$.

An asynchronous bounded SN P system with local synchronization using standard rules Π can be constructed as in the proof of Theorem 1 to simulate the strongly monotonic register machine M: we place the rule $a^2 \to a$ in neuron σ_1 (it is associated with register 1 and it is also the output neuron), also considering the ls-set consisting of neurons σ_1 and $\sigma_{l_v^{(3)}}, \sigma_{l_v^{(4)}}$ for all ADD instructions l_v of M. Moreover, in neuron σ_{l_h} we provide no rule (when M halts, also Π halts, so the FIN module is here not necessary). The above mentioned new ls-set make sure that as soon as the simulation of an ADD instruction of M reaches the neurons $\sigma_{l_j^{(3)}}, \sigma_{l_j^{(4)}}$, then also neuron σ_1 spikes, thus getting empty, ready for a further increment by one. That is, neuron σ_1 is either empty or it contains two spikes – in the latter case, one spike is sent to the environment. Thus, the equivalence of M and Π is obvious. □

By Lemmas 1 and 2, the following theorem holds.

Proof. **Theorem 3.** $NSpik_{out}P_*^{locsyn}(boun, *) = SLIN$.

7 Conclusions and Remarks

In this work, inspired by the fact that neurons in the same functioning brain motif or community work synchronously to achieve some specific biological functions, we introduce local synchronization into the framework of SN P systems. The computation power of asynchronous SN P systems with local synchronization is investigated. Asynchronous SN P systems with local synchronization consisting of general neurons (resp. unbounded neurons) and using standard spiking rules are proved to be universal. Asynchronous SN P systems with local synchronization consisting of bounded neurons and using standard spiking rules can characterize the semilinear sets of natural numbers. It was already known that (1) asynchronous general SN P systems with extended rules are universal; (2) asynchronous unbounded SN P systems with extended rules can only characterize semilinear sets of natural numbers [2]. However, the computation power of asynchronous general (resp. unbounded) SN P systems with standard rules is unknown. The results from this paper show that such systems can reach universality if they are provided with the "programming capability" of local synchronization. So, local synchronization is a powerful ingredient for achieving "Turing creative capability".

The local synchronization degrees in Corollary 1 and Corollary 2 are 5 and 12, respectively. It remains open whether or not these values can be improved. We conjecture that the value two is enough to achieve universality for asynchronous SN P systems with local synchronization consisting of general neurons (resp. unbounded neurons) and using standard spiking rules.

In the systems constructed in Theorem 1 and Theorem 2, forgetting rules are used. It remains open whether forgetting rules can be removed without any loss of computation power. We conjecture the answer is positive (that is, we believe that the feature of local synchronization is powerful enough to remove forgetting rules without decreasing the computation power).

Acknowledgment

This work of the first two authors was supported by National Natural Science Foundation of China (61033003, 91130034 and 30870826), Ph.D. Programs Foundation of Ministry of Education of China (20100142110072), Fundamental Research Funds for the Central Universities (2010ZD001 and 2010MS003), and National Science Foundation of Hubei Province (2008CDB113 and 2011CDA027). The work of Gh. Păun is supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. U. Alon: *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman&Hall/CRC, 2006
2. M. Cavaliere, O. Egecioglu, O.H. Ibarra, S. Woodworth, M. Ionescu, Gh. Păun: Asynchronous spiking neural P systems: decidability and undecidability. *Proc. 13th Int. Meeting on DNA Computing* (M.H. Garzon, H. Yan, eds.), Memphis, USA, LNCS 4848, Springer, Berlin, 2008, 246–255.
3. H. Chen, M. Ionescu, T. Ishdorj, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules: universality and languages. *Natural Computing*, 7 (2008), 147–166.
4. O.H. Ibarra, S. Woodwort: Spiking neural P systems: some characterizations. *Preproc. 16th International Symposium on Fundamentals of Computation Theory, FCT 2007* (E. Csuhaj-Varjú, Z. Ésik, eds.), Budapest, Hungary, LNCS 4639, Springer, Berlin, 2007, 23–37.
5. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71 (2006), 279–308.
6. I. Korec: Small universal register machines. *Theoretical Computer Sci.*, 168 (1996), 267–301.
7. C. Martin-Vide, J. Pazos, Gh. Păun, A. Rodriguez-Patón: Tissue P systems. *Theoretical Computer Sci.*, 296 (2003), 295–326.
8. M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, New Jersey, 1967.
9. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143.
10. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
11. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford Univ. Press, Oxford, 2010.
12. Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems: An overview. *Advancing Artificial Intelligence through Biological Process Applications* (A.B. Porto, A. Pazos, W. Buno, eds.), PA: Medical Information Science Reference, Hershey, 2008, 60–73.
13. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Springer, Berlin, 1991.

Improving the Universality Results of Enzymatic Numerical P Systems

Cristian Ioan Vasile[1], Ana Brânduşa Pavel[1], and Ioan Dumitrache[1]

Department of Automatic Control and Systems Engineering
Politehnica University of Bucharest
Splaiul Independenţei 313, 060042 Bucharest, Romania
{cvasile, apavel, idumitrache}@ics.pub.ro

Summary. This paper provides the proof that Enzymatic Numerical P Sytems with deterministic, but parallel, execution model are universal, even when the production functions used are polynomials of degree 1. This extends previous known results and provides the optimal case in terms of polynomial degree.

1 Enzymatic Numerical P Systems

Numerical P Systems (NP Systems) are a type of P systems [8], inspired by the cell structure, in which numerical variables evolve inside the compartments by means of programs; a program (or rule) is composed of a production function and a repartition protocol. The variables have a given initial value and the production function is a multivariate polynomial. The value of the production function for the current values of the variables is distributed among variables in certain compartments according to a repartition protocol. Formal definition of NPS can be found in [7] where the authors introduce this type of P systems with possible applications in economics.

NP systems were designed both as deterministic and non-deterministic systems [7]. Non-deterministic NPS allow the existence of more rules per each membrane and the best rule is selected by an "oracle", while the deterministic NPS can have only one or no rule per each membrane. NP Systems were used as a naturally parallel and distributed modeling tool for the design of robot controllers [1], [5], [6]. Designing robot controllers requires deterministic mechanisms. Therefore, an extension of NPS, Enzymatic Numerical P systems (EN P Systems), in which enzyme-like variables allow the existence of more than one program (rule) in each membrane, while keeping the deterministic nature of the system, were introduced in [3]. Due to their properties, EN P Systems represent a more powerful modeling tool for robot behaviors than classical N P Systems [5], [6].

2 The power of Enzymatic Numerical P Systems

In [9] the authors prove and analyze the universality of EN P Systems. The main results in [7] and [9] regarding the power of NP Systems and ENP Systems are the following:

Theorem 1. $NRE = N^+P_8(poly^5(5), seq) = N^+P_7(poly^5(6), seq) = NP_7(poly^5(5), enz, seq) = NtP_*(poly^1(11), enz, oneP) = NP_{254}(poly^2(253), enz, allP, det)$.

The family of sets of numbers $N^+(\Pi)$ computed by NP Systems with at most m membranes, production functions which are polynomials of degree at most n, with integer coefficients, with at most r variables in each polynomial, is denoted by $N^+P_m(poly^n(r), seq)$, $m \geq 1, n \geq 0, r \geq 0$, where the fact that they work in the sequential mode (in each step, only one program is applied), is indicated by *seq*. If one of the parameters m, n, r is not bounded, then it is replaced by $*$. (Both in $N^+(\Pi)$ and in $N^+P_m(poly^n(r), seq)$, the superscript + indicates the fact that as the result of a computation we only consider positive natural numbers, zero excluded. If any value of x_{j_0,i_0} is accepted, then the superscript + is removed.) When tissue-like systems are used, we write $NtP_m(poly^n(r), \alpha, \beta)$.

In the next section, the authors present an improvement of the universality results of EN P Systems by reducing the number of membranes, the polynomial degree of the production functions and the number of variables of the production functions.

3 Improving the universality results

The main result proposed here is the following theorem about the power of EN P Systems. The theorem extends previous results about the benefits of adding the enzymatic mechanism in terms of the computational power of the model. It also provides the optimal result regarding the polynomial degree of the production functions, namely 1. The execution model considered in the theorem is a deterministic, but parallel one, in which all active rules are executed in parallel. Rules, which share variables, will use the current value of the variable and execute independently of each other.

Theorem 2. $NRE = NP_4(poly^1(6), enz, allP, det)$.

Proof. The proof is done by constructing a membrane system which enumerates the positive values of some polynomial with integer coefficients corresponding to tuples of natural numbers. It is proven in [2], that polynomials of degree at most 5 with 5 variables are sufficient to imply the universality of the models. This technique is used to show that standard NP Systems are universal [7]. The following system is a modified version of the one used in [9] (it is also shown in graphical form in figure 1):

$$
\begin{aligned}
\Pi = (4, H, \mu, &(Var_{Generate}, Pr_{Generate}, Var_{Generate}(0))\\
&(Var_{Compute}, Pr_{Compute}, Var_{Compute}(0), enum),\\
&(Var_{Pow5}, Pr_{Pow5}, Var_{Pow5}(0)),\\
&(Var_{Mult}, Pr_{Mult}, Var_{Mult}(0))),\\
H &= \{Generate, Compute, Pow5, Mult\},\\
\mu &= [_{Generate}[_{Compute}[_{Pow5}\ [_{Mult}]_{Mult}\ ]_{Pow5}]_{Compute}]_{Generate},\\
Var_{Generate} &= \{x_i, e_j, ez_k, er_i, n, e_t, g, gc :\ 1 \le i \le 5, 1 \le i \le 7, 1 \le k \le 5\},\\
Pr_{Generate} &= \{n \to 1|n,\ e_t \to 1|gc,\\
&\quad 1 + x_1|_{e_1} \to 1|er_1,\ -1 + g|_{e_1} \to 1|x_1,\ 1 + n + x_1|_{e_1} \to 1|x_1\}\\
&\cup \{j \cdot e_j \to 1|e_{j+1} + (j-1)|e_t :\ 1 \le j \le 5\}\\
&\cup \{1 + x_i|_{e_1} \to 1|ez_i,\ 1 - i + e_t|_{er_{i-1}} \to 1|x_i :\ 2 \le i \le 5\}\\
&\cup \{g + (ez_i + er_{i-1})|_{e_i} \to 1|er_i,\ 2 - i + n + e_t|_{er_i} \to 1|x_i\\
&\quad :\ 2 \le i \le 5\}\\
&\cup \{2 + e_t|_{er_5} \to \sum_{i=1}^{5} 1|x_i + 1|n,\ er_5|_{e_6} \to 1|gc,\ e_6 \to 1|e_7,\\
&\quad e_7 \to 1|e_1^c\}\\
&\cup \{g + 2 \cdot x_i|_{e_7} \to 1|x_i + 1|x_i^c\ :\ 1 \le i \le 5\},\\
Var_{Generate}(0) &= (5,5,5,5,5,1,0,\ldots,0,5,0,0,0),\\
Var_{Compute} &= \{x_i^c, e_j^c, t, g^*, ep_t, f_Q, enum, aux, e_f :\ 1 \le i \le 5, 1 \le j \le 506\},\\
Pr_{Compute} &= \{g^* + \left(\sum_{i=1}^{5} a_{i,k} \cdot x_i^c + a_{6,k}\right)|_{e^c_{2k-1}} \to 1|s_1,\\
&\quad 3 \cdot e^c_{2k-1} \to 1|e^c_{2k} + 2|ep_1,\ e^c_{2k}|_{ep_t} \to 1|e^c_{2k+1},\\
&\quad g^* - 2 \cdot \beta_k t|_{e^c_{2k+1}} \to 1|aux + 1|e_f\ :\ 1 \le k \le 252\}\\
&\cup \{2 \cdot e^c_{505} \to 1|e_f + 1|e^c_{506},\ aux|_{e_f} \to 1|f_Q,\ -f_Q|_{g*} \to 1|enum,\\
&\quad -(f_Q + e^c_{506}) \to 1|e_f,\ enum + f_Q + ep_t \to 1|gc,\ e^c_{506} \to 1|e_1\}\\
Var_{Compute}(0) &= (0,0,\ldots,0),\\
Var_{Pow5} &= \{s_1, s_2, ep_i, e_M, gc^*, z\ :\ 1 \le i \le 7\},\\
Pr_{Pow5} &= \{z + 3 \cdot s_1|_{ep_1} \to 1|a + 1|b + 1|s_1,\ z + 2 \cdot s_2|_{ep_3} \to 1|a + 1|b,\\
&\quad z + s_1|_{ep_5} \to 1|a,\ z + s_2|_{ep_5} \to 1|b,\ z + s_2|_{ep_7} \to 1|t,\\
&\quad ep_7 \to 1|ep_t,\ e_M \to 1|gc^*\}\\
&\cup \{3 \cdot ep_{2k-1} \to 1|ep_{2k} + 2|e_s,\ ep_{2k}|_{e_M} \to 1|ep_{2k+1},\ 1 \le k \le 3\}\\
Var_{Pow5}(0) &= (0,0,\ldots,0),\\
Var_{Mult} &= \{a, b, z^*, d, u, e_s\},\\
Pr_{Mult} &= \{z^* + 1.5 \cdot a|_b \to 2|a + 1|s_2,\ z^* - (1+d)|_b \to 1|d,\ d \to 1|b
\end{aligned}
$$

$$e_s + b|_u \to 1|e_M,\ a + b|_u \to 1|gc^*\}$$
$$Var_{Mult}(0) = (0, 0, 0, 0, 1, 0).$$

The proposed membrane system is mainly composed of two parts: the 5-tuple generation part and the computation of the polynomial's value. The generating part, implemented in the *Generate* membrane, is the same as in the proof from [9]. Only the last rule of the membrane, $e_7 \to 1|e_1^c$, was changed in order to synchronize it with the *Computation* membrane. The 5-tuple generation process is described in detail in [9]. The five variables forming the tuple are regarded as a single number with 5 digits in a certain base. The algorithm counts down from the highest 5-digit number to zero. Therefore, if the current base is $b+1$, the membrane will generate the numbers from *bbbbb* to 00000. When the null tuple is reached the variables are reset to the highest 5-digit number of the next base. The algorithm start with base 6 from the tuple $(5, 5, 5, 5, 5)$ and generates $(5, 5, 5, 5, 4)$, ..., $(5, 5, 5, 5, 0)$, $(5, 5, 5, 4, 5)$, ..., $(0, 0, 0, 0, 0)$. At this point it will move to the tuple $(6, 6, 6, 6, 6)$ which corresponds to the highest 5-digit number in base 7. The process repeats indefinitely, thus generating all 5-tuples of natural numbers in a deterministic way.

For the next part of the proof it is important to recall that every polynomial f of degree 5 with 5 variables can be put in the following form (lemma from [9]):

$$f(x_1, \ldots, x_5) = \sum_{i=1}^{m} \beta_i \cdot (a_{1,i}x_1 + \ldots + a_{5,i}x_5 + a_{6,i})^5 \tag{1}$$

where m is 252 and represents the maximum number of terms of f in the general form, β_i are polynomial specific coefficients and $a_{j,i}$ are some constants. This form of the polynomial is used in order to compute the values corresponding to the generated 5-tuples in the first part of the procedure in the *Generate* membrane.

The *Compute* membrane was rewritten such that only polynomials of degree one are used as production functions. This is achieved by noting that the only part where polynomials of degree greater than one are needed is when the 5-th power of a number is computed, more specifically a natural number [9]. Computing the power of a number can be done using only multiplication; computing the 5-th power of x can be done by first computing $a = x \cdot x = x^2$, then $b = a \cdot a = x^4$ and finally $c = x \cdot b = x^5$. Since x in the system is a natural number, multiplication can be performed as a repeated addition, $a \cdot b = \underbrace{a + \ldots + a}_{b}$. This procedure is implemented in the *Pow5* membrane which repeatedly uses the *Mult* membrane to compute the products of natural numbers. Thus the degree of the polynomials in all production functions is reduced to 1, the optimal value.

Also, the number of membranes needed in the computation was reduced by reusing some membranes, *Pow5* and *Mult*. Instead of using $m = 252$ *Pow5* membranes in order to compute the m terms of the polynomial (in the form from equation 1), the membrane is used repeatedly to compute each term. Therefore, the number of membranes is reduced to 4. □

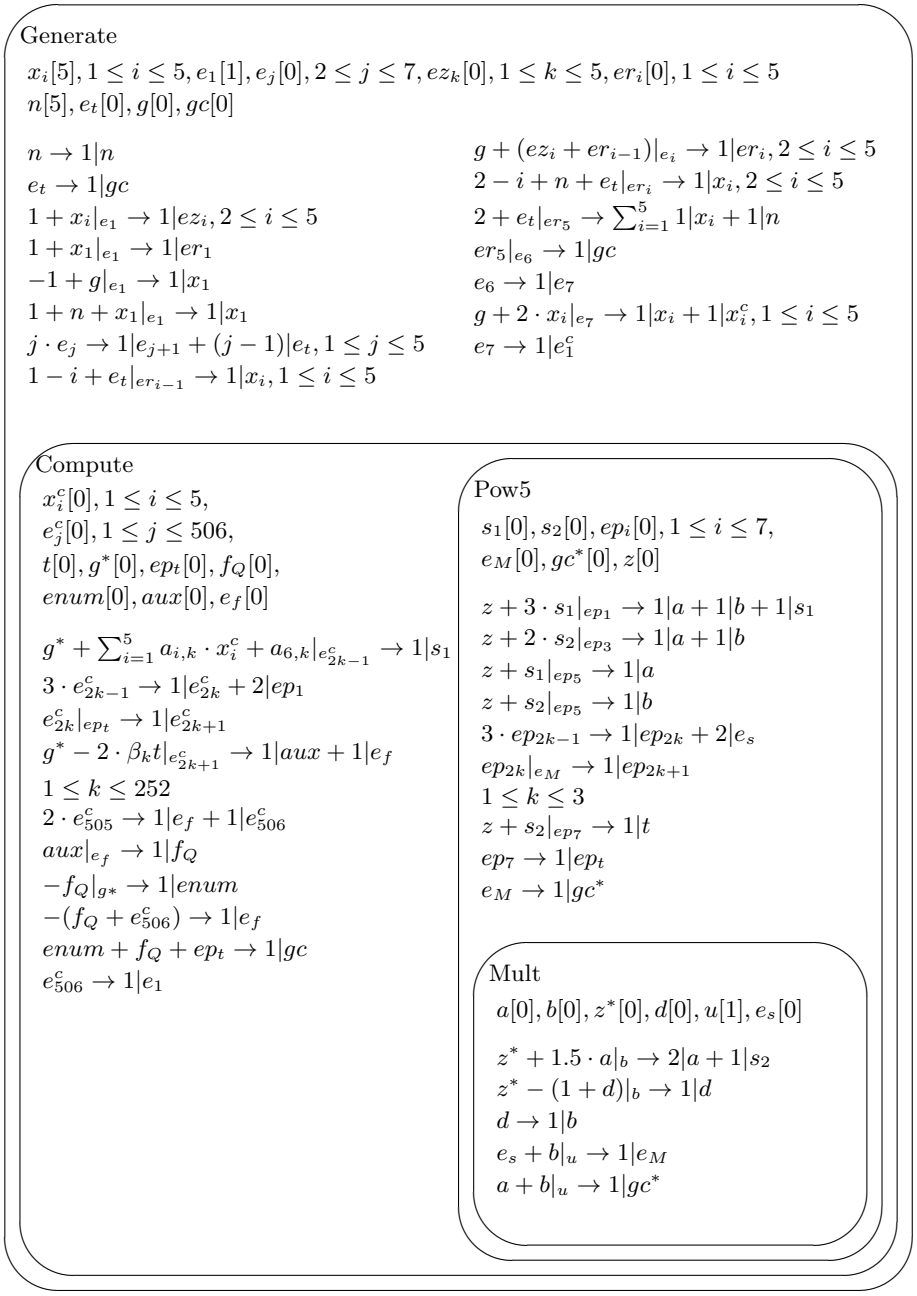


Fig. 1. The EN P system from the proof of Theorem 2

Parts of the proposed membrane system, *Generate* membrane and *Pow5* membrane, were simulated and verified using SimP, an EN P Systems simulator proposed in [4].

4 Remarks

In the proof of theorem 2 a method of reusing membranes was used in order to reduce the number of membranes in the system. It is, however, important to notice that it also constrained the system to perform most important computations in a serial manner. In practice, it may be more convenient to have more membranes that compute in parallel, because it allows the underlying runtime environment to perform optimizations based on available hardware and software platform. It is also important to note that there are more rules dedicated to program control flow in the membrane system from theorem 2 than there are in the one from theorem 4 in [9].

Another important observation is that even though computation can be done with polynomial production functions of degree 1, in some cases it is more convenient to use higher degree polynomials. However, most rules used for program flow control are of degree 1 and also, most rules with higher degree polynomial productions functions have few terms. These observations are relevant for optimizing the data structures and algorithms used for simulating EN P Systems.

Acknowledgments

This paper is supported by the Sectorial Operational Program Human Resources Development, financed from the European Social Fund and by the Romanian Government under the contract number SOP HRD/107/1.5/S/82514.

References

1. Buiu, C., Vasile, C.I., Arsene, O.: Development of membrane controllers for mobile robots. Information Sciences (in press), doi: 10.1016/j.ins.2011.10.007
2. Minsky, M. (ed.): Computation: Finite and Infinite Machines. Prentice-Hall (1967)
3. Pavel, A., Arsene, O., Buiu, C.: Enzymatic numerical P systems - a new class of membrane computing systems. In: The IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010) Liverpool. pp. 1331–1336 (September 2010)
4. Pavel, A.B.: Membrane controllers for cognitive robots. Master's thesis, Department of Automatic Control and System Engineering, Politehnica University of Bucharest, Romania (February 2011)
5. Pavel, A.B., Buiu, C.: A software tool for modeling and simulation of numerical P systems. Natural Computing (in press), doi: 10.1007/s11047-011-9286-5
6. Pavel, A.B., Vasile, C.I., Dumitrache, I.: Robot localization implemented with enzymatic numerical P systems (submitted)

7. Păun, G., Paun, A.: Membrane Computing and Economics: Numerical P Systems. Fundamenta Informaticae pp. 213–227 (2004)
8. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
9. Vasile, C.I., Pavel, A.B., Dumitrache, I., Păun, G.: On the Power of Enzymatic Numerical P Systems (submitted)

Implementing Obstacle Avoidance and Follower Behaviors on Koala Robots Using Numerical P Systems

Cristian Ioan Vasile[1], Ana Brânduşa Pavel[1], Ioan Dumitrache[1], and Jozef Kelemen[2]

[1] Department of Automatic Control and Systems Engineering
Politehnica University of Bucharest
Splaiul Independenţei 313, 060042 Bucharest, Romania
{cvasile, apavel, idumitrache}@ics.pub.ro
[2] Institute of Computer Science
Silesian University in Opava
kelemen@fpf.slu.cz

Summary. Membrane controllers have been developed using Numerical P Systems and their extension, Enzymatic Numerical P Systems, for controlling mobile robots like e-puck and Khepera III. In this paper we prove that membrane controllers can be easily adapted for other types of robotic platforms. Therefore, obstacle avoidance and follower behaviors were adapted for Koala robots. The membrane controllers for Koala robots have been tested on real and simulated platforms. Experimental results and performance analysis are presented.

1 Introduction

Numerical P systems represent a type of membrane systems introduced by Gh. Păun in [7]. The main difference of this computational model in comparison to other types of P systems [8] is that compartments contain numerical variables (instead of symbols) which evolve by means of programs (rules). A membrane system has a tree-like structure and computation takes place in parallel in all membranes. Using membrane computing paradigm for modeling robot controllers is a new approach that was discussed and analysed in several papers [2], [5], [1]. Numerical P Systems (NPS) and their extension, Enzymatic Numerical P Systems (ENPS), were successfully applied for modeling robot behaviors like obstacle avoidance, wall following, following another robot, localization [2], [5], [6]. The robot behaviors were tested on real and simulated two wheel differential robots: e-puck and Khepera III. These two types of robots are small educational robots with diameters between about 7 and 13 cm. In this paper, we propose robot controllers for following a leader and obstacle avoidance behaviors, which were tested on Koala

educational robots. Koala robot is a bigger robot, about 30x30 size, with different infrared sensor placement than e-puck and Khepera III. In this way we prove that membrane controllers can be easily adapted to any types of robots and are a scalable modeling tool for robotics applications. The membrane controllers for the follower behavior are modeled based on classical control laws (proportional controller) using membrane systems. The obstacle avoidance behavior is presented in detail in [5]. The membrane controllers' performance will be analyzed and presented in this paper.
In order to introduce the NPS and ENPS models and the mathematical notations, we further present formal definitions which were adapated from other papers. For instance numerical P systems are presented in detail in [7]. Their definition is the following:

$$\Pi = (m, H, \mu, (Var_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, Var_m(0))) \tag{1}$$

where:

- m is the number of membranes used in the system, degree of Π; $m \geq 1$;
- H is an alphabet that contains m symbols (the labels of the membranes);
- μ is a membrane structure;
- Var_i is the set of variables from compartment i, and the initial values for these variables are $Var_i(0)$;
- Pr_i is the set of programs (rules) from compartment i. Programs process variables and have two components, a production function and a repartition protocol.
  The j-th program has the following form:

$$Pr_{j,i} = (F_{j,i}(x_{1,i}, ..., x_{k_i,i}), c_{j,1}|v_1 + ... + c_{j,n_i}|v_{n_i}) \tag{2}$$

  where:
  - $F_{j,i}(x_{1,i}, ..., x_{k_i,i})$ is the production function;
  - k_i represents the number of variables in membrane i;
  - $c_{j,1}|v_1 + ... + c_{j,n_i}|v_{n_i}$ is the repartition protocol;
  - n_i represents the number of variables contained in membrane i, plus the the number of variables contained in the parent membrane of i, plus the number of variables contained in the children membranes of i.

The variables $c_{j,1}, \ldots, c_{j,n_i}$ are natural numbers (they may be also 0, case in which it is omitted to write "$+0|x$") [7]. These coefficients specify the proportion of the current production distributed to each variable $v_1, ..., v_{n_i}$. Let,

$$C_{j,i} = \sum_{n=1}^{n_i} c_{j,n} \tag{3}$$

A program $Pr_{j,i}$ is executed as follows. At any time t, the function $F_{j,i}(x_{1,i}, ..., x_{k_i,i})$ is computed. The value:

$$q = \frac{F_{j,i}(x_{1,i}, ..., x_{k_i,i})}{C_{j,i}} \tag{4}$$

represents the "unitary portion" to be distributed to variables $v_1, \ldots, v_{n_i}$, according to coefficients $c_{j,1}, \ldots, c_{j,n_i}$ in order to obtain the values of these variables at time $t + 1$. Specifically, variable v_s which belongs to the repartition protocol of program j, will receive:

$$q * c_{j,s}, for 1 \leq s \leq n_i \tag{5}$$

The variables which receive new values from a rule must be contained within the current, the parent or a child membrane. If a variable belongs to membrane i, it can appear in the repartition protocol of the parent membrane of i and also in the repartition protocol of the child membranes of i. After applying all the rules, if a variable receives such "contributions" from several neighboring compartments, then they are added in order to produce the next value of the variable.

A production function which belongs to membrane i may depend only on some of the variables from membrane i. Those variables which appear in the production function become 0 after the execution of the program.

Deterministic NPS have only one rule per membrane ($card(Pr_i) = 1$) or must have a selection mechanism that can decide which rule to apply. The NPS model with multiple rules per membrane is a non-deterministic system. However, NPS are well suited for applications which involve numerical variables and require a deterministic behavior, such as control systems for mobile robots. Thus a selection mechanism for the active rules is defined in the extended model, enzymatic numerical P systems (ENPS), proposed [3].

ENPS is defined as a NPS with special enzyme-like variables which control the execution of the rules:

$$\Pi = (m, H, \mu, (Var_1, E_1, Pr_1, Var_1(0)), \ldots, (Var_m, Pr_m, E_m, Var_m(0))) \tag{6}$$

where:

- E_i is a set of enzyme variables from compartment i, $E_i \subset Var_i$
- Pr_i is the set of programs from compartment i. Programs have one of the two following forms:
  1. non-enzymatic form, which is exactly like the one from the standard NPS:

$$Pr_{j,i} = (F_{j,i}(x_{1,i}, \ldots, x_{k_i,i}), c_{j,1}|v_1 + ... + c_{j,n_i}|v_{n_i}) \tag{7}$$

  2. enzymatic form

$$Pr_{j,i} = (F_{j,i}(x_{1,i}, \ldots, x_{k_i,i}), e_{t,i}, c_{j,1}|v_1 + ... + c_{j,n_i}|v_{n_i}) \tag{8}$$

  where $e_{t,i} \in E_i$

There can be more than one active rule in a membrane or none. A rule is active if it is in the non-enzymatic form or if the associated enzyme has a greater value than one of the variables involved in the production function, in absolute value. All active rules in the membrane system are executed in parallel in one computational step. The enzymatic mechanism and the advantages of ENPS are detailed in [5], [1].

2 Behaviors on Koala robot

In this paper two behaviors are presented and tested on simulated and real Koala robots: obstacle avoidance and following a leader. The obstacle avoidance behavior is simple to define: the robot has to be able to maintain a minimum distance from all other objects in the environment. Although simple in principle, a lot of problems can arise by taking into account the physical constraints of the robot, such as the limited perception (sensors' detection range, precision, accuracy, sampling time, etc.) and limited effector action (speed, acceleration, torque, force-stress, etc.). These factors together with the nature and structure of the environment play a very important role in the design of an effective control strategy not only for obstacle avoidance, but also for any other behavior (simple or complex). The control law of the obstacle avoidance behavior uses the infrared sensor's readings to compute appropriate speeds for the two motors of the Koala robot (figure 3). This data is sufficient to ensure that the robot will not come in contact with any object in the environment. If no obstacle are in sight, the robot just cruises forward at a given constant speed.

The behavior of following a leader robot is more difficult to perform just from infrared sensors' readings (figure 3), because objects from the environment cannot be distinguished from the leader just from this data. On the other hand the control program that uses only the infrared sensors is much more simple and easier to implement.

In this paper, experiments are carried out to show that membrane controllers are viable control strategies for robots operating in a semi-structured office-like environment.

In previous work, a proportional controller for the follower behavior was designed using Numerical P Systems, for Khepera III and e-puck robots [2]. The membrane structure is illustrated in figure 1.

The control law implemented by the membrane structure in figure 1 is the following:

$$\begin{aligned} lw = {} & CruiseSpeedLeft - k_{Dist} * (ref_{Dist} - 0.5 * (s_{Dist1} + s_{Dist2})) \\ & + k_{Heading} * (ref_{Heading} - (s_{R1} + s_{R2} - (s_{L1} + s_{L2}))) \end{aligned} \tag{9}$$

$$\begin{aligned} rw = {} & CruiseSpeedRight - k_{Dist} * (ref_{Dist} - 0.5 * (s_{Dist1} + s_{Dist2})) \\ & - k_{Heading} * (ref_{Heading} - (s_{R1} + s_{R2} - (s_{L1} + s_{L2}))) \end{aligned} \tag{10}$$

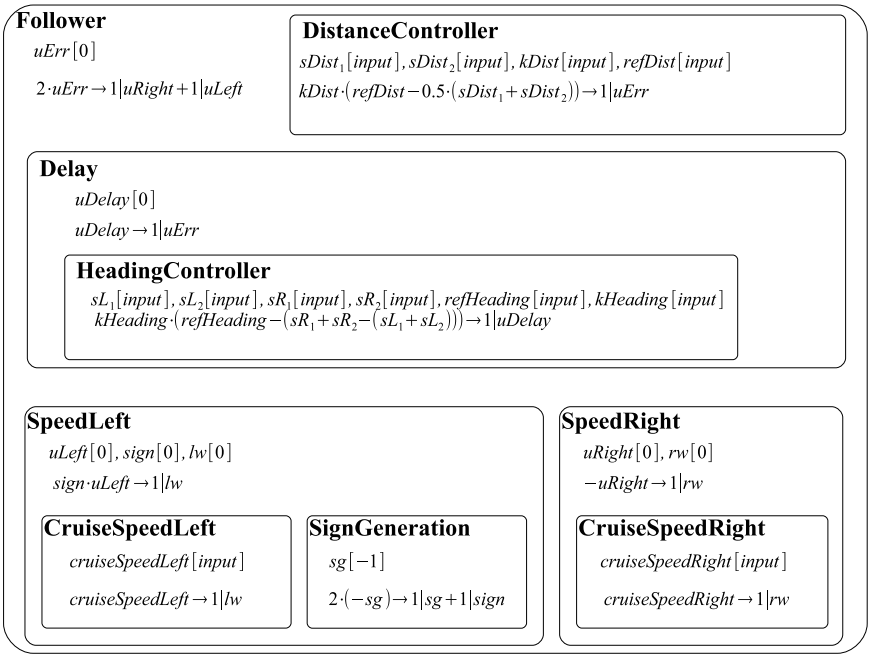


Fig. 1. A proportional controller for the follower behavior, implemented with NPS, for Koala III and e-puck robots

The follower controller was adapted to work with Koala robots, taking into account the sensors' placement on this type of robots. Therefore, we can prove that membrane controllers can be adapted to work on any type of robotic platforms (figure 2).

The control law for the follower behavior of Koala robot is:

$$lw = CruiseSpeedLeft - k_{Dist} * (ref_{Dist} - \frac{1}{6}\sum_{i=1}^{6} s_{Disti}) + k_{Heading} * (ref_{Heading} - (\sum_{i=1}^{6} s_{Ri} - \sum_{i=1}^{6} s_{Li})) \quad (11)$$

$$rw = CruiseSpeedRight - k_{Dist} * (ref_{Dist} - \frac{1}{6}\sum_{i=1}^{6} s_{Disti}) - k_{Heading} * (ref_{Heading} - (\sum_{i=1}^{6} s_{Ri} - \sum_{i=1}^{6} s_{Li})) \quad (12)$$

The NPS structure tested on Koala was computed using SimP simulator proposed in [4]. The follower robot behavior was simulated using Webots simulator and then tested on real Koala robots (figure 3).

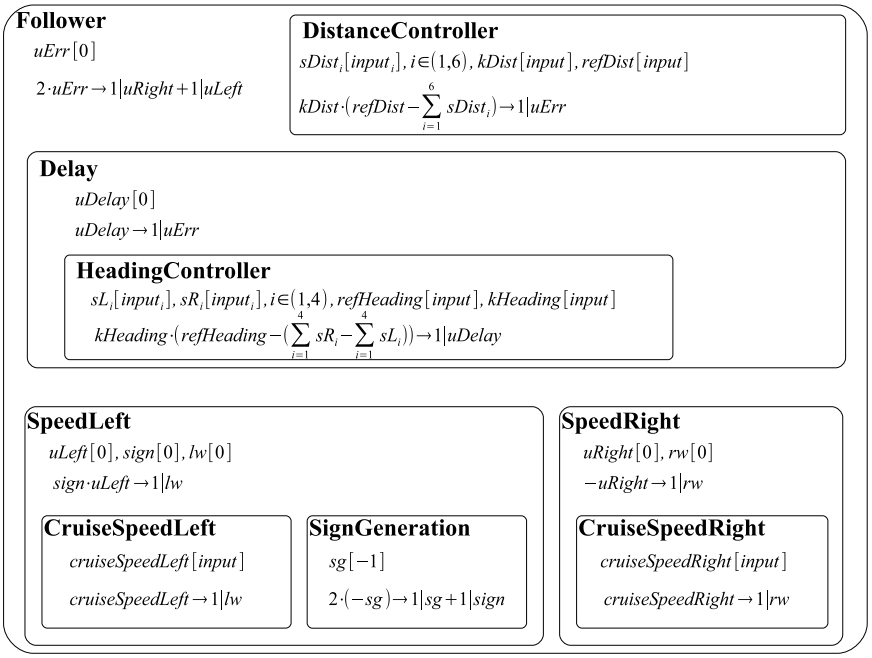


Fig. 2. A proportional controller for the follower behavior, implemented with NPS, for Koala robot

Fig. 3. Koala robot

An equivalent membrane controller using ENPS model has been created as well and is shown in figure 4. The structure of the ENPS controller is simpler and easier to understand and use. It also has less rules and membranes.

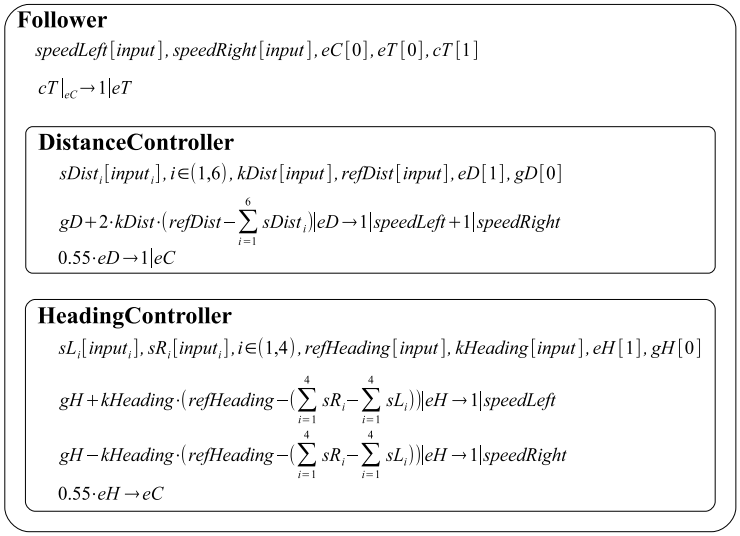


Fig. 4. A proportional controller for the follower behavior, implemented with ENPS, for Koala robot

In the experiments the leader Koala robot performs different predefined motions and also obstacle avoidance behavior. The membrane controller for obstacle avoidance was adapted from the one in [5] and is shown in figure 5. It is based on the ENPS model and the only thing that needed to be changed was to add more membranes for the extra infrared sensors of the Koala robot and the weight of each sensor, which has a different placement than on the other robots. In [5] it is presented how the membrane controller for obstacle avoidance works and what all variables are used for.

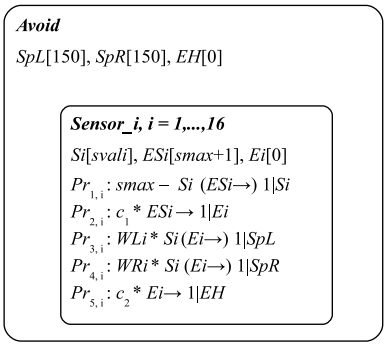


Fig. 5. A controller for the obstacle avoidance behavior, implemented with ENPS, for Koala robot

3 Experiment setup and performance analysis

As stated above, experiments were performed in the WeBots robotic simulator and in a real world setting. In both cases, the environment is a semi-structured office-like environment with or without obstacles. Both obstacle avoidance and follow a leader behaviors were tested, first separately and then together (the leader was performing obstacle avoidance).

All experiments used the SimP membrane simulator to execute the membrane controller that drive the robots. SimP[3] is described in [4] and can be used standalone or as a library. For the experiments on the real Koala robots, a server version was developed in order to respond to the robots' queries. In this setup, a TCP/IP connection is established from each robot to a host machine (in this case a laptop) through a wireless router. The robots run a program that queries the SimP server for the current motor commands based on the sensors' readings. This is done, because SimP is implemented in Java and the robots do not posses the necessary computational capabilities in order to run a Java Virtual Machine.

The performance of the controllers is analyzed based on the speed profiles of the leader and follower robots and also based on the duration of a cycle, the execution time of a membrane controller. The two behaviors were tested separately and together in different scenarios as follows.

Figures 6, 7, 8, 9 show the speed profiles of the leader and follower robots in simulated experiments. These figures are the result of three experiments for the following the leader behavior.

In the first experiment, the leader robot has a forward constant motion with a cruising speed (15) different then that of the follower robot (12). It is clear from figure 6 that the follower matched the speed of the leader and thus maintaining the desired distance to the leader.

In the second experiment, the leader robot has a forward variable motion. The speed of the leader oscillates around the cruising speed of 15 with amplitude $A = 10$ and frequency of about $f = 1.6e - 4$ (equation 13) . Figure 7 shows that the follower is able to match the speed of the leader in this experiment as well. However, a slight phase shift and amplitude difference can be observed in the graph and this is due to the reaction time of the follower, which is at least as long as the duration of a controller cycle. The cruising speed of the follower is also 12 in this experiment.

$$SpeedLeft = CruiseSpeed + A \cdot \sin(2\pi \cdot f \cdot t) \tag{13}$$

$$SpeedRight = CruiseSpeed + A \cdot \sin(2\pi \cdot f \cdot t) \tag{14}$$

In the third experiment, the leader has a variable sine motion. The speed is computed as composition of the constant forward cruising speed and a variable

[3] For the Java binary program (.jar) please contact A.B. Pavel at anabrandusa@gmail.com

turning speed (equation 15). The speed of the left motor is phase shifted by π from the right one. In this case, the follower is still able to follow the leader. However, there is a big difference in the speed profiles of the two robots. These are due to the fact that the Koala robots are square; when the leader turns, its back gets closer to the follower even though the leader is moving away. This is why the follower has to slow down when the leader changes direction and this can be seen in figures 8, 9.

$$SpeedLeft = CruiseSpeed + A \cdot \sin(2\pi \cdot f \cdot t) \tag{15}$$

$$SpeedRight = CruiseSpeed - A \cdot \sin(2\pi \cdot f \cdot t) \tag{16}$$

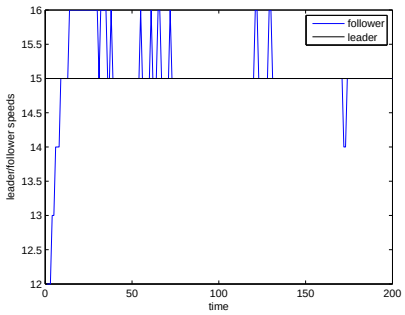


Fig. 6. Speeds of the leader and follower during the forward constant motion

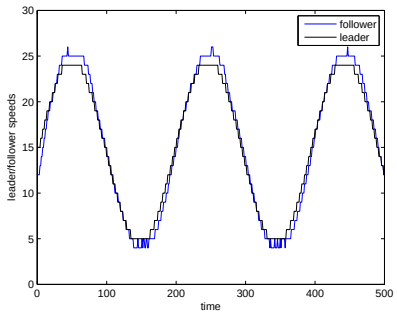


Fig. 7. Speeds of the leader and follower during the forward variable motion

The next figures were obtained from experiments in a real world experiments. Figure 10 shows the speed profile for obstacle avoidance controller. It can be noted that the peaks in the graph correspond to the reaction of the robot when obstacles are detected. The Koala robot was able to avoid all obstacles in the environment.

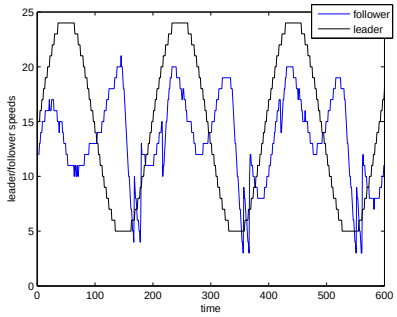


Fig. 8. Speeds of the leader and follower during the sine motion on the left wheels

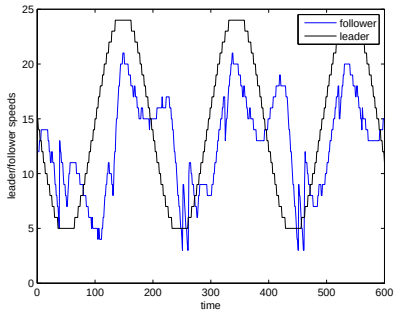


Fig. 9. Speeds of the leader and follower during the sine motion on the right wheels

In figure 11, the speed profiles of a follower robot is shown. The leader has a variable sine motion (equation refeq:leader-sine). It can be seen that the follower robot is able to match the movement of the leader and to follow it.

In the last experiment, the two behavior are used together: the leader performs obstacle avoidance while the other robot follows it. The speed profiles of both robots are presented in figures 12 and 13. The follower is able to keep track of the leader. It is important to note, that the follower must not get to close to other objects in the environment, because it can not distinguish them from the leader robot, only based on the infrared sensors' readings.

In the last figure (figure 14) the execution time of the two membrane controllers in each cycle is shown. It can be seen that the execution time is very stable and small. The first cycles take more time, however, due to initialization of the Java environment. After that, there are no significant peaks in the execution time for the proposed membrane systems.

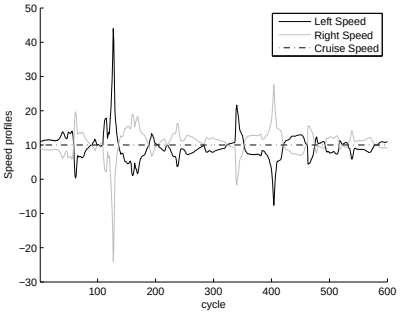


Fig. 10. avoid speed profiles

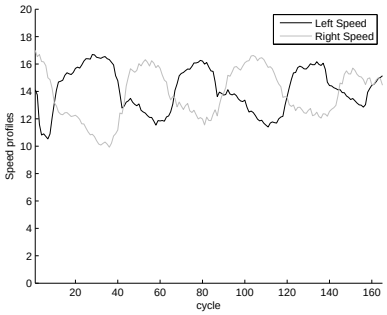


Fig. 11. follow sine speed profiles

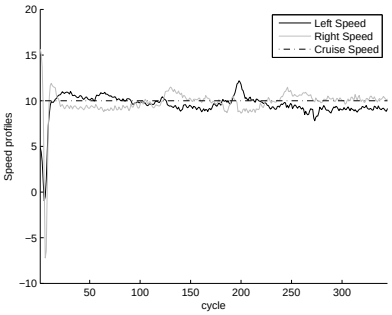


Fig. 12. follow avoid speed profiles

4 Conclusions

In this paper, we have proven that NPS and their extension, ENPS, are a flexible and scalabale modeling tool which can be succesfully used to design robot controllers. Among the advantages of using this computational paradigm in robotics

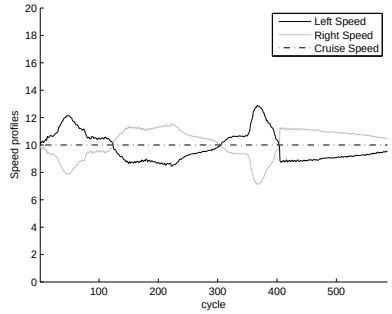


Fig. 13. leader avoid speed profiles

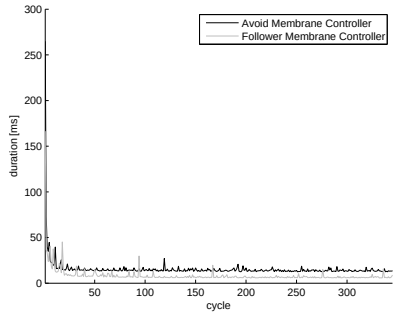


Fig. 14. Avoid/Follow cycle duration

we mention the parallel nature of the model and the possibility of encapsulating behaviors and functionalities as modules which can be executed in parallel. The membranes are independent from the control program of the robotic system and can be easily adapted to other types of robots or applications by only modifying some parameters in the xml files which store the membrane structures.

Future work includes extending the current membrane controllers to other robots, but also to implement other behaviors and functionalities using ENPS model.

References

1. Buiu, C., Pavel, A., Vasile, C., Dumitrache, I.: Perspectives of using membrane computing in the control of mobile robots. In: In Proc. of the Beyond AI - Interdisciplinary Aspect of Artificial Inteligence Conference, Pilsen, Czech Republic. pp. 21–26 (December 2011)
2. Buiu, C., Vasile, C.I., Arsene, O.: Development of membrane controllers for mobile robots. Information Sciences 187, 22–51 (March 2012), doi: 10.1016/j.ins.2011.10.007

3. Pavel, A., Arsene, O., Buiu, C.: Enzymatic numerical P systems - a new class of membrane computing systems. In: The IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010) Liverpool. pp. 1331–1336 (September 2010)
4. Pavel, A.B.: Membrane controllers for cognitive robots. Master's thesis, Department of Automatic Control and System Engineering, Politehnica University of Bucharest, Romania (February 2011)
5. Pavel, A.B., Buiu, C.: Using enzymatic numerical P systems for modeling mobile robot controllers. Natural Computing (in press), doi: 10.1007/s11047-011-9286-5
6. Pavel, A.B., Vasile, C.I., Dumitrache, I.: Robot localization implemented with enzymatic numerical P systems (submitted)
7. Păun, G., Paun, A.: Membrane Computing and Economics: Numerical P Systems. Fundamenta Informaticae pp. 213–227 (2004)
8. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)

A Note on the Probabilistic Evolution for P Systems

Sergey Verlan

Laboratoire d'Algorithmique, Complexité et Logique,
Université Paris Est – Créteil Val de Marne,
61, av. gén. de Gaulle, 94010, Créteil, France
verlan@univ-paris12.fr

Summary. In this note we propose a method that permits to describe in a uniform manner variants of probabilistic/stochastic P systems. We give examples of such a description for existing models of P systems using probabilities.

1 Introduction

The idea of enriching P systems with probabilities and using a probabilistic or stochastic evolution appears very early in the development of the area [7, 6]. Such kind of models (we shall call them probabilistic P systems) were shown to be very useful for simulations of biological phenomena, we cite here only [3, 1, 10]. While some of these models are using the Gillespie stochastic simulation algorithm (SSA) [4, 5] for the evolution step, the others are introducing different approximations of it or choose a completely different strategy. The definitions used to define the models often use specific notions and terminology, so their comparison and understanding quickly becomes a difficult task.

In this note we propose a systematical approach to the definition of such systems based on the association of a probability to a group of rules, which is a natural generalization of a probability for a single rule. The used method permits to establish a framework that can be used to compare existing definitions and gives a possibility to extend them. As an example of the application of the method we translate the definition of the evolution step of two variants of probabilistic P systems into our framework and we show the equivalent strategies of computation of individual rule probabilities leading to a corresponding group probability.

2 Preliminaries

We do not present here standard definitions. We refer to [12] for all details. We will denote by $|M|$ the cardinality of a set M or the size of a multiset M. By $|M|_x$ we will denote the number of elements x in the multiset M.

We also assume that the reader is familiar with standard notions of P systems, which can be consulted in the books [8] and [9] or at the web page [11]. We shall only focus on the semantics of the evolution step. We will follow the approach given in [2], however we will not enter into deep details concerning the notation and the definition of derivation modes given there. Consider a P system Π of any type evolving in any derivation mode. The key point of the semantics of P systems is that according to the type of the system and the derivation mode δ for any configuration of the system C a set of multisets (over $\mathcal{R}$) of applicable rules, denoted by $Appl(\Pi, C, \delta)$, is computed. After that, one of the elements from this set is chosen, non-deterministically, for the further evolution of the system.

We remark that from the point of view of the computer simulation of P systems the non-deterministic choice can be considered equivalent to a probabilistic choice where each multiset of rules has an equal probability to be chosen. Permitting these multisets to have a *different probability* is the main idea of this paper. More precisely, for each multiset of rules $R \in Appl(\Pi, C, \delta)$ we compute the probability $p(R, C)$ based on the propensity function $f : \mathcal{R}^\circ \times (\mathbb{N} \times O^\circ)^* \to \mathbb{R}$ that associates a real value for a multiset of rules with respect to a configuration. Hence the value $f(R, C)$ depends not only on the multiset of rules R, but also on the configuration C.

The probability to choose a multiset $R \in Appl(\Pi, C, \delta)$ is defined as the normalization of corresponding probabilities:

$$p(R, C) = \frac{f(R, C)}{\sum_{R' \in Appl(\Pi, C, \delta)} f(R', C)} \tag{1}$$

3 Discussion

In the previous section we didn't discuss the propensity function f, which is the main ingredient of the model. Below we will give examples of simple propensity functions each leading to different execution strategies.

Constant strategy: each rule r from $\mathcal{R}$ has a constant contribution to f and equal to c_r:

$$f(R, C) = \prod_{r \in R} c_r \tag{2}$$

Multiplicity-dependent strategy: each rule r from $\mathcal{R}$ has a contribution to f proportional to the number of times this rule can be applied and to a stochastic constant c_r that only depends on r:

$$N_r(C) = \min_{x \in lhs(r)} \left[\frac{|C|_x}{|lhs(r)|_x} \right] \tag{3}$$

$$f(R, C) = \prod_{r \in R} c_r N_r(C) \tag{4}$$

Concentration-dependent strategy: each rule r from $\mathcal{R}$ has a contribution to f proportional to $h_r(C)$, the number of distinct combinations of objects from C that activate r, and to a stochastic constant c_r that only depends on r (below we denote by $\binom{a}{b}$ the binomial function):

$$h_r(C) = \prod_{x \in lhs(r)} \binom{|C|_x}{|lhs(r)|_x} \tag{5}$$

$$h_R(C) = \prod_{r \in R} c_r h_r(C) \tag{6}$$

$$f(R, C) = h_R(C) \tag{7}$$

Gillespie strategy: each rule r from $\mathcal{R}$ has a contribution to f that depends on the order in which it was chosen and it is equal to $c_r \cdot h_r(C')$, where C' is the configuration obtained by applying all rules that were chosen before r.

We remark that the concentration-dependent strategy is not equal to Gillespie strategy. More precisely, in a Gillespie run the probability to choose a new rule depends on the objects consumed and produced by previously chosen rules. We can consider a Gillespie run as a sequence of sequential (single-rule) applications using concentration-dependent strategy.

We also remark that the Gillespie algorithm uses the notion of time that we do not consider in this paper. However, the definitions can be easily adapted for to handle this case.

4 Examples

4.1 Dynamical Probabilistic P Systems

Dynamical probabilistic P (DPP) systems were introduced in [10]. We present below the definition of the evolution step. For the sake of the simplicity we will consider only one compartment, however the discussion below can be easily generalized to several compartments.

Let C be the current configuration and $\mathcal{R}$ be the set of all rules. Then the system evolves from C to C' as follows.

1. For each rule $r \in \mathcal{R}$ the propensity of $a_r(C) = c_r * h_r(C)$ (h_r being defined as in Equation (5)) is computed.

2. The propensities are normalized giving a probability for a rule r to be chosen: $p_r(C) = \frac{a_r(c)}{\sum_{r' \in \mathcal{R}} a_{r'}(C)}$.
3. The rules to be applied are chosen according to their probabilities. If a non-applicable rule is chosen, the choice is repeated.
4. The process stops when a maximal (parallel) multiset of rules R is obtained.
5. The multiset of rules obtained at the previous step is applied and yields a new configuration C'.

It can be easily seen that since the probabilities to apply a rule (p_r) are computed only at the beginning of each step, then the maximal multiset of rules R is composed from independent rules (the order in which the rules were chosen has no influence). Hence the probability to choose a multiset of rules R is equal to the product of the probabilities of each rule: $p_R(C) = \prod_{r \in R} p_r$. Now if we normalize this value with respect to all possible maximally parallel multisets of rules we obtain:

$$\frac{\prod_{r \in R} p_r(C)}{\sum_{R' \in Appl(\Pi, C, max)} \prod_{z \in R'} p_z(C)} = \frac{\prod_{r \in R} \frac{a_r(C)}{\sum_{r' \in \mathcal{R}} a_{r'}(C)}}{\sum_{R' \in Appl(\Pi, C, max)} \prod_{z \in R'} \frac{p_z(C)}{\sum_{r' \in \mathcal{R}} a_{r'}(C)}} = \frac{\prod_{r \in R} a_r(C)}{\sum_{R' \in Appl(\Pi, C, max)} \prod_{z \in R'} a_z(C)} \tag{8}$$

Since for the concentration-dependent strategy we have $f(R, C) = \prod_{r \in R} a_r(C)$, it follows that (8) equals to (1). Hence we just showed that DPP systems can be translated to probabilistic P systems with a concentration-dependent strategy.

4.2 Probabilistic Functional Extended P Systems

Probabilistic functional extended P (PFEP) systems where introduced in [1] as a part of a framework used to model eco-systems. In order to simplify the presentation we consider a flattening of the structure of the P system, hence using only multiset rewriting rules. We also consider that the rules having the same left-hand side form a partition of the set of rules $\mathcal{R}$ into n subsets $\mathcal{R} = \mathcal{R}_1 \dots \mathcal{R}_n$, where $r_1, r_2 \in \mathcal{R}_i \Rightarrow lhs(r_1) = lhs(r_2)$, $1 \leq i \leq n$.

The evolution of a PFEP system is done as follows:

1. A maximally parallel multiset of rules R is chosen.
2. R is partitioned into submultisets based on the left-hand side of rules: $R_i = \{r \in R \mid r \in \mathcal{R}_i\}$.
3. For each non-empty partition R_i, $|R_i|$ rules from $\mathcal{R}_i$ are chosen according to the given probability $f_r(a)$, where $r \in R_i$ and a is a moment of time.
4. The resulting multiset of rules is applied yielding a new configuration.

From the description of the strategy it is clear that it corresponds to the multiplicity-dependent strategy for a maximally parallel derivation mode (and where the constant c_r is replaced by $f_r(a)$).

5 Final Remarks

In this note we presented a new method to describe P systems working with probabilities. The main aim of this method is to provide a common framework permitting to describe variants of probabilistic P systems. Such a framework could be useful for the comparison of different variants and for the extension of existing ones.

The used method is based on the assignment of a probability to a multiset of applicable rules (according to some derivation mode). The evolution step then chooses a multiset of rules according to its probability and then applies it. We were particularly interested by the cases when the probability of a multiset of rules R can be represented as a product of individual probabilities of rules $r \in R$. We gave example of three strategies for the computation of the individual probabilities of a rule. The first strategy supposes that the rule probability is constant, the second strategy supposes that the rule probability is proportional to the number of its applications, while the third strategy corresponds to the mass action law. We showed that the DPP systems from [10] are using the third strategy, while the PFEP systems from [1] the second. An interesting direction for the further research is to consider the above strategies in combination with different derivation modes.

We remark that the obtained strategies are not equivalent to the Gillespie stochastic simulation algorithm (SSA), except if the sequential derivation mode is used, because they do not take into account the intermediate modifications of the configuration. In some sense they correspond to the tau-leaping method, which is an approximation of the Gillespie SSA. An interesting topic for a further research could be the expression of different Gillespie-based strategies in the proposed framework. This can give rise to new variants of P systems suitable for stochastic simulations.

References

1. 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. *Natural Computing*, **10**(1): 39–53, 2011.
2. R. Freund, S. Verlan: A formal framework for static (tissue) P systems. In *Proc. of WMC 2008* (G. Eleftherakis et al., eds.), Thessaloniki, Greece, Springer, 2007, LNCS 4860, 271–284.
3. M. Gheorghe, V. Manca, F. J. Romero-Campero, Deterministic and stochastic P systems for modelling cellular processes. *Natural Computing* **9**(2): 457–473, 2010.
4. D.T. Gillespie, A general method for numerically simulating the stochastic time evolution of coupled chemical reactions, J. Comput. Phys. 22 (4)(1976) 403-434.
5. D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.
6. M. Madhu, Probabilistic rewriting P systems. *International Journal of Foundations of Computer Science*, **14**(1): 157–166, 2003.

7. A. Obtulowicz, Gh. Păun. (In search of) Probabilistic P systems, *BioSystems*, **70**(2): 107–121, 2003.
8. Gh. Păun, *Membrane Computing. An Introduction*. Springer–Verlag, 2002.
9. G. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook Of Membrane Computing*. Oxford University Press, 2009.
10. D. Pescini, D. Besozzi, G. Mauri, and C. Zandron. Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, **17**:183–204, 2006.
11. *The Membrane Computing Web Page*: `http://ppage.psystems.eu`
12. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*. Springer–Verlag, Berlin, 1997.

Adaptive Fuzzy Spiking Neural P Systems for Fuzzy Inference and Learning

Jun Wang[1] and Hong Peng[2]

[1] School of Electrical and Information Engineering,
Xihua University, Chengdu, Sichuan, 610039, China
[2] School of Mathematics and Computer Engineering,
Xihua University, Chengdu, Sichuan, 610039, China
wangjun@mail.xhu.edu.cn

Summary. Spiking neural P systems (in short, SN P systems) and their variants, including fuzzy spiking neural P systems (in short, FSN P systems), generally lack learning ability so far. Aiming at this problem, a class of modified FSN P systems are proposed in this paper, called adaptive fuzzy spiking neural P systems (in short, AFSN P systems). The AFSN P systems not only can model weighted fuzzy production rules in fuzzy knowledge base but also can perform dynamically fuzzy reasoning. It is more important that the AFSN P systems have learning ability like neural networks. Based on neuron's firing mechanisms, a fuzzy reasoning algorithm and a learning algorithm are developed. An example is included to illustrate the learning ability of the AFSN P systems.

Key words: Spiking neural P systems, Fuzzy spiking neural P systems, Adaptive fuzzy spiking neural P systems, Fuzzy reasoning, Learning pronlem

1 Introduction

Spiking neural P systems (in short, SN P systems) firstly introduced by Ionescu et al. in 2006 [1], are a class of distributed parallel computing models, which are incorporated into membrane computing from the way that biological neurons communicate through electrical impulses of identical form (spikes) [2]. Since then, a large number of SN P systems and their variants have been proposed [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. From the viewpoint of real-world applications, SN P systems have attractive due to the following features: (i) parallel computing advantage, (ii) high understandability (due to their directed graph structure), (iii) dynamic feature (neurons firing and spiking mechanisms make them suitable to model dynamic behaviors of a system), (iv) synchronization (that makes them suitable to describe concurrent events or activities), (v) non-linearly (that makes them suitable to process non-linear situation), and so on. Recently, in order to take full the advantage of SN P systems, a class of extended SN P systems were

proposed by introducing fuzzy logic, which were called fuzzy spiking neural P systems (in short, FSN P systems) [12, 13, 14, 15]. The motivation of proposing these FSN P systems is to deal with the representation of fuzzy knowledge and model fuzzy reasoning in some real-world applications, such as process control, expert system, fault diagnosing, etc. As we know, since knowledge in real-world applications mentioned above is frequently updated, they are essentially dynamic systems. This requires that the FSN P systems should be adaptive, that at, FSN P systems must have ability to adjust themselves. However, the FSN P systems might fails to cope with potential changes of actual systems due to their lack of adaptive or learning mechanism. Besides, a few of adaptive SN P systems have been addressed in recent years [16, 17].

In this paper, we propose a class of modified FSN P systems, which are called adaptive fuzzy spiking neural P systems (in short, AFSN P systems). The practical motivation is to build a novel way to deal with the learning problem of dynamical fuzzy knowledge in some real-world applications under the framework of SN P systems. For this purpose, based on neuron's firing mechanisms, a fuzzy reasoning algorithm and a learning algorithm are developed in this paper.

The rest of this paper is organized as follows. In Section 2, we firstly present the AFSN P systems, and then describe a way to model weighted fuzzy production rules by the AFSN P systems, finally move on to give the developed fuzzy reasoning algorithm and learning algorithm. Simulation example is provided in Section 3. Finally, Section 4 draws the conclusions.

2 AFSN P Systems

2.1 Definition of AFSN P Systems

Currently, fuzzy spiking neural P systems (FSN P Systems, in short) have been discussed [12, 13, 14, 15]. However, they can not adjust themselves and lack learning ability. In this paper, we will introduce "adaptive" mechanism into the FSN P systems to propose a class of adaptive FSN P systems, called AFSN P systems.

Definition 1. *An AFSN P systems (*of degree $m \geq 1$*) is a construct of the form*

$$\Pi = (A, N_p, N_r, syn, I, O)$$

where

1) A=$\{a\}$ *is the singleton alphabet (the object a is called spike);*
2) $N_p = \{\sigma_{p1}, \sigma_{p2}, \ldots, \sigma_{pm}\}$ *is called proposition neuron set, where* σ_{pi} *is its i-th proposition neuron associated with a fuzzy proposition in weighted fuzzy production rules,* $1 \leq i \leq m$*. Each proposition neuron* σ_{pi} *has the form* $\sigma_{pi} = (\alpha_i, \boldsymbol{\omega}_i, \lambda_i, r_i)$*, where:*
   a) $\alpha_i \in [0, 1]$ *and it is called the (potential) value of pulse contained in proposition neuron* σ_{pi}*.* α_i *is used to express fuzzy truth value of the proposition associated with proposition neuron* σ_{pi}*.*

b) $\boldsymbol{\omega}_i = (\omega_{i1}, \omega_{i2}, \ldots, \omega_{is_i})$ *is called the output weight vector of the neuron* σ_{pi}*, where component* $\omega_{ij} \in [0,1]$ *is the weight on* j*-th output synapse (arc) of the neuron,* $1 \leq j \leq s_i$*, and* s_i *is the number of all output synapses (arc) of the neuron.*

c) r_i *is a firing/spiking rule, of the form* $E/a^{\alpha} \to a^{\alpha}$*, where* $\alpha \in [0,1]$*.* $E = \{\alpha \geq \lambda_i\}$ *is called the firing condition, i.e., if* $\alpha \geq \lambda_i$*, then the firing rule will be enabled, where* $\lambda_i \in [0,1)$ *is called the firing threshold.*

3) $N_r = \{\sigma_{r1}, \sigma_{r2}, \ldots, \sigma_{rn}\}$ *is called rule neuron set, where* σ_{ri} *is its* i*-th rule neuron associated with a weighted fuzzy production rule,* $1 \leq i \leq n$*. Each rule neuron* σ_{ri} *has the form* $\sigma_{ri} = (\alpha_i, \gamma_i, \tau_i, r_i)$*, where*

a) $\alpha_i \in [0,1]$ *is called the* (potential) value *of pulse contained in rule neuron* σ_{ri}*.*

b) $\gamma_i \in [0,1]$ *is called the certain factor. It represents the strength of belief of the weighted fuzzy production rule associated with rule neuron* σ_{ri}*. At the same time,* γ_i *is also the weight on output synapse (arc) of the neuron.*

c) r_i *is a firing/spiking rule, of the form* $E/a^{\alpha} \to a^{\beta}$*, where* $\alpha, \beta \in [0,1]$*.* $E = \{\alpha \geq \tau_i\}$ *is called the firing condition, i.e., if* $\alpha \geq \tau_i$*, then the firing rule will be enabled, where* $\tau_i \in [0,1)$ *is called the firing threshold.*

4) $syn \subseteq (N_p \times N_r) \bigcup (N_r \times N_p)$ *indicates synapses between both proposition neurons and rule neurons. Note that there are no synapse connections between any two proposition neurons or between any two rule neurons;*

5) $I, O \subseteq N_p$ *are input neuron set and output neuron set, respectively.*

In the AFSN P systems, there are two types of neurons: proposition neurons and rule neurons. In this paper, we denote proposition neurons and rule neurons by circles and rectangles respectively, shown in Fig. 1.

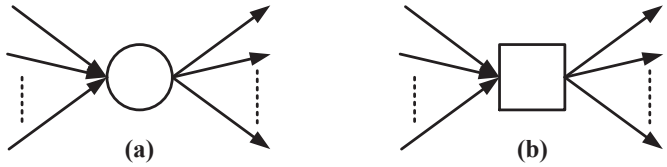


Fig. 1. Two types of neurons: (a) a proposition neuron; (b) a rule neuron.

For a proposition neuron, its content is used to express the fuzzy truth value of the fuzzy proposition associated with it. When its firing condition $E = \{\alpha \geq \lambda_i\}$ is satisfied, the neuron fires and its firing/spiking rule $E/a^{\alpha} \to a^{\alpha}$ can be applied. Applying the firing/spiking rule $E/a^{\alpha} \to a^{\alpha}$ means that the spike contained in the neuron is consumed, and then it produces a spike with value α, which will be weighted by the corresponding weight factor. Thus, its outputs are $\alpha \cdot \omega_i (i = 1, 2, \ldots, s)$.

Note that each rule neuron is assigned only an output weight ν. Suppose that a rule neuron has k predecessor proposition neurons. When it receives k spikes from its all predecessor proposition neurons and its firing condition $E = \{\alpha \geq \tau_i\}$ is satisfied, then it fires and its firing/spiking rule $E/a^{\alpha} \rightarrow a^{\beta}$ can be applied. The value of the received k spikes is calculated as its content α: $\alpha = x_1 + x_2 + \ldots + x_k$. Applying the firing/spiking rule $E/a^{\alpha} \rightarrow a^{\beta}$ means that the spike contained in the neuron is consumed, and then it produces a spike with value β where $\beta = \alpha \cdot \gamma$. Thus, its all outputs are $\alpha \cdot \gamma$.

Suppose that a proposition neuron has k predecessor rule neurons and it receives k spikes from them. Let output weights of the k predecessor rule neurons be $\gamma_1, \gamma_2, \ldots, \gamma_k$ respectively. If (potential) values of the received k spikes are $x_1, x_2, \ldots, x_k$ respectively, then its new content is computed by $\alpha = (x_1 + x_2 + \ldots + x_k)/(\gamma_1 + \gamma_2 + \ldots + \gamma_k)$.

2.2 Modeling Weighted Fuzzy Production Rules by AFSN P Systems

In many real-world applications such as expert system, fault diagnosing and process control, fuzzy production rules are used to describe the fuzzy relation between two propositions. In order to consider the degree of importance of each proposition in the antecedent contributing to the consequent, weighted fuzzy production rule has been introduced, and a more detained description can be found in [18, 19, 20].

However, we will discuss the following three types of weighted fuzzy production rules in order to study AFSN P systems in this paper.

Type 1: A simple fuzzy production rule

$$R: \text{ IF } p_1 \text{ THEN } p_2 \ (CF = \gamma), \ \tau, \ \omega$$

Type 2: A composite conjunctive rule

$$R: \text{ IF } p_1 \text{ AND } p_2 \text{ AND } \cdots \text{ AND } p_n \text{ THEN } p_{n+1} \ (CF = \gamma), \ \tau, \ \omega_1, \ \omega_2, \ldots, \omega_n$$

Type 3: A composite disjunctive rule

$$R: \text{ IF } p_1 \text{ OR } p_2 \text{ OR } \cdots \text{ OR } p_n \text{ THEN } p_{n+1} \ (CF = \gamma), \ \tau, \ \omega_1, \ \omega_2, \ldots, \omega_n$$

Above three types of weighted fuzzy production rules can be modeled by the proposed AFSN P systems according to the idea that each fuzzy proposition is mapped into one proposition neuron and each fuzzy production rule is mapped into one rule neuron or several rule neurons. Thus, the three types of weighted fuzzy production rules are represented by the following three AFSN P systems, Π_1, Π_2 and Π_3, respectively:

- $\Pi_1 = (A, \{\sigma_{p1}, \sigma_{p2}\}, \{\sigma_{r1}\}, syn, I, O)$ where:
  (1) $A = \{a\}$
  (2) For each j $(j = 1, 2)$, $\sigma_{pj} = (\alpha_j, \boldsymbol{\omega}, \lambda, r_j)$ is a proposition neuron associated with proposition p_j, and r_j is a spiking rule of the form $E/a^{\alpha} \rightarrow a^{\alpha}$.

(3) $\sigma_{r1} = (\alpha_3, \gamma, \tau, r_3)$ is a rule neuron associated with rule R, and r_3 is a spiking rule of the form $E/a^{\alpha} \to a^{\beta}$.
(4) $syn = \{(\sigma_{p1}, \sigma_{r1}), (\sigma_{r1}, \sigma_{p2})\}$.
(5) $I = \{\sigma_{p1}\}$, $O = \{\sigma_{p2}\}$.

Fig. 2(a) shows the AFSN P system model of *Type 1*: Π_1.

- $\Pi_2 = (A, \{\sigma_{p1}, \sigma_{p2}, \ldots, \sigma_{pn}, \sigma_{p(n+1)}\}, \{\sigma_{r1}\}, syn, I, O)$ where:
(1) $A = \{a\}$
(2) For each j $(j = 1, \ldots, n, n+1)$, $\sigma_{pj} = (\alpha_j, \boldsymbol{\omega}_j, \lambda_j, r_j)$ is a proposition neuron associated with proposition p_j, and r_j is a spiking rule of the form $E/a^{\alpha} \to a^{\alpha}$.
(3) $\sigma_{r1} = (\alpha_{n+2}, \gamma, \tau, r_{n+2})$ is a rule neuron associated with rule R, and r_{n+2} is a spiking rule of the form $E/a^{\alpha} \to a^{\beta}$.
(4) $syn = \{(\sigma_{p1}, \sigma_{r1}), (\sigma_{p2}, \sigma_{r1}), \ldots, (\sigma_{pn}, \sigma_{r1}), (\sigma_{r1}, \sigma_{p(n+1)})\}$.
(5) $I = \{\sigma_{p1}, \sigma_{p2}, \ldots, \sigma_{pn}\}$, $O = \{\sigma_{p(n+1)}\}$.

Fig. 2(b) shows the AFSN P system model of *Type 2*: Π_2 (in the case of $n = 2$).

- $\Pi_3 = (A, \{\sigma_{p1}, \sigma_{p2}, \ldots, \sigma_{pn}, \sigma_{p(n+1)}\}, \{\sigma_{r1}, \sigma_{r2}, \ldots, \sigma_{rn}\}, syn, I, O)$ where:
(1) $A = \{a\}$
(2) For each j $(j = 1, \ldots, n, n+1)$, $\sigma_j = (\alpha_j, \boldsymbol{\omega}_j, \lambda_j, r_j)$ is a proposition neuron associated with proposition p_j, and r_j is a spiking rule of the form $E/a^{\alpha} \to a^{\alpha}$.
(3) For each j $(j = 1, \ldots, n)$, $\sigma_{rj} = (\alpha_{n+j+1}, \gamma_j, \tau_j, r_{n+j+1})$ is a rule neuron associated with rule R, and r_{n+j+1} is a spiking rule of the form $E/a^{\alpha} \to a^{\beta}$.
(4) $syn = \{(\sigma_{p1}, \sigma_{r1}), (\sigma_{p2}, \sigma_{r2}), \ldots, (\sigma_{pn}, \sigma_{rn}), (\sigma_{r1}, \sigma_{p(n+1)}), (\sigma_{r2}, \sigma_{p(n+1)}),$ $\ldots, (\sigma_{rn}, \sigma_{p(n+1)})\}$.
(5) $I = \{\sigma_{p1}, \sigma_{p2}, \ldots, \sigma_{pn}\}$, $O = \{\sigma_{p(n+1)}\}$.

Fig. 2(c) shows the AFSN P system model of *Type 3*: Π_3 (in the case of $n = 2$).

2.3 Fuzzy Reasoning Based on AFSN P systems

Since the presented AFSN P systems mainly focus on the weighted fuzzy reasoning, we assume that firing threshold of every proposition neuron is $\lambda = 0$. This means that once a proposition neuron contains a spike with $\alpha > 0$ it will fires. According to firing mechanism of AFSN P systems, fuzzy reasoning processes of above three types of weighted fuzzy production rules can be described as follows:

- For *Type 1*, we can set $\omega = 1$ since there is only one proposition in antecedent of the rule R. Initially, assume that neuron σ_{p1} contains a spike with $\alpha_1 > 0$. At first step, neuron σ_{p1} fires and emits a spike with α_1. At second step, neuron σ_{r1} receives the spike. If $\alpha_1 \geq \tau$, then neuron σ_{r1} fires and emits a spike with

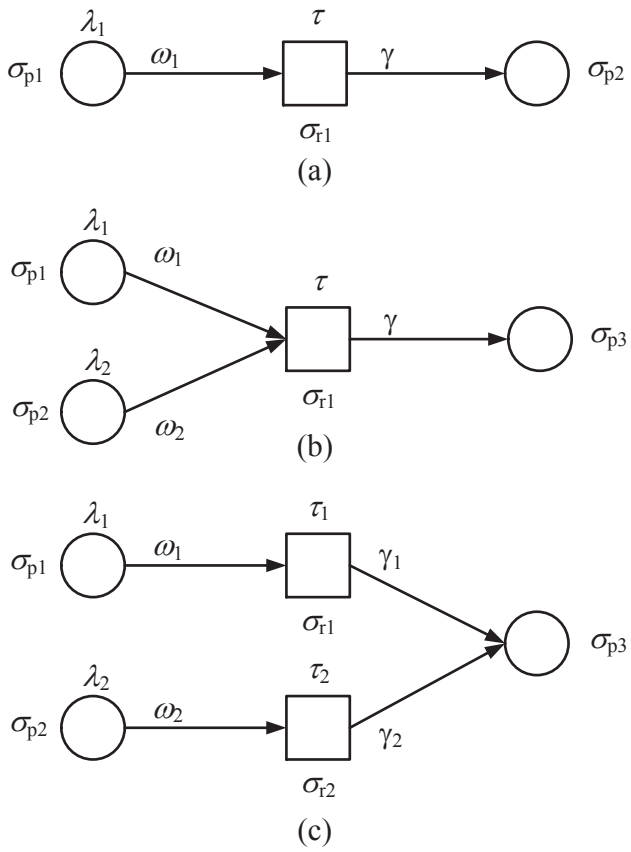


Fig. 2. AFSN P systems of weighted fuzzy production rules of three types: (a) *Type 1*; (b) *Type 2*; (c) *Type 3*.

$\alpha_1 \cdot \gamma$. Neuron σ_{p2} will receive the spike at next step. Thus, α_2 can be expressed by

$$\alpha_2 = \begin{cases} \alpha_1 \cdot \gamma, & \text{if } \alpha_1 \geq \tau \\ 0, & \text{if } \alpha_1 < \tau \end{cases} \tag{1}$$

- For *Type 2*, assume that neurons $\sigma_{p1}, \sigma_{p2}, \ldots, \sigma_{pn}$ contain a spike with $\alpha_1 > 0, \alpha_2 > 0, \ldots, \alpha_n > 0$, respectively. At first step, the n neuron fire simultaneously, and emit a spike with $\alpha_1, \alpha_2, \ldots, \alpha_n$, respectively. At second step, neuron σ_{r1} receives the n spikes and its content is updated as $\sum_{i=1}^{n} \alpha_i \cdot \omega_i$. If $(\sum_{i=1}^{n} \alpha_i \cdot \omega_i) \geq \tau$, then neuron σ_{r1} fires and emits a spike with $(\sum_{i=1}^{n} \alpha_i \cdot \omega_i) \cdot \gamma$. Neuron $\sigma_{p(n+1)}$ will receive the spike at next step. Thus, α_{n+1} can be expressed by

$$\alpha_{n+1} = \begin{cases} \Big(\sum\limits_{i=1}^{n} \alpha_i \cdot \omega_i\Big) \cdot \gamma, \text{ if } \Big(\sum\limits_{i=1}^{n} \alpha_i \cdot \omega_i\Big) \geq \tau \\ 0, \qquad\qquad\qquad \text{if } \Big(\sum\limits_{i=1}^{n} \alpha_i \cdot \omega_i\Big) < \tau \end{cases} \tag{2}$$

- For *Type 3*, we can set $\omega_1 = \omega_2 = 1$. Assume that neurons $\sigma_{p1}, \sigma_{p2}, \ldots, \sigma_{pn}$ contain a spike with $\alpha_1 > 0, \alpha_2 > 0, \ldots, \alpha_n > 0$, respectively. At first step, the n neuron fire simultaneously, and emit a spike with $\alpha_1, \alpha_2, \ldots, \alpha_n$, respectively. At second step, each neuron σ_{ri} receives a spike sent by σ_{pi}, whose value is α_i, $i = 1, 2, \ldots, n$. Let $J = \{j \mid \alpha_j \geq \tau_j, j = 1, 2, \ldots, n\}$. Then neurons $\sigma_{rj} (j \in J)$ fire and each neuron of them emits a spike. Neuron $\sigma_{p(n+1)}$ will receive the spikes at next step. Thus, α_{n+1} can be expressed by

$$\alpha_{n+1} = \begin{cases} \Big(\sum\limits_{j\in J} \alpha_j \cdot \gamma_j\Big) \Big/ \Big(\sum\limits_{j\in J} \gamma_j\Big), \text{ if } \alpha_j \geq \tau_j, j \in J \\ 0, \qquad\qquad\qquad\qquad \text{if } \alpha_j < \tau_j, j = 1, 2 \ldots, n \end{cases} \tag{3}$$

From fuzzy reasoning process described above, we can see that fuzzy reasoning based on AFSN P systems are easily implemented. Thus, through firing mechanism of AFSN P systems, certainty factors can be reasoned from a set of known antecedent propositions to a set of consequent propositions step by step.

Let $P_{current} = \{\sigma_{pi} \mid \sigma_{pi} \in N_p, \alpha_i > 0\}$ be a set of current enabled proposition neurons. If a neuron $\sigma_{pi} \in P_{current}$, then it will fire. Let $R_{current} = \{\sigma_{rj} \mid \sigma_{rj} \in N_r, \alpha_j > \tau_j\}$ be a set of current enabled rule neurons. Likewise, if a neuron $\sigma_{rj} \in R_{current}$, then it will fire. Therefore, fuzzy reasoning algorithm based on AFSN P systems can be summarized as follows.

```
program Fuzzy_reasoning_algorithm
input
   Certainty factors of a set of antecedent propositions, which
      are corresponding to I of AFSN P systems;
output
   Certainty factors of a set of consequence propositions, which
      are corresponding to O of AFSN P systems;
begin
   Pcurrent := I;
   Rcurrent := {}
   P := Np;
   R := Nr;
   repeat
      Compute the outputs of current enabled proposition
         neurons in Pcurrent;
      Find current enabled rule neurons Rcurrent form R;
      Compute the outputs of current enabled proposition
         neurons in Rcurrent;
      P := P - Pcurrent;
```

```
        R := R - Rcurrent;
        Find current enabled proposition neurons Pcurrent form P;
    until P = {} and R = {}
end.
```

2.4 Learning of AFSN P systems

In order to deal with the learning problem of AFSN P systems, we assume that

1) AFSN P system model Π has been developed;
2) In the AFSN P system model, weights and thresholds of all rule neurons are known;
3) Certainty factor values of all neurons in I and O are given.

From the discussion above, we know that the presented AFSN P systems are mainly used to model weighted fuzzy production rules and these rules consist of three types. So, an AFSN P system model can be divided into three types of sub-structures, which are shown in Fig.2(a)-(c). Therefore, the learning of entire system can be decomposed to several simpler learning procedures of the sub-nets. This means that the complexity of the learning algorithm can be greatly reduced. According to above assumption, certainty factors of the proposition neurons associated with antecedent propositions are known, however, their weights are unknown. Therefore, these weights need to be learned. Note that for AFSN P system Π_1 of *Type 1*, we have $\omega_1 = 1$, while we have $\omega_1 = \omega_2 = 1$ for AFSN P system Π_3 of *Type 3*. So, only weights of AFSN P system Π_2 of *Type 2* need to be learned. In order to carry out the weight learning, the AFSN P system Π_2 of *Type 2* can be converted to a single-layer neural network, shown in Fig.3. So, Widrow-Hoff learning law (Least Mean Square) can be applied in this paper.

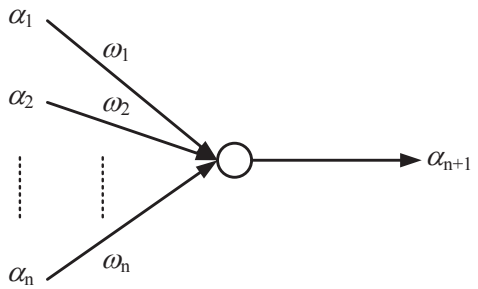


Fig. 3. The single-layer neural network converted by the AFSN P system Π_2 of *Type 2*.

We can summarize the learning algorithm of AFSN P systems as follows

```
program Weight_learning_algorithm
input
   Training data set D;
   m = |D|;
   Learning rate delta;
output
   The weights (w1, w2,..., wn);
begin
   Select a set of initial weights;
   i=1;
   repeat
      Compute the outputs error of i-th training sample;
      Update the weights (w1, w2,..., wn) using Widrow-Hoff
         learning law with learning rate delta;
      i = i + 1
   until i>m
end.
```

3 Simulation

In this section, a typical example is selected to illustrate the learning ability.

Example 1. Let p_1, p_2, p_3, p_4, p_5 and p_6 are related propositions of a knowledge base of fault diagnosis. There are the following weighted fuzzy production rules:
R_1: IF p_1 THEN p_4 (γ_1, τ_1)
R_2: IF p_2 AND p_4 THEN p_5 (ω_2, ω_4, γ_2, τ_2)
R_3: IF p_3 AND p_5 THEN p_6 (γ_3, γ_4, τ_3, τ_4)

This example includes three types of rules: R_1 is a simple rule and R_2 is a composite conjunctive rule, while R_3 is a composite disjunctive rule. These weighted fuzzy production rules can be modeled by the following AFSN P system Π:

- $\Pi = (A, \{\sigma_{p1}, \sigma_{p2}, \sigma_{p3}, \sigma_{p4}, \sigma_{p5}, \sigma_{p6}\}, \{\sigma_{r1}, \sigma_{r2}, \sigma_{r3}, \sigma_{r4}\}, syn, I, O)$
  where:
  (1) $A = \{a\}$
  (2) For each j ($j = 1, 2, 3, 4, 5, 6$), $\sigma_{pj} = (\alpha_j, \boldsymbol{\omega}_j, \lambda_j, r_j)$ is a proposition neuron associated with proposition p_j, and r_j is a spiking rule of the form $E/a^\alpha \rightarrow a^\alpha$. Here, $\lambda_j (j = 1, 2, \ldots, 6) = 0$, and $\omega_1 = \omega_3 = \omega_5 = 1$.
  (3) For each j ($j = 1, 2, 3, 4$), $\sigma_{rj} = (\alpha_{k+j}, \gamma_j, \tau_j, r_{k+j})$ is rule neuron. σ_{r1} and σ_{r2} are associated with rule R_1 and R_2 respectively, while σ_{r3} and σ_{r4} are associated with rule R_3. $r_{k+j} (j = 1, 2, 3, 4)$ are spiking rule of the form $E/a^\alpha \rightarrow a^\beta$.
  (4) $syn = \{(\sigma_{p1}, \sigma_{r1}), (\sigma_{p2}, \sigma_{r2}), (\sigma_{p3}, \sigma_{r3}), (\sigma_{p4}, \sigma_{r2}), (\sigma_{p5}, \sigma_{r4}), (\sigma_{r1}, \sigma_{p4}), (\sigma_{r2}, \sigma_{p5}), (\sigma_{r3}, \sigma_{p6}), (\sigma_{r4}, \sigma_{p6})\}$.
  (5) $I = \{\sigma_{p1}, \sigma_{p2}, \sigma_{p3}\}$, $O = \{\sigma_{p4}, \sigma_{p5}, \sigma_{p6}\}$.

Fig.4 shows the AFSN P system Π. The AFSN P system has three input proposition neurons $\{\sigma_{p1}, \sigma_{p2}, \sigma_{p3}\}$ and three output proposition neurons $\{\sigma_{p4}, \sigma_{p5}, \sigma_{p6}\}$. Suppose parameters of the AFSN P system are given as follows:

$$\begin{array}{l} \gamma_1 = 0.80,\ \gamma_2 = 0.85,\ \gamma_3 = 0.85,\ \gamma_4 = 0.90 \\ \tau_1 = 0.40,\ \tau_2 = 0.60,\ \tau_3 = 0.55,\ \tau_4 = 0.45 \end{array} \tag{4}$$

Here, weights ω_2 and ω_4 are unknown. Assume the ideal weights are $\omega_2^* = 0.63$ and $\omega_4^* = 0.37$. Using fuzzy reasoning algorithm, we can obtain a set of output data (certainty factors of consequence propositions) according to the input data (certainty factors of antecedent propositions). Table 1 gives the part of the reasoning results of the AFSN P system.

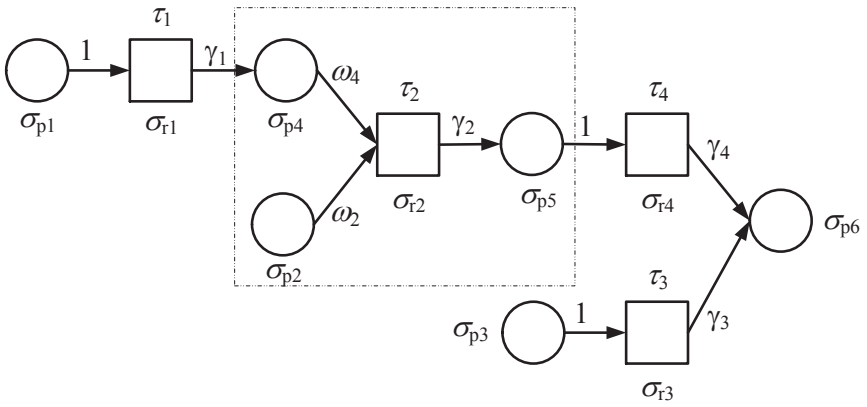


Fig. 4. AFSN P system of *Example* 1.

Table 1. The reasoning results of AFSN P systems.

No.	α_1	α_2	α_3	α_4	α_5	α_6
1	0.8762	0.7724	0.8536	0.7010	0.6341	0.7470
2	0.8325	0.8271	0.6124	0.6660	0.6524	0.6365
3	0.7518	0.8912	0.5896	0.6390	0.6782	0.6326
4	0.6785	0.7216	0.6518	0.5767	0.5678	0.6110
5	0.6127	0.6874	0.7829	0.5208	0.5319	0.6610
6	0.5866	0.8516	0.5908	0.4986	0.6128	0.6015
7	0.5236	0.7835	0.5862	0.4451	0.5595	0.5732
8	0.3645	0.7845	0.6628	0.0	0.0	0.3409
9	0.5235	0.5648	0.7461	0.4450	0.0	0.3837
10	0.3246	0.6324	0.5582	0.0	0.0	0.2871
...	...	...	...	...	...	...

From Fig.4, we can see that only two weights ω_2 and ω_4 need to be learned in the AFSN P system Π. In this paper, neural network technique will be employed to adjust the two weights. The learning part of the AFSN P system Π (see the part in the dashed box of Fig.4) can be transformed as a single layer neural network (see Fig.5):

$y(t) = W(t)^T X(t) + b$

where t is time, $X(t) = [\alpha_2(t), \alpha_4(t)]^T$ is input vector, $W(t) = [\omega_2(t), \omega_4(t)]^T$ is weights vector, and b is the bias.

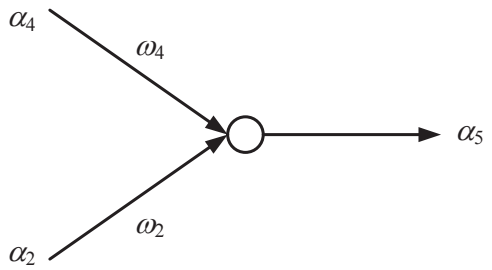


Fig. 5. The neural network transformed by the learning part in the AFSN P system of *Example* 1.

In order to learn these weights by using neural networks, Widrow-Hoff learning law can be applied as follows

$$W(t+1) = W(t) + 2\delta e(t) X(t), \tag{5}$$

$$e(t) = y^*(t) - y(t) \tag{6}$$

Here, we select $\delta = 1.23$. Let initial weights be $W(0) = [\omega_2(0), \omega_4(0)]^T = [0.5, 0.2]^T$. By applying Widrow-Hoff learning law, after a training process ($t > 33$), the two weights convergence to their real values. Fig.6 shows simulation results.

Form the example, we can see that the fuzzy reasoning algorithm and the Widrow-Hoff learning are very effective if we do not know the weights of AFSN P systems. After a training process, we can build a good input-output mapping relation of a knowledge system.

4 Conclusion

In this paper, we presented a class of modified fuzzy spiking neural P systems: adaptive fuzzy spiking neural P systems (AFSN P systems, in short). In addition to fuzzy knowledge representation and dynamically fuzzy reasoning, they have learning ability as neural netwarks. Therefore, fuzzy knowledge in knowledge base not only can be modeled by a AFSN P system but also can be learning through the

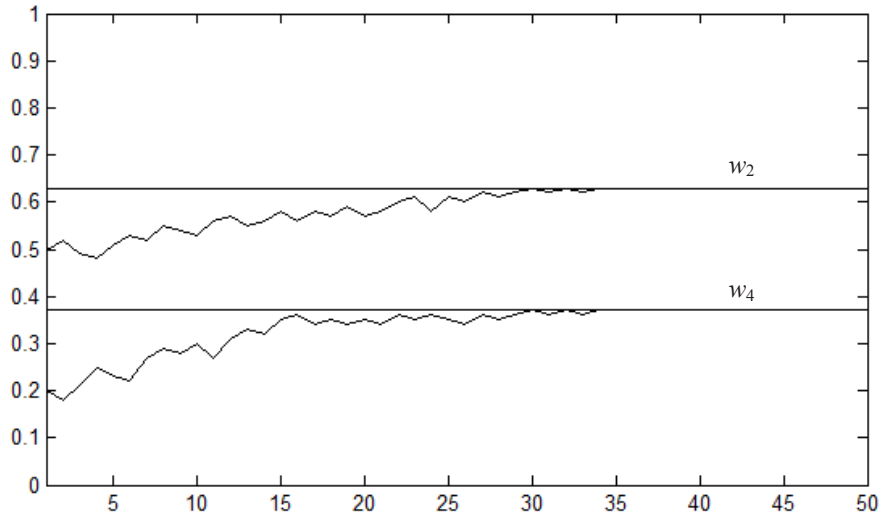


Fig. 6. The weight learning results of *Example* 1.

AFSN P system. The results presented in this paper provide a novel way to solve the knowledge learning problem in some real-world applications, such as expert systems, fault diagnosis, process control, and so on.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (Grant No. 61170030), Research Fund of Sichuan Provincial Key Discipline of Power Electronics and Electric Drive, Xihua University (No. SZD0503-09-0), Research Fund of Sichuan Key Laboratory of Intelligent Network Information Processing (No. SGXZD1002-10), and the Importance Project Foundation of Xihua University (No. Z1122632), China.

References

1. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking Neural P Systems. Fundameta Informaticae 71(2-3), 279–308 (2006)
2. Păun, Gh., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrance Computing. Oxford Unversity Press, New York (2010)
3. Păun, Gh., Pérez-Jiménez, M.J., Rozenberg, G.: Spike Train in Spiking Neural P Systems. Int. J. Found. Comp. Sci. 17(4), 975–1002 (2006)

4. Chen, H., Ishdorj, T.-O., Păun, Gh., Perez-Jimenez, M.J.: Handling Languages with Spiking Neural P Systems with Extended Rules. Romanian Journal of Information Science and Technology 9(3), 151–162 (2006)
5. Chen, H., Ishdorj, T.-O., and Gh. Păun, Computing Along The Axon. Progress in Natural Science 17(4), 417–423 (2007)
6. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking Neural P Systems with An Exhaustive Use of Rules. Int. J. of Unconvent. Comt. 3(2), 135–154 (2007)
7. Freund, R., Ionescu, M., Oswald, M.: Extended Spiking Neural P Systems with Decaying Spikes and/or Total Spiking. Int. J. of Foundations of Computer Science 19(5), 1223–1234 (2008)
8. Pan, L., Păun, G.: Spiking Neural P Systems with Anti-Spikes. Int. J. of Computers, Communications & Control, 4(3), 273-282 (2009)
9. Wang, J., Hoogeboom, H.J., Pan, L., Păun, Gh.: Spiking Neural P Systems with Weights and Thresholds. In: Proceedings of Tenth Workshop on Membrane Computing (WMC10), August 2009, pp. 514-533 (2009)
10. Cavaliere, M. Ibarra, O.H., Păun, G., Egecioglu, O., Ionescu, M., Woodworth, S. Asynchronous Spiking Neural P Systems. Theoretical Computer Science 410(24-25) 2352–2364 (2009)
11. Pan, L., Păun, G.: Spiking Neural P Systems: An Improved Normal Form. Theoretical Computer Science 411(6) 906–918 (2010)
12. Wang, J., Peng, H.: Fuzzy Knowledge Representation Based on An Improving Spiking Neural P System. In: 2010 Sixth International Conference of Natural Computing, ICNC2010, 6, 3012–3015 (2010)
13. Wang, T., Wang, J. Peng, H. Deng, Y.L.: Knowledge Representation Using Fuzzy Spiking Neural P System. In: 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications, Volume-1, 586–590 (2010)
14. Wang, J., Zhou, L., Peng, H., Zhang, G.X.: An Extended Spiking Neural P System for Fuzzy Knowledge Representation. International Journal of Innovative Computing, Information and Control, 7(7A), 3709–3724 (2011)
15. Peng, H., Wang, J., Perez-Jimenez, M.J., Wang, H., Shao, J., Wang, T.: Fuzzy Reasoning Spiking Neural P System for Fault Diagnosis. Information Sciences, 2012 (Accepted)
16. Gutierrez-Naranjo, M.A., Perez-Jimenez, M.J.: Hebbian Learning from Spiking Neural P Systems View. Lecture Notes in Computer Science, Volume 5391/2009, 217–230 (2009)
17. Peng, H., Wang, J.: Adaptive Spiking Neural P Systems. In: 2010 Sixth International Conference of Natural Computing, ICNC2010, 6, 3008–3011 (2010)
18. Yeung, D.S., Tsang, E.C.C.: Weighted Fuzzy Production Rules. Fuzzy Sets and Systems, 88, 299–313 (1997)
19. Yeung, D.S., Tsang, E.C.C.: A Multilevel Weighted Fuzzy Reasoning Algorithm for Expert Systems. IEEE Trans. Syst., Man, Cybern. A, 28(2), 149–158 (1998)
20. Chen, S.-M., Ko, Y.-K., Chang, Y.-C., Pan J.-S.: Weighted Fuzzy Interpolative Reasoning Based on Weighted Increment Transformations and Weighted Ratio Transformation Techniques. IEEE Transactions on Fuzzy Systems, 17(6), 1412–1427 (2009)

Modelling Intelligent Energy Distribution Systems by Hyperdag P Systems

Adrian Zafiu[1], Cristian Ştefan[2]

[1] IMT Bucharest, Romania
[2] University of Piteşti, Romania
{adrian.zafiu,cristi.stefan}@upit.ro

Summary. The paper introduces a new model in membrane computing, using the hyperdag P systems to simulate a complex, feedback-driven energy distribution system. The proposed model is tested within an ad-hoc developed simulator, and the evolution of the system is presented step by step.

1 Introduction

The P systems are a computational model inspired from cellular biology, introduced by Păun [10] in 1998 in order to simulate the behaviour of natural systems by means of formal specifications.

Membrane computing is a vast research field, involving contributions from different areas, like parallel and distributed systems, financial case studies and evolution of living cells populations. There are many types of P systems, like tissue P systems, neural (spiking) P systems or asynchronous P systems. The model was further examined in Păun et al. [11].

The hyperdag P systems are a refinement of the original model, in which the tree structure is replaced by a directed acyclic graph (dag), introduced by Nicolescu [8] in 2008.

P system models allow realistic simulations of evolving systems, as transition rules can be applied separately for each cell, taking into account the environment factors (represented as promoters or inhibitors) and the received information from other cells (transported symbols).

2 Energy distribution systems (EDS) - a case study

When we consider an energy distribution system (EDS), there are two approaches. One involves the big-scale entities, like power plants, the national network of transformers and transmission lines, and finally the consumers (industrial-grade and home).

The small-scale approach regards the self-powered home, that has its own generators using renewable energy sources like the sun (photovoltaic panels) and the wind (eolian turbines).

The main goal in designing an ecological household regards the control of energy consumption level and the ways to optimize it.

In order to bypass short-time fluctuations that this kind of generators can suffer due to sudden changes in the environment factors, the system makes use of a set of batteries, that store the energy when it is available and give it back instantly if ecological power falls for a short period.

To summarize, the EDS (figure 1) has the following components:

- connection to the grid
- grid controller
- batteries
- battery regulator
- generators
- generator regulator
- the consumers
- the consumer controller (inverter)
- sensors
- sensor monitor
- memory for comfort variables
- main control unit (MCU)

As the natural factors change all the time, one cannot rely solely on independent generators to supply all the necessary power to a household for everyday needs. Thus, the connection to the national power grid is mandatory. To control how much power is taken from the grid and to monitor the costs involved, a grid controller is taken into account.

The set of batteries acts both as a buffer in case of short outages (temporary lack of wind or sunlight), and as an affordable alternative source for low-power requirements (night lighting, standby current for different devices), where grid energy can be avoided. Batteries charge only when there is enough *green power*, in order to keep the costs as low as possible. Their cycle is regulated by a dedicated controller to prevent overcharging or over-discharging, both being equally dangerous for the internal chemistry.

The generator set is the core of a self-powered home system, transforming the freely available energy from the natural sources, like sun and wind, into usable electrical power to drive all the devices that surround us and make our life easier. Using such energy implies reduced costs, long-term sustainability and a reduced impact on the planet's resources. The controller to which they are attached to is used for monitoring their usage, reporting failures and disconnecting them when power requirements indicate there's no need for more, in order to protect the life of moving components (turbine).

The consumers are all the electrical appliances that the owner makes use of but, for the case study in this paper, only the lighting and air conditioning systems

are considered to be monitored and adjusted according to the desired parameters. Their controller has a function in conversion also, as the supplied DC voltage (usually 12 to 48V) from batteries and generators must be raised and converted to AC before it can be used.

The sensors read the instantaneous values (available light level and temperature) from the environment, and report them through their monitor to the MCU. They play the key role in the feed-back mechanism.

The desired values for the comfort variables (temperature and amount of light) that the user sets are stored in a dedicated memory that the MCU will read each time it needs to make an adjustment.

The MCU is the brain of the system, containing all necessary logic (rules) to request data from the memory and sensor controller, calculate the difference between values and issue the appropriate commands for the generators and consumers to adjust their behaviour as required. It is connected directly with all other controllers and the memory, as all communication between them passes through it.

3 Intelligent Energy distribution systems

3.1 The model structure and logic

The *intelligence* involved in the distribution is achieved through the rules implemented by the MCU, with the declared goal of minimizing the consumption from the grid. When the parameters need to be adjusted upwards, the first source considered are always the generators, as their energy comes almost for free (after the investment has been recovered). If they cannot supply the necessary instantaneous power, the second choice are the batteries, as they have an amount of power that comes also at no cost. If they are empty or have already reached the maximum that they can offer, there is no other option than to take the rest from the grid. This is the costly solution, but sometimes it's the only one left. When renewable power is available again, the first to be satisfied are the consumers, followed by the batteries who need to be refilled.

The philosophy behind such a system is to react promptly to the changes that occur and to satisfy the current needs without wasting energy when there's no one home, or at night, when there's usually no need for powerful lighting. If the temperature in the house is already at the desired level, the air conditioning system will not be started and, if the desired level has been reached after an increased consumption, the controller will just keep with it, without other increases.

3.2 Architecture - Hyperdag P systems

As mentioned in the Introduction, hyperdag P systems are a new family of P systems that R. Nicolescu proposed as an alternative to other existent types (tissue

and neural P systems) in order to offer a more flexible way to communicate between cells, but respecting the hierarchical structure. In this approach the messages can be passed also to the cells on the same level (siblings), rewriting rules can be applied in a deterministic or parallel way, and the transfer modes can be dedicated (a single receiver) or spread across a domain (broadcast). The efficiency of such P systems has been proven by modelling problems like Synchronization in P Modules [2], the Byzantine Agreement [1] and optimizations to FSSP [4].

The basic definitions and notions from graph theory will not be discussed again here, as they can be found very easily in the literature.

The definition of hyperdag P systems and the two extensions are the ones given in Part A of the technical report by R. Nicolescu [7], [8], [9].

Definition 1. *A* hP system *(of degree m) is a system $\Pi = (O, \sigma_1, \ldots, \sigma_m, \delta, I_{out})$, where:*

1. *O is an ordered finite non-empty alphabet of* objects*;*
2. *$\sigma_1, \ldots, \sigma_n$ are* cells*, of the form $\sigma_i = (Q_i, s_{i0}, w_{i0}, P_i), 1 \leq i \leq m$, where:*
   - *Q_i is a finite set (of* states*),*
   - *$s_{i0} \in Q_i$ is the* initial state*,*
   - *$w_{i0} \in O^*$ is the* initial multiset *of objects,*
   - *P_i is a finite set of multiset rewriting* rules *of the form $sx \to s'x'u_{\uparrow}v_{\downarrow}w_{\leftrightarrow}y_{go}z_{out}$, where $s, s' \in Q_i, x, x' \in O^*, u_{\uparrow} \in O^*_{\uparrow}, v_{\downarrow} \in O^*_{\downarrow}, w_{\leftrightarrow} \in O^*_{\leftrightarrow}, y_{go} \in O^*_{go}$, and $z_{out} \in O^*_{out}$ with the restriction that $z_{out} = \lambda$ for all $i \in \{1, \ldots, m\} \setminus I_{out}$;*
3. *δ is a set of dag parent/child arcs on $\{1, \ldots, m\}$, i.e., $\delta \subseteq \{1, \ldots, m\}x\{1, \ldots, m\}$, representing* bidirectional *communication channels between the cells;*
4. *$I_{out} \subseteq \{1, \ldots, m\}$ indicates the* output cells*, the only cells allowed to send objects to the "environment".*

In addition to this definition, there are two more elements that should be presented in order to fully describe the simulation mechanism. One is the object transfer mode and the other is rewriting mode for symbols. Both define how rules are applied.

Regarding the object transfer mode, there are three options:

- *replication*: the replicated symbols are transmitted to all parents (↑), all children (↓) or all siblings (↔);
- *one*: the object will be delivered to a single, randomly chosen, parent (↑), child (↓) or sibling (↔);
- *spread*: the multiset will be decomposed and the parts are to be sent arbitrarily to the parents (↑), children (↓) or siblings (↔).

Regarding the symbol rewriting mode, there are also three options:

- *min*: the rule is applied once, if possible;
- *par*: rule is applied in parallel manner for all available symbols;
- *max*: a rule is applied as many times as possible.

It is important to mention that rules are applied in *weak priority* order, meaning that the ones with higher priority (appear at the beginning) come first, and that lower priority rules are applied only if they **do not** change the target state reached from the previous rules.

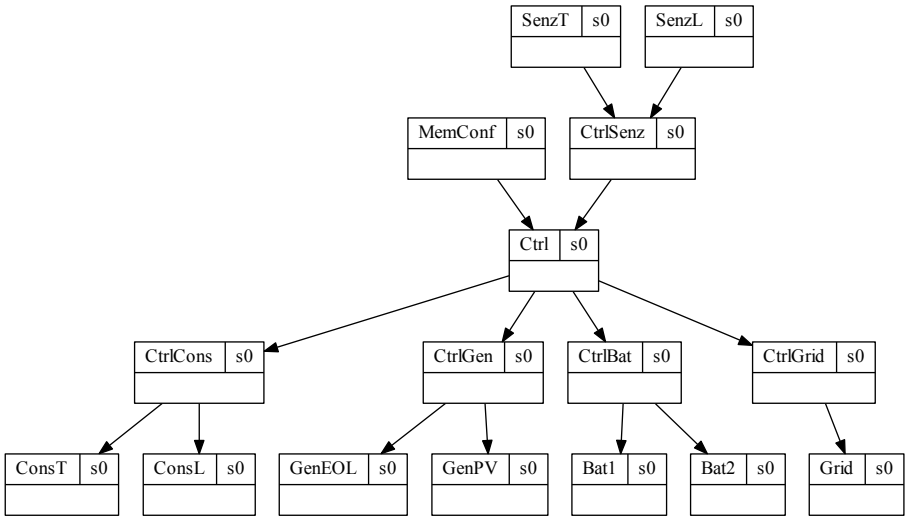


Fig. 1. General view of the system

3.3 Algorithm - The rule set and types

At the beginning, we define the connexions between the cells, by mentioning the parent and all its children. The initial cell configurations follow *@ lines*, as we define the memory for comfort variables *MemConf*, current u and maximum m energy values for each entity. Each consumer has a maximum amount of power that it can take, each generator has a limit of what it can give. The sensors store the values read from the environment. The first step is to send the trigger command q from the main control unit *Ctrl* to the memory and the sensors, in order to ask them to reply with their content. Sensors report to their dedicated controller *CtrlSenz*, which then sends the information to the main *Ctrl*.

After receiving all data, the controller is able to make the decision to increase or decrease the amount of energy offered by the generators, by calculating the difference between the desired temperature (or light) level - stored in *MemConf*, and the current one, reported by the sensors. The confirmation of energy availability

will be sent to the consumers (temperature, lights), and they will increase their current consumption by one unit at each step.

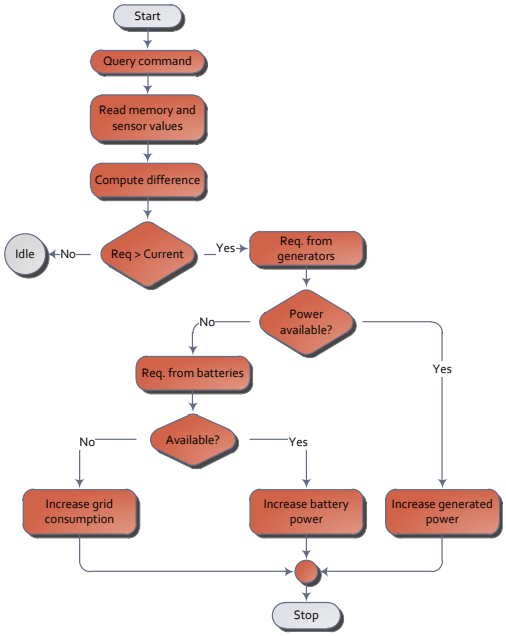


Fig. 2. Logical scheme

4 The simulator overview

The simulator implements the hyperdag P systems in respect with **Def. 1**, object rewriting rules and object transfer modes. At its core we defined the digraph structure, with arcs, nodes, rules, states and symbols as components, grouped into Configurations. We also implemented the *Rewriting* and *Transition types* as described above.

The direction for symbol transfer is indicated within the Behaviour class, all communication channels being considered as bi-directional. There are four options:

- *down*: symbols are sent to direct children of the current node;
- *up*: symbols are sent to the directly connected parent(s);
- *sibling*: objects reach the nodes on the same level and which are connected with the emitting cell;

- *out*: this is the production of the P system calculation, symbols are sent from the Output cell to the environment, and their multiplicity is regarded as the final result.

Rules form a separate class, each one having defined the initial and final states, the priority, rewriting and transition types. Rules are defined as strings entered by the user, as one would usually describe them, in the following form:

* **cell** *init.state* `<sym._mult.>` `->` *f.state* `<sym._mult._dir.rewr.transf.>`

As an example, we present a rule for the *MCU*, which will propagate *down* the commands to increase the light and the power from the Grid, without changing the current state *sqc*. The rule is applied as many times as possible, and symbols are replicated to all the children:

$* \, Ctrl \; sqc \; ALu \; Su \rightarrow sqc \; ALu_{\downarrow} \; Su_{\downarrow} \; max \; repl$

Before entering the rules (marked by *), one needs to define the cells in the system, their connections and their initial states (lines beginning with @). When the command *Create* is given, the parser reads each line, builds the dag structure and the graphical representation on the fly (using *GraphViz* [3]) and loads each cell with its rule set. From that point, the system can evolve fully in one step (*Run* command) or step by step, with the currently applied rules being showed in red, for easier understanding of the transitions, and the content of each cell being updated in real time. Execution uses the parallel features of the .NET platform, cells that can evolve simultaneously have dedicated threads for their computations.

The simulator will be available for download in the near future at the following address: `http://fmi.upit.ro/psim/`.

5 Description of the rule set

In this section we present all the rules, grouped in subsections by each cell, and explain their roles in the system.

5.1 The memory for the comfort variables

For the cell *MemConf*, there are the following rules:

1. $s_0 \; q \; \rightarrow \; sqt \; min \; rep$
2. $sqt \; t \; \rightarrow \; sql \; t \; MT_{\downarrow} \; max \; rep$
3. $sqt \; \rightarrow \; sql \; min \; rep$
4. $sql \; l \; \rightarrow \; sqa \; l \; ML_{\downarrow} \; max \; rep$
5. $sql \; \rightarrow \; sqa \; min \; rep$
6. $sqa \; \rightarrow \; s_0 \; a_{\downarrow} \; max \; rep$

The meaning of the symbols are detailed in Table 1.

5.2 Main Control Unit

The *Ctrl* cell analyses and regulates the functioning of the entire system, and thus it has an increased number of rules and symbols. It communicates with all other Controllers in a full cycle. The states for this cell are, as follows:

Symbol	Description
q	The query request
t	The desired temperature value, stored in memory
l	The desired light level, stored in memory
MT	The response symbol for temperature
ML	The response symbol for light

Table 1. MemConf cell symbols

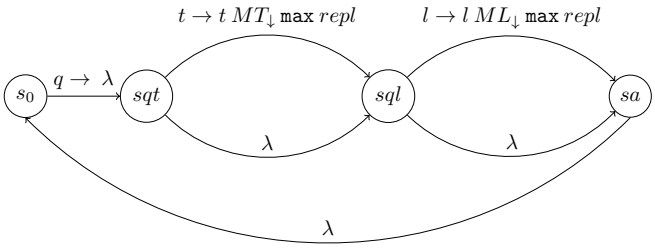


Fig. 3. State diagram for MemConf

s_0: initial state;
sq: waiting for a system query;
sqa: the analysis phase;
sqc: computing the base regulation (phase I), computing the commands for the full regulation (phase II) and cleaning the unnecessary symbols.

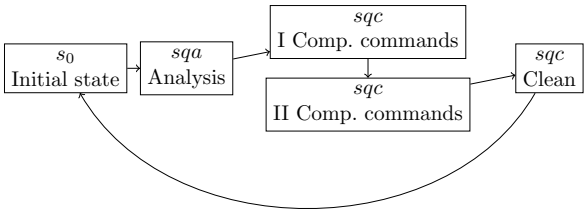


Fig. 4. State diagram for Ctrl

The role of the symbols are described in Table 2.

Symbol	Description
a	An answer
MT	The temperature value stored in memory
ST	The system temperature
CTu	Temperature can be increased
CTd	Temperature can be decreased
ATu	The computing answer is to increase the temperature
ATd	The computing answer is to decrease the temperature
ML	The light level stored in memory
SL	the light measured by the sensor
CLu	Light can be in increased
CLd	Light can be decreased
ALu	The computing answer is to increase the light
ALd	The computing answer is to decrease the light
GEu	The eolian generator has available power
GEd	The eolian generator cannot increase power
GPu	The photovoltaic generator has available power
GPd	The photovoltaic generator cannot increase power
B1u	Battery 1 can be charged
B1d	Battery 1 can be used for power
B2u	The battery 2 can be charged
B2d	The battery 2 can be used for power
Su	The Grid can offer more power
Sd	The Grid cannot offer more power

Table 2. Ctrl cell symbols

Request rules:

1. $s_0 \rightarrow sq\, q_{\updownarrow}\, min\, rep$
2. $sq\, a_6 \rightarrow sqa\, min\, rep$

The first step when starting the system is to send a query command (q) to all components. After receiving six answers (a symbols), the *Ctrl* has all necessary informations and it can start computing the differences and adjust the parameters by sending the appropriate commands to the other controllers.

The state analysis:

1. $sqa\, MT\, ST \rightarrow sqa\, max\, rep$
2. $sqa\, MT\, CTu \rightarrow sqa\, ATu\, min\, rep$
3. $sqa\, ST\, CTd \rightarrow sqa\, ATd\, min\, rep$
4. $sqa\, MT \rightarrow sqa\, max\, rep$
5. $sqa\, ST \rightarrow sqa\, max\, rep$
6. $sqa\, CTu \rightarrow sqa\, min\, rep$
7. $sqa\, CTd \rightarrow sqa\, min\, rep$
8. $sqa\, ML\, SL \rightarrow sqa\, max\, rep$
9. $sqa\, ML\, CLu \rightarrow sqa\, ALu\, min\, rep$
10. $sqa\, SL\, CLd \rightarrow sqa\, ALd\, min\, rep$
11. $sqa\, ML \rightarrow sqa\, max\, rep$
12. $sqa\, SL \rightarrow sqa\, max\, rep$
13. $sqa\, CLu \rightarrow sqa\, min\, rep$
14. $sqa\, CLd \rightarrow sqa\, min\, rep$

Rule #1 computes the difference between the desired Temperature value that is stored in Memory (MT) and the current one, read by the Sensor (ST). If there

are MT symbols left and it is possible to increase the consumption for the heating device (CTu is present), the command is issued by creating the ATu symbol. If there are ST symbols present and it is possible to reduce the consumption (CTd is present), then we produce the command to decrease the Temperature, ATd. Rules 4 to 7 clean the remaining symbols.

Rule #8 makes the difference between the desired Light level form the Memory (ML) and the current one from the Sensor (SL). If there are ML symbols present and it is possible to increase the power for lighting (CLu is present), then we produce the command ALu with rule #9. If there are SL symbols left, and it is possible to reduce the power for lighting (presence of CLd), then symbol Ald is produced. Rules #11...#14 do the cleaning.

The first set of regulation rules is the following:

1. $sqa \rightarrow sqc\, min\, rep$
2. $sqc\, B1o\, Sd \rightarrow sqc\, B1o_{\downarrow}\, Sd_{\downarrow}\, max\, rep$
3. $sqc\, B2o\, Sd \rightarrow sqc\, B2o_{\downarrow}\, Sd_{\downarrow}\, max\, rep$
4. $sqc\, GEu\, Sd \rightarrow sqc\, GEu_{\downarrow}\, Sd_{\downarrow}\, max\, rep$
5. $sqc\, GPu\, Sd \rightarrow sqc\, GPu_{\downarrow}\, Sd_{\downarrow}\, max\, rep$
6. $sqc\, GEu\, B1i \rightarrow sqc\, GEu_{\downarrow}\, B1i_{\downarrow}\, max\, rep$
7. $sqc\, GPu\, B1i \rightarrow sqc\, GPu_{\downarrow}\, B1i_{\downarrow}\, max\, rep$
8. $sqc\, GEu\, B2i \rightarrow sqc\, GEu_{\downarrow}\, B2i_{\downarrow}\, max\, rep$
9. $sqc\, GPu\, B2i \rightarrow sqc\, GPu_{\downarrow}\, B2i_{\downarrow}\, max\, rep$

Rule #1 puts the $Ctrl$ cell in a state where potential anomalies are detected and removed (rules #2...#9).

2, 3: If the Batteries can give more power (presence of $B1o$ or $B2o$) and the consumption from the Grid can be decreased (presence of Sd), then the appropriate commands will be propagated down.

4, 5: If the Generators can give more power (GEu or GPu are present) and the consumption from the Grid can be decreased (Sd is there), then the commands are sent down.

6, 7, 8, 9: If the battery charge current can be increased (we have $B1i$ or $B2i$) and the Generators can offer more energy, the appropriate commands are sent to their controller.

The second set of regulation rules consists of:

1. $sqc\, ATu\, GEu \rightarrow sqc\, ATu_{\downarrow}\, GEu_{\downarrow}\, max\, rep$
2. $sqc\, ATu\, GPu \rightarrow sqc\, ATu_{\downarrow}\, GPu_{\downarrow}\, max\, rep$
3. $sqc\, ATu\, B1o \rightarrow sqc\, ATu_{\downarrow}\, B1o_{\downarrow}\, max\, rep$
4. $sqc\, ATu\, B2o \rightarrow sqc\, ATu_{\downarrow}\, B2o_{\downarrow}\, max\, rep$
5. $sqc\, ATu\, Su \rightarrow sqc\, ATu_{\downarrow}\, Su_{\downarrow}\, max\, rep$
6. $sqc\, ATd\, Sd \rightarrow sqc\, ATd_{\downarrow}\, Sd_{\downarrow}\, max\, rep$
7. $sqc\, ATd\, B1i \rightarrow sqc\, ATd_{\downarrow}\, B1i_{\downarrow}\, max\, rep$
8. $sqc\, ATd\, B2i \rightarrow sqc\, ATd_{\downarrow}\, B2i_{\downarrow}\, max\, rep$
9. $sqc\, ATd\, GEg \rightarrow sqc\, ATd_{\downarrow}\, GEd_{\downarrow}\, max\, rep$

10. $sqc\,ATd\,GPg \;\rightarrow\; sqc\,ATd_{\downarrow}\,GPd_{\downarrow}\,max\,rep$
11. $sqc\,ALu\,GEu \;\rightarrow\; sqc\,ALu_{\downarrow}\,GEu_{\downarrow}\,max\,rep$
12. $sqc\,ALu\,GPu \;\rightarrow\; sqc\,ALu_{\downarrow}\,GPu_{\downarrow}\,max\,rep$
13. $sqc\,ALu\,B1o \;\rightarrow\; sqc\,ALu_{\downarrow}\,B1o_{\downarrow}\,max\,rep$
14. $sqc\,ALu\,B2o \;\rightarrow\; sqc\,ALu_{\downarrow}\,B2o_{\downarrow}\,max\,rep$
15. $sqc\,ALu\,Su \;\rightarrow\; sqc\,ALu_{\downarrow}\,Su_{\downarrow}\,max\,rep$
16. $sqc\,ALd\,Sd \;\rightarrow\; sqc\,ALd_{\downarrow}\,Sd_{\downarrow}\,max\,rep$
17. $sqc\,ALd\,B1i \;\rightarrow\; sqc\,ALd_{\downarrow}\,B1i_{\downarrow}\,max\,rep$
18. $sqc\,ALd\,B2i \;\rightarrow\; sqc\,ALd_{\downarrow}\,B2i_{\downarrow}\,max\,rep$
19. $sqc\,ALd\,GEg \;\rightarrow\; sqc\,ALd_{\downarrow}\,GEd_{\downarrow}\,max\,rep$
20. $sqc\,ALd\,GPg \;\rightarrow\; sqc\,ALd_{\downarrow}\,GPd_{\downarrow}\,max\,rep$

These rules are in charge of the increase (ATu, ALu) and decrease commands (ATd, ALd) for the Temperature and Light levels.

1..5: We try to increase the consumption for the Temperature (ATu) by checking the available sources, in the following order: Eolian Generator (GEu), Photovoltaic Generator (GPu), and the Batteries ($B1o$, $B2o$) and finally, as a last resort, the national power Source (Su). If any one of those has available power, the appropriate commands are sent to it.

6..10: We try to decrease the consumption for the Temperature and take into account the sources in reverse order: grid (Sd), batteries ($B1d$, $B2d$) and the generators (GEd, GPd). If the amount taken form any of these sources can be decreased, the commands are to be sent accordingly.

10..15: We try to increase the consumption for the Light (ALu) by checking the available sources, in the following order: Eolian Generator (GEu), Photovoltaic Generator (GPu), and the Batteries ($B1o$, $B2o$) and finally, as a last resort, the national power Source (Su). If any one of those has available power, the appropriate commands are sent to it.

16..20: We try to decrease the consumption for the Light and take into account the sources in reverse order: grid (Sd), batteries ($B1d$, $B2d$) and the generators (GEd, GPd). If the amount taken form any of these sources can be decreased, the commands are to be sent accordingly.

We use the following rules for cleaning:

1. $sqc\,MT \;\rightarrow\; sqc\,max\,rep$
2. $sqc\,ST \;\rightarrow\; sqc\,max\,rep$
3. $sqc\,ML \;\rightarrow\; sqc\,max\,rep$
4. $sqc\,SL \;\rightarrow\; sqc\,max\,rep$
5. $sqc\,B1i \;\rightarrow\; sqc\,max\,rep$
6. $sqc\,B1o \;\rightarrow\; sqc\,max\,rep$
7. $sqc\,Su \;\rightarrow\; sqc\,max\,rep$
8. $sqc\,Sd \;\rightarrow\; sqc\,max\,rep$
9. $sqc\,GEu \;\rightarrow\; sqc\,max\,rep$
10. $sqc\,GEd \;\rightarrow\; sqc\,max\,rep$
11. $sqc\,GPu \;\rightarrow\; sqc\,max\,rep$
12. $sqc\,GPd \;\rightarrow\; sqc\,max\,rep$
13. $sqc\,B2i \;\rightarrow\; sqc\,max\,rep$
14. $sqc\,B2o \;\rightarrow\; sqc\,max\,rep$
15. $sqc\,CTu \;\rightarrow\; sqc\,max\,rep$
16. $sqc\,CTd \;\rightarrow\; sqc\,max\,rep$
17. $sqc\,CLu \;\rightarrow\; sqc\,max\,rep$
18. $sqc\,CLd \;\rightarrow\; sqc\,max\,rep$
19. $sqc \;\rightarrow\; s_0$

In the end, all unused symbols from this cell are cleared, rule #19 having the role to prepare the cell for a new computation cycle.

5.3 The sensor controller

For the cell *CtrlSens* we defined the following rules:

1. $s_0\ q\ \rightarrow\ s_0\ q_{\uparrow}\ min\ rep$
2. $s_0\ a_2\ \rightarrow\ sq\ a\ min\ rep$
3. $sq\ ST\ \rightarrow\ sq\ ST_{\downarrow}\ max\ rep$
4. $sq\ SL\ \rightarrow\ sq\ SL_{\downarrow}\ max\ rep$
5. $sq\ a\ \rightarrow\ s_0\ a_{\downarrow}\ min\ rep$

Description for these rules is given below:

1: the response request is forwarded to the sensors;
2: the cell waits for all sensors to answer;
3, 4: the measured values are relayed to the general controller;
5: cell confirms that all measured values have been submitted.

If the Temperature sensor receives a query, it will answer with an ST symbol for each t, and the Light sensor will answer with an SL symbol for each l it contains.

The description of the symbols is given in the table 3.

Symbol	Description
q	Query
t	The measured temperature
ST	The response symbol for temperature
l	The measured light
SL	The response symbol for light

Table 3. Sensor Controller symbols

5.4 Sensors

As announced, we have two sensors that measure the current temperature and amount of light from the environment we wish to monitor and control.

The rules for the Temperature sensor ($SensT$) are the following:

1. $s_0\ q\ \rightarrow\ sqt\ min\ rep$
2. $sqt\ t\ \rightarrow\ sqa\ t\ ST_{\downarrow}\ max\ rep$
3. $sqt\ \rightarrow\ sqa\ min\ rep$
4. $sqa\ \rightarrow\ s_0\ a_{\downarrow}\ min\ rep$

For the Light sensor *SensL* we have:

1. $s_0\, q \;\rightarrow\; sqt\, min\, rep$
2. $sql\, t \;\rightarrow\; sqa\, l\, SL_{\downarrow}\, max\, rep$
3. $sql \;\rightarrow\; sqa\, min\, rep$
4. $sqa \;\rightarrow\; s_0\, a_{\downarrow}\, min\, rep$

Again, they will answer with ST and SL for the queries, the same as their Controller, symbols having the same sense.

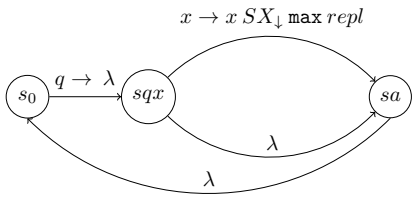


Fig. 5. State diagram for the Sensors

5.5 Consumers controller

The main property for the consumers is the availability to increase or decrease their current power absorbed. Let variable X represent the consumers. The request received by the consumer controller will be sent to all consumers. They will answer with CXu, meaning that the consumption for variable X (where $X \in \{Temp., Light\}$) can be increased, or with CXd, meaning that the consumption for that variable can be decreased. These symbols are further forwarded to the MCU, when answers from all consumers have been received.

The second phase regards treating the commands for actually increasing or decreasing the consumptions. These are forwarded to the consumers themselves for execution. Unnecessary symbols are then cleared.

Rules for the cell *CtrlCons* are the following:

1. $s_0\, q \;\rightarrow\; sq\, q_{\downarrow}\, min\, rep$
2. $sq\, a_2 \;\rightarrow\; sa\, min\, rep$
3. $sa\, CTu \;\rightarrow\; sa\, CTu_{\uparrow}\, min\, rep$
4. $sa\, CTd \;\rightarrow\; sa\, CTd_{\uparrow}\, min\, rep$
5. $sa\, CLu \;\rightarrow\; sa\, CLu_{\uparrow}\, min\, rep$
6. $sa\, CLd \;\rightarrow\; sa\, CLd_{\uparrow}\, min\, rep$
7. $sa \;\rightarrow\; s_0\, a_{\uparrow}\, min\, rep$
8. $s_0\, ATu \;\rightarrow\; s_0\, ATu_{\downarrow}\, max\, rep$
9. $s_0\, ATd \;\rightarrow\; s_0\, ATd_{\downarrow}\, max\, rep$
10. $s_0\, ALu \;\rightarrow\; s_0\, ALu_{\downarrow}\, max\, rep$
11. $s_0\, ALd \;\rightarrow\; s_0\, ALd_{\downarrow}\, max\, rep$
12. $s_0\, GEu \;\rightarrow\; s_0\, max\, rep$
13. $s_0\, GEd \;\rightarrow\; s_0\, max\, rep$
14. $s_0\, GPu \;\rightarrow\; s_0\, max\, rep$
15. $s_0\, GPd \;\rightarrow\; s_0\, max\, rep$
16. $s_0\, B1o \;\rightarrow\; s_0\, max\, rep$
17. $s_0\, B1i \;\rightarrow\; s_0\, max\, rep$
18. $s_0\, B2o \;\rightarrow\; s_0\, max\, rep$
19. $s_0\, B2i \;\rightarrow\; s_0\, max\, rep$
20. $s_0\, Su \;\rightarrow\; s_0\, max\, rep$
21. $s_0\, Sd \;\rightarrow\; s_0\, max\, rep$

Description for these rules is given below:

1: the request is transmitted to all consumers;
2: the cell waits for all consumers to answer;
3..6: the answers are relayed to the general controller;
7: the cell confirms that all answers have been submitted;
8..11: all commands are distributed to the consumers;
12..21: clean the unnecessary symbols.

The symbols' meanings are shown in Table 4.

Symbol	Description
q	Query
a	Counting the consumers answers
CTu	Temperature can be increased
CTd	Temperature can be decreased
ATu	The computing answer is to increase the temperature
ATd	The computing answer is to decrease the temperature
CLu	Light can be in increased
CLd	Light can be in decreased
ALu	The computing answer is to increase the light
ALd	The computing answer is to decrease the light

Table 4. CtrlCons cell symbols

5.6 Consumers

Each consumer can be asked to report it's actual state. The state consists of it's current consumption level (the multiplicity of the symbol u, if $u > 0$) and the availability to increase it (if $u < m$, m being the maximum value). The answer can be positive or negative. The second set of commands is about actually increasing the consumption, which is executed. Unknown commands are to be cleared.

The description is given for a generic consumer, indicated by X. The rules for a consumer cell are the following:

1. $s_0\, q \;\to\; sqd\, min\, rep$
2. $sqd\, u \;\to\; squ\, u\, ud\, min\, rep$
3. $sqd \;\to\; squ\, min\, rep$
4. $squ\, u\, m \;\to\; squ\, d\, max\, rep$
5. $squ\, m \;\to\; sqa\, m\, uu\, min\, rep$
6. $sqa\, d \;\to\; sqa\, u\, m\, max\, rep$
7. $sqa\, uu \;\to\; sqa\, CXCu_{\uparrow}\, min\, rep$
8. $sqa\, ud \;\to\; sqa\, CXd_{\uparrow}\, min\, rep$
9. $sqa \;\to\; s0\, a_{\uparrow}\, min\, rep$
10. $s0\, AXu \;\to\; s0\, u\, max\, rep$
11. $s0\, AXd\, u \;\to\; s0\, max\, rep$

The rules description is given below:

1: request to for the consumer state;
2..6: computing the difference between maximum and current values;

7, 8: the cell emits the answer;
9: cell confirms that answers were submitted;
10: the cell increases consumption;
11: the cell decreases consumption.

The symbols' meaning are shown in Table 5.

Symbol	Description
u	The consumption
m	The maximum value for the consumption
q	Query
a	Answer acknowledge
CXu	Consumption can be increased
CXd	Consumption can be decreased
AXu	Request to increase the consumption
AXd	Request to decrease power

Table 5. Consumer cell symbols

5.7 Generators controller

The generators have or not the ability to increase or decrease the amount of power they give at each moment. Let X be the generic name for a generator. The request received by the generator controller is further spread to all generators defined. They will answer with either *GXu*, meaning that they can increase the power, or *GXd*, if they can decrease their power. These symbols are to be delivered to the MCU when all generators have sent their answers.

The second phase of using a generator occurs when commands for increasing or decreasing the given power are actually received. These are forwarded to the generators, and the unnecessary symbols are to be cleaned up from their Controller.

The specific rules for the *CtrlGen* cell are as follows:

1. $s_0\, q \;\rightarrow\; sq\, q_{\downarrow}\, min\, rep$
2. $sq\, a_2 \;\rightarrow\; sa\, min\, rep$
3. $sa\, GEu \;\rightarrow\; sa\, GEu_{\uparrow}\, max\, rep$
4. $sa\, GEd \;\rightarrow\; sa\, GEd_{\uparrow}\, max\, rep$
5. $sa\, GPu \;\rightarrow\; sa\, GPu_{\uparrow}\, max\, rep$
6. $sa\, GPd \;\rightarrow\; sa\, GPd_{\uparrow}\, max\, rep$
7. $sa \;\rightarrow\; s_0\, a_{\uparrow}\, max\, rep$
8. $s_0\, GEu \;\rightarrow\; s_0\, GEu_{\downarrow}\, max\, rep$
9. $s_0\, GEd \;\rightarrow\; s_0\, GEd_{\downarrow}\, max\, rep$
10. $s_0\, GPu \;\rightarrow\; s_0\, GPu_{\downarrow}\, max\, rep$
11. $s_0\, GPd \;\rightarrow\; s_0\, GPd_{\downarrow}\, max\, rep$
12. $s_0\, ATu \;\rightarrow\; s_0\, max\, rep$
13. $s_0\, ATd \;\rightarrow\; s_0\, max\, rep$
14. $s_0\, ALu \;\rightarrow\; s_0\, max\, rep$
15. $s_0\, ALd \;\rightarrow\; s_0\, max\, rep$
16. $s_0\, B1o \;\rightarrow\; s_0\, max\, rep$
17. $s_0\, B1i \;\rightarrow\; s_0\, max\, rep$
18. $s_0\, B2o \;\rightarrow\; s_0\, max\, rep$
19. $s_0\, B2i \;\rightarrow\; s_0\, max\, rep$
20. $s_0\, Su \;\rightarrow\; s_0\, max\, rep$
21. $s_0\, Sd \;\rightarrow\; s_0\, max\, rep$

Description for these rules is given below:

1: the response request is forwarded to the generators;
2: the cell waits for all generators to answer;
3..6: the measured values are relayed to the general controller;
7: cell confirms that all measured values were submitted;
8..11: all commands are submitted to generators;
12..21: cleaning rules.

The symbols GXu and GXd play two roles:

1. if the cell is in the state sa, then the symbol indicating a generator state (G) is sent to the general controller;
2. if the cell is in state s_0, then the symbol designating a command forwarded to the each generator.

Unlike the consumers, symbols GXu and GXd have the multiplicity equal with the number of units that the power amount can be increased or decreased with.

The symbols' meaning are shown in Table 6.

Symbol	Description
q	Query
a	Answer acknowledge
GEu	Eolian generator can increase power
GEd	Eolian generator decrease power
GPu	Photovoltaic generator can offer more
GPd	Photovoltaic generator can offer less

Table 6. CtrlGen cell symbols

5.8 Generators

Each generator can be queried about it's current state. The state is about the actual power it gives, indicated by the multiplicity of the symbol u, and the availability to increase that power if the maximum value (m) has not been reached. The answer can be positive or negative.

The second mode for the generators occurs when they receive actual power increase or decrease commands, which are to be executed directly. Unnecessary commands need to be cleared.

The description is given for a generic generator, indicated by X:

The rules for a generator are the following:

1. $s_0\, q \;\rightarrow\; sqd\, min\, rep$
2. $sqd\, u \;\rightarrow\; squ\, u\, ud\, max\, rep$
3. $sqd \;\rightarrow\; squ\, min\, rep$
4. $squ\, u\, m \;\rightarrow\; squ\, d\, max\, rep$
5. $squ\, m \;\rightarrow\; sqa\, m\, uu\, max\, rep$
6. $sqa\, d \;\rightarrow\; sqa\, u\, m\, max\, rep$
7. $sqa\, uu \;\rightarrow\; sqa\, GXu_{\uparrow}\, max\, rep$
8. $sqa\, ud \;\rightarrow\; sqa\, GXd_{\uparrow}\, max\, rep$
9. $sqa \;\rightarrow\; s_0\, a_{\uparrow}\, min\, rep$
10. $s_0\, GXu \;\rightarrow\; s_0\, u\, max\, rep$
11. $s_0\, GXd\, u \;\rightarrow\; s_0\, max\, rep$

Rules description:

1: the request to report the consumer state;
2..6: computing the difference between maximum and current values;
7, 8: cell returns the answer;
9: the cell confirms that answers were submitted;
10: the cell increases generated power;
11: the cell decreases energy offered.

The symbols' meaning are shown in Table 7.

Symbol	Description
u	The actual power level
m	The maximum power
q	Query
a	Answer acknowledge
GXu	Request to increase the generated power
GXd	Request to decrease the generated power

Table 7. Generator cell symbols

5.9 Battery controller

Batteries are defined by the availability to increase or decrease the power they offer at each instant. The request to the *CtrlBat* is further disseminated to all batteries. They will answer each with *BXi* - the value with which the charge current can be increased, or *BXo* - the value with which the amount of power they give can be increased, where $Xin\{1 \ldots n\}$. These symbols are further relayed to the MCU when answers from all batteries have been received.

Rules for *CtrlBat* are as follows:

1. $s_0\, q \;\rightarrow\; sq\, q_{\downarrow}\, min\, rep$
2. $sq\, a_2 \;\rightarrow\; sa\, min\, rep$
3. $sa\, B1i \;\rightarrow\; sa\, B1i_{\uparrow}\, max\, rep$
4. $sa\, B1o \;\rightarrow\; sa\, B1o_{\uparrow}\, max\, rep$
5. $sa\, B2i \;\rightarrow\; sa\, B2i_{\uparrow}\, max\, rep$
6. $sa\, B2o \;\rightarrow\; sa\, B2o_{\uparrow}\, max\, rep$
7. $sa \;\rightarrow\; s_0\, a_{\uparrow}\, max\, rep$
8. $s_0\, B1o \;\rightarrow\; s_0\, B1o_{\downarrow}\, max\, rep$
9. $s_0\, B1i \;\rightarrow\; s_0\, B1i_{\downarrow}\, max\, rep$
10. $s_0\, B2o \;\rightarrow\; s_0\, B2o_{\downarrow}\, max\, rep$
11. $s_0\, B2i \;\rightarrow\; s_0\, B2i_{\downarrow}\, max\, rep$
12. $s_0\, ATu \;\rightarrow\; s_0\, max\, rep$

13. $s_0\, ATd \rightarrow s_0\, max\, rep$
14. $s_0\, ALu \rightarrow s_0\, max\, rep$
15. $s_0\, ALd \rightarrow s_0\, max\, rep$
16. $s_0\, GEu \rightarrow s_0\, max\, rep$
17. $s_0\, GEd \rightarrow s_0\, max\, rep$
18. $s_0\, GPu \rightarrow s_0\, max\, rep$
19. $s_0\, GPd \rightarrow s_0\, max\, rep$
20. $s_0\, Su \rightarrow s_0\, max\, rep$
21. $s_0\, Sd \rightarrow s_0\, max\, rep$

Rules description:

1: the request is retransmitted to the batteries;
2: cell waits for all batteries to answer;
3..6: measured values are relayed to the general controller;
7: cell confirms that all measured values were submitted;
8..11: all commands are submitted to batteries;
12..21: cleaning rules.

The symbols BXi and BXo play two roles:

1. if the cell is in the state sa, then the symbol indicating a battery state (B) is sent to the general controller;
2. if the cell is in state s_0, then the symbol designating a command forwarded to the each battery.

Unlike the consumers, symbols BXi and BXo have the multiplicity equal with the number of units that the power amount taken or given (for charging) can be increased or decreased with.

The symbols' meaning are shown in Table 8.

Symbol	Description
q	Query
a	Answer acknowledge
BXi	The battery X can give more energy
BXo	The battery X can charge more

Table 8. Battery controller symbols

5.10 Batteries

Each battery can be queried about its current state. The actual state refers to:

u, m: Energy level and maximum level;
iu, im: Charge level and maximum charge;
du, dm: Discharge level and maximum discharge.

A battery can have dual behaviour: can be a generator as it discharges, but becomes a consumer when it charges back. The two states differ by the maximum instantaneous values. Thus, the maximum amount with which the charging can be

done is given by the stopping of discharging and maximizing the charge current. Also, the maximum available power is given when the charge current is 0.

The second state for the batteries occurs when they receive commands to increase or decrease their given power amount. The command translates in changes for the values *iu* and *du*. The unnecessary commands are to be removed.

The description is given for a generic battery, indicated by X.

Rules for cell Battery X are:

1. $s_0\, iu\, du \;\rightarrow\; s_0\, max\, rep$
2. $s_0\, q \;\rightarrow\; sqd\, min\, rep$
3. $sqd\, u\, m \;\rightarrow\; sqd\, d\, max\, rep$
4. $sqd\, iu\, im \;\rightarrow\; sqd\, id\, BXo_{\uparrow}\, max\, rep$
5. $sqd\, du\, dm \;\rightarrow\; sqd\, dd\, BXi_{\uparrow}\, max\, rep$
6. $sqd\, m \;\rightarrow\; sqi\, m\, min\, rep$
7. $sqd\, d \;\rightarrow\; sqo\, d\, min\, rep$
8. $sqd \;\rightarrow\; sqa\, min\, rep$
9. $sqi\, im \;\rightarrow\; sqo\, im\, BXi_{\uparrow}\, max\, rep$
10. $sqi \;\rightarrow\; sqo\, min\, rep$
11. $sqo\, dm \;\rightarrow\; sqa\, dm\, BXo_{\uparrow}\, max\, rep$
12. $sqo \;\rightarrow\; sqa\, min\, rep$
13. $sqa\, d \;\rightarrow\; sqa\, u\, m\, max\, rep$
14. $sqa\, id \;\rightarrow\; sqa\, iu\, im\, max\, rep$
15. $sqa\, dd \;\rightarrow\; sqa\, du\, dm\, max\, rep$
16. $sqa \;\rightarrow\; s_0\, a_{\uparrow}\, min\, rep$
17. $s_0\, BXo \;\rightarrow\; s_0\, du\, max\, rep$
18. $s_0\, BXi \;\rightarrow\; s_0\, iu\, max\, rep$

Rules description:

1: request to submit the battery state;
2..7: computing the difference between maximum value and inverse value for charge and discharge (decrease the inverse flow);
8, 12: computing the direct values for charge and discharge (increase the direct flow);
13..15: restore the initial cell's values;
16: cell confirms that answers were submitted;
10: the cell increases output flow;
11: the cell decreases input flow.

The symbols' meaning are shown in Table 9.

Symbol	Description
u	The available energy level
m	The maximum energy level
iu	The charge level
im	The maximum charge level
du	The discharge level
dm	The maximum discharge level
q	Query
a	Answer acknowledge
BXi	The input can be increased
BXd	The output can be increased

Table 9. Battery cell symbols

5.11 Grid controller

The power Grid is defined by the installed power (m) which can be taken from the physical line. Let X represent the source identifier (one can have many connections for different voltages, like 220V and 380V). The request from the Grid Controller is forwarded to the source itself. Each of them answers with *SXu*, if the amount of power can be increased, or *SXd*, if the power can be decreased. The symbols are then delivered to the MCU, when all answers from the sources have been received.

The second phase of using the Controller occurs when actual increase or decrease commands are received. These are forwarded to the sources themselves, and all unnecessary symbols are cleaned up.

Rules for the cell *CtrlGrid* are presented below:

1. $s_0\, q \;\rightarrow\; sq\, q_{\downarrow}\, min\, rep$
2. $sq\, a \;\rightarrow\; sa\, min\, rep$
3. $sa\, Su \;\rightarrow\; sa\, Su_{\uparrow}\, max\, rep$
4. $sa\, Sd \;\rightarrow\; sa\, Sd_{\uparrow}\, max\, rep$
5. $sa \;\rightarrow\; s_0\, a_{\uparrow}\, max\, rep$
6. $s_0\, Su \;\rightarrow\; s_0\, Su_{\downarrow}\, max\, rep$
7. $s_0\, Sd \;\rightarrow\; s_0\, Sd_{\downarrow}\, max\, rep$
8. $s_0\, ATu \;\rightarrow\; s_0\, max\, rep$
9. $s_0\, ATd \;\rightarrow\; s_0\, max\, rep$
10. $s_0\, ALu \;\rightarrow\; s_0\, max\, rep$
11. $s_0\, ALd \;\rightarrow\; s_0\, max\, rep$
12. $s_0\, GEu \;\rightarrow\; s_0\, max\, rep$
13. $s_0\, GEd \;\rightarrow\; s_0\, max\, rep$
14. $s_0\, GPu \;\rightarrow\; s_0\, max\, rep$
15. $s_0\, GPd \;\rightarrow\; s_0\, max\, rep$
16. $s_0\, B1o \;\rightarrow\; s_0\, max\, rep$
17. $s_0\, B1i \;\rightarrow\; s_0\, max\, rep$
18. $s_0\, B2o \;\rightarrow\; s_0\, max\, rep$
19. $s_0\, B2i \;\rightarrow\; s_0\, max\, rep$

Rules description:

1: the request is retransmitted to the sources;
2: the cell waits for all sources to answer;
3..4: measured values are relayed to the general controller;
5: cell confirms that all measured values were submitted;
6..7: all commands are submitted to sources;
8..19: cleaning rules.

The symbols *SXu* and *SXd* play two roles:

1. if the cell is in the state *sa*, then the symbol indicating a source state (S) is sent to the general controller;
2. if the cell is in state s_0, then the symbol designating a command forwarded to the each source.

Unlike the external sources, symbols *SXu* and *SXd* have the multiplicity equal with the number of units that the power amount taken can be increased or decreased with.

The symbols' meaning are shown in Table 10.

Symbol	Description
q	Query
a	Answer acknowledge
Sxu	Source can have more load
Sxd	Source decrease load

Table 10. External power sources controller symbols

5.12 External sources (GRID)

The Grid is defined by the installed power (m) and by the actual power given, u. The request from the Grid is about the instantaneous power u, if that is above 0, and the possibility to increase the amount offered if maximum value m has not been reached. The answer can be positive or negative.

The second state occurs when actual increase or decrease commands are received and executed. Unnecessary symbols will be removed from the cell.

Rules for the cell GridX are as follows:

1. $s_0\, q \;\rightarrow\; sqd\, min\, rep$
2. $sqd\, u \;\rightarrow\; squ\, u\, ud\, max\, rep$
3. $sqd \;\rightarrow\; squ\, min\, rep$
4. $squ\, u\, m \;\rightarrow\; squ\, d\, max\, rep$
5. $squ\, m \;\rightarrow\; sqa\, m\, uu\, max\, rep$
6. $sqa\, d \;\rightarrow\; sqa\, u\, m\, max\, rep$
7. $sqa\, uu \;\rightarrow\; sqa\, SXu_{\uparrow}\, max\, rep$
8. $sqa\, ud \;\rightarrow\; sqa\, SXd_{\uparrow}\, max\, rep$
9. $sqa \;\rightarrow\; s_0\, a_{\uparrow}\, min\, rep$
10. $s_0\, SXu \;\rightarrow\; s_0\, u\, max\, rep$
11. $s_0\, SXd\, u \;\rightarrow\; s_0\, max\, rep$

Rules description:

1: request to submit the current state;
2..6: computing the difference between maximum value and current value;
7,8: the cell returns an answer;
9: cell confirms that answers were submitted;
10: the cell increases amount given;
11: the cell decreases power.

The symbols' meaning are shown in Table 11.

Symbol	Description
u	Current consumption
m	Maximum available for consumption
q	Query
a	Answer acknowledge
SXu	Consumption can be increased and the request to increase the used power
SXd	Consumption can be decreased and the request to decrease the used power

Table 11. External source cell symbols

6 Conclusions and future work

In this paper we presented a model for a real-life working system ([6]), we described the use of hyperdag P systems for a feedback-oriented infrastructure that quickly reacts to environment conditions and adapts the parameters accordingly. A detailed description of the system is followed by the complete rule set and transition diagrams, in order to better understand the concept. The model was tested and validated using the ad-hoc built simulator and the results were the ones expected.

Future work involves extending the simulator to accept new types of P systems and development of other models (like network-related algorithms) that can be simulated by using this architecture. Another aspect to be considered is a formal testing of the proposed model, by using techniques like the ones indicated in [5].

Acknowledgements

The authors wish to thank Dr. Raluca Lefticaru for her valuable advice and the SOP-HRD programme of the EU for funding the research. The work of Adrian Zafiu was supported by SOP-HRD grant 89/1.5/S/63700. The work of Cristian Ştefan was supported by SOP-HRD grant 88/1.5/S/52826.

References

1. Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. P systems and the Byzantine agreement. Report CDMTCS-375, Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, Auckland, New Zealand, January 2010.
2. Michael J. Dinneen, Yun-Bum Kim, and Radu Nicolescu. Synchronization in P modules. Report CDMTCS-378, Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, Auckland, New Zealand, February 2010.
3. J. Ellson, E. Gansner, and many others. Graphviz 2.28: Graph visualization software, 2012.
4. Florentin Ipate, Radu Nicolescu, Ionut-Mihai Niculescu, and Cristian Stefan. Synchronization of p systems with simplex channels. *CoRR*, abs/1108.3430, 2011.
5. Raluca Lefticaru, Marian Gheorghe, and Florentin Ipate. An empirical evaluation of p system testing techniques. *Natural Computing*, 10(1):151–165, 2011.
6. P. L. Milea, Adrian Zafiu, M. Drągulinescu, and O. Oltu. Mpp tracking method for pv systems, based on three points prediction algorithm. *UPB Scientific Bulletin*, 72(4):149–160, 2010.
7. Radu Nicolescu, Michael J. Dinneen, and Yun-Bum Kim. Structured modelling with hyperdag P systems: Part A. Report CDMTCS-342, Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, Auckland, New Zealand, December 2008.
8. Radu Nicolescu, Michael J. Dinneen, and Yun-Bum Kim. Structured modelling with hyperdag P systems: Part A. In *Brainstorming Week on Membrane Computing*, pages 85–107, 2009.

9. Radu Nicolescu, Michael J. Dinneen, and Yun-Bum Kim. Structured modelling with hyperdag P systems: Part B. Report CDMTCS-373, Centre for Discrete Mathematics and Theoretical Computer Science, The University of Auckland, Auckland, New Zealand, October 2009.
10. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
11. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.

7 Appendix

We present here the complete trace of the P-system evolution, at each step indicating the contents and current states of the cells.

Step	MemConf	Ctrl
0	s0 t23	s0
1	s0 t23 q	sq
2	sqt t23	sq
3	sql t23	sq MT23
4	sqa t23	sq MT23
5	s0 t23	sq MT23 a
6	s0 t23	sq MT23 a
7	s0 t23	sq MT23 a ST20
8	s0 t23	sq MT23 a2 ST20
9	s0 t23	sq MT23 a2 ST20
10	s0 t23	sq MT23 a2 ST20
11	s0 t23	sq MT23 a2 ST20 B1i500
12	s0 t23	sq MT23 a2 ST20 B1i500 CTu GEu788 B1o1000 Su2999
13	s0 t23	sq MT23 a2 ST20 B1i500 CTu GEu788 B1o1000 Su2999 CTd GEd12 B2i1000 Sd
14	s0 t23	sq MT23 a3 ST20 B1i500 CTu GEu788 B1o1000 Su2999 CTd GEd12 B2i1000 Sd CLu GPu793 B2o500
15	s0 t23	sq MT23 a4 ST20 B1i500 CTu GEu788 B1o1000 Su2999 CTd GEd12 B2i1000 Sd CLu GPu793 B2o500 CLd GPd7
16	s0 t23	sq MT23 a6 ST20 B1i500 CTu GEu788 B1o1000 Su2999 CTd GEd12 B2i1000 Sd CLu GPu793 B2o500 CLd GPd7
17	s0 t23	sqc MT23 ST20 B1i500 CTu GEu788 B1o1000 Su2999 CTd GEd12 B2i1000 Sd CLu GPu793 B2o500 CLd GPd7
18	s0 t23	sqc MT3 B1i500 CTu GEu788 B1o1000 Su2999 CTd GEd12 B2i1000 Sd CLu GPu793 B2o500 CLd GPd7
19	s0 t23	sqc MT2 B1i500 GEu788 B1o1000 Su2999 CTd GEd12 B2i1000 Sd CLu GPu793 B2o500 CLd GPd7 ATu
20	s0 t23	sqc B1i500 GEu788 B1o1000 Su2999 CTd GEd12 B2i1000 Sd CLu GPu793 B2o500 CLd GPd7 ATu
21	s0 t23	sqc B1i500 GEu788 B1o1000 Su2999 GEd12 B2i1000 Sd CLu GPu793 B2o500 CLd GPd7 ATu
22	s0 t23	sqc B1i500 GEu788 B1o1000 Su2999 GEd12 B2i1000 Sd GPu793 B2o500 CLd GPd7 ATu
23	s0 t23	sqc B1i500 GEu788 B1o1000 Su2999 GEd12 B2i1000 Sd GPu793 B2o500 GPd7 ATu
24	s0 t23	sqa B1i500 GEu788 B1o1000 Su2999 GEd12 B2i1000 Sd GPu793 B2o500 GPd7 ATu
25	s0 t23	sqa B1i500 GEu788 B1o999 Su2999 GEd12 B2i1000 GPu793 B2o500 GPd7 ATu
26	s0 t23	sqa GEu288 B1o999 Su2999 GEd12 B2i1000 GPu793 B2o500 GPd7 ATu
27	s0 t23	sqa B1o999 Su2999 GEd12 B2i712 GPu793 B2o500 GPd7 ATu
28	s0 t23	sqa B1o999 Su2999 GEd12 GPu81 B2o500 GPd7 ATu
29	s0 t23	sqa B1o999 Su2999 GEd12 GPu80 B2o500 GPd7
30	s0 t23	sqa Su2999 GEd12 GPu80 B2o500 GPd7
31	s0 t23	sqa GEd12 GPu80 B2o500 GPd7
32	s0 t23	sqa GPu80 B2o500 GPd7
33	s0 t23	sqa B2o500 GPd7
34	s0 t23	sqa B2o500
35	s0 t23	sqa

Table 1. System evolution - Part 1

SenzT	CtrlSenz	SenzL	CtrlCons	CtrlGen	CtrlBat
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0 q	s0	s0 q	s0 q	s0 q
s0 t20 q	s0	s0 q	sq	sq	sq
sqt t20	s0	sql	sq	sq	sq
sqa t20	s0 ST20	sqa	sq	sq	sq
s0 t20	s0 ST20 a2	s0	sq	sq	sq
s0 t20	sq ST20 a	s0	sq	sq	sq B1i500 B2i1000
s0 t20	sq a	s0	sq	sq	sq B1i500 B2i1000 B1o1000 B2o500
s0 t20	s0	s0	sq CTu CLu	sq GEu788 GPu793	sq B1i500 B2i1000 B1o1000 B2o500
s0 t20	s0	s0	sq CTu CLu CTd CLd	sq GEu788 GPu793 GEd12 GPd7	sq B1i500 B2i1000 B1o1000 B2o500 a2
s0 t20	s0	s0	sq CTu CLu CTd CLd a2	sq GEu788 GPu793 GEd12 GPd7 a2	sa B1i500 B2i1000 B1o1000 B2o500
s0 t20	s0	s0	sa CTu CLu CTd CLd	sa GEu788 GPu793 GEd12 GPd7	sa B2i1000 B1o1000 B2o500
s0 t20	s0	s0	sa CLu CTd CLd	sa GPu793 GEd12 GPd7	sa B2i1000 B2o500
s0 t20	s0	s0	sa CLu CLd	sa GPu793 GPd7	sa B2o500
s0 t20	s0	s0	sa CLd	sa GPd7	sa
s0 t20	s0	s0	sa	sa	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0 B1o Sd	s0 B1o Sd	s0 B1o Sd
s0 t20	s0	s0	s0 Sd GEu500 B1i500	s0 GEu500 Sd B1i500	s0 B1i500 Sd GEu500
s0 t20	s0	s0	s0 Sd GEu288 B1i500 B2i288	s0 GEu288 Sd B1i500 B2i288	s0 B1i500 B2i288 Sd GEu288
s0 t20	s0	s0	s0 Sd B1i500 B2i1000 GPu712	s0 GPu712 Sd B1i500 B2i1000	s0 B1i500 B2i1000 Sd GPu712
s0 t20	s0	s0	s0 Sd B1i500 B2i1000 GPu ATu	s0 GPu Sd B1i500 B2i1000 ATu	s0 B1i500 B2i1000 Sd GPu ATu
s0 t20	s0	s0	s0 Sd B1i500 B2i1000 GPu	s0 GPu Sd B1i500 B2i1000	s0 B1i500 B2i1000 Sd GPu
s0 t20	s0	s0	s0 Sd B1i500 B2i1000	s0 Sd B1i500 B2i1000	s0 B1i500 B2i1000 Sd
s0 t20	s0	s0	s0 Sd B2i1000	s0 Sd B2i1000	s0 B2i1000 Sd
s0 t20	s0	s0	s0 Sd	s0 Sd	s0 Sd
s0 t20	s0	s0	s0	s0	s0
s0 t20	s0	s0	s0	s0	s0

Table 1. System evolution - Part 2

CtrlGrid	ConsT	ConsL	GenEOL	GenPV
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0 q	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
sq	s0 u10 m100 q	s0 u10 m150 q	s0 u12 m800 q	s0 u7 m800 q
sq	sqd u10 m100	sqd u10 m150	sqd u12 m800	sqd u7 m800
sq	squ u10 m100 ud	squ u10 m150 ud	squ u12 m800 ud12	squ u7 m800 ud7
sq	squ m90 ud d10	squ m140 ud d10	squ m788 ud12 d12	squ m793 ud7 d7
sq	sqa m90 ud d10 uu	sqa m140 ud d10 uu	sqa m788 ud12 d12 uu788	sqa m793 ud7 d7 uu793
sq	sqa u10 m100 ud uu	sqa u10 m150 ud uu	sqa u12 m800 ud12 uu788	sqa u7 m800 ud7 uu793
sq Su2999	sqa u10 m100 ud	sqa u10 m150 ud	sqa u12 m800 ud12	sqa u7 m800 ud7
sq Su2999 Sd	sqa u10 m100	sqa u10 m150	sqa u12 m800	sqa u7 m800
sq Su2999 Sd a	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
sa Su2999 Sd	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
sa Sd	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
sa	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0 Sd B1o	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0 Sd GEu500 B1i500	s0 u10 m100	s0 u10 m150	s0 u12 m800	s0 u7 m800
s0 Sd GEu288 B1i500 B2i288	s0 u10 m100	s0 u10 m150	s0 u12 m800 GEu500	s0 u7 m800 GEu500
s0 Sd B1i500 B2i1000 GPu712	s0 u10 m100	s0 u10 m150	s0 u512 m800 GEu288	s0 u7 m800 GEu288
s0 Sd B1i500 B2i1000 GPu ATu	s0 u10 m100	s0 u10 m150	s0 u800 m800 GPu712	s0 u7 m800 GPu712
s0 Sd B1i500 B2i1000 GPu	s0 u10 m100 ATu	s0 u10 m150 ATu	s0 u800 m800	s0 u719 m800
s0 Sd B1i500 B2i1000	s0 u11 m100	s0 u10 m150	s0 u800 m800 GPu	s0 u719 m800 GPu
s0 Sd B2i1000	s0 u11 m100	s0 u10 m150	s0 u800 m800	s0 u720 m800
s0 Sd	s0 u11 m100	s0 u10 m150	s0 u800 m800	s0 u720 m800
s0	s0 u11 m100	s0 u10 m150	s0 u800 m800	s0 u720 m800
s0	s0 u11 m100	s0 u10 m150	s0 u800 m800	s0 u720 m800

Table 1. System evolution - Part 3

Bat1	**Bat2**	**Grid**	**Out**
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000 q	s0 u2000 m20000 im1000 dm500 q	s0 u m3000 q	
sqd u2000 m5000 im500 dm1000	sqd u2000 m20000 im1000 dm500	sqd u m3000	
sqd m3000 im500 dm1000 d2000	sqd m18000 im1000 dm500 d2000	squ u m3000 ud	
sqi m3000 im500 dm1000 d2000	sqi m18000 im1000 dm500 d2000	squ m2999 ud d	
sqo m3000 im500 dm1000 d2000	sqo m18000 im1000 dm500 d2000	sqa m2999 ud d uu2999	
sqa m3000 im500 dm1000 d2000	sqa m18000 im1000 dm500 d2000	sqa u m3000 ud uu2999	
sqa u2000 m5000 im500 dm1000	sqa u2000 m20000 im1000 dm500	sqa u m3000 ud	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	sqa u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000 B1o	s0 u2000 m20000 im1000 dm500 B1o	s0 u m3000	
s0 u2000 m5000 im500 dm1000 du	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000 du	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000 du	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000 du	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000 du	s0 u2000 m20000 im1000 dm500	s0 u m3000	
s0 u2000 m5000 im500 dm1000 du B1i500	s0 u2000 m20000 im1000 dm500 B1i500	s0 u m3000	
s0 u2000 m5000 im500 dm1000 du B2i1000 iu500	s0 u2000 m20000 im1000 dm500 B2i1000	s0 u m3000	
s0 u2000 m5000 im500 dm1000 B2i1000 iu499	s0 u2000 m20000 im1000 dm500 iu1000	s0 u m3000 Sd	
s0 u2000 m5000 im500 dm1000 iu499	s0 u2000 m20000 im1000 dm500 iu1000	s0 m3000	

Table 1. System evolution - Part 4

A Membrane-Inspired Evolutionary Algorithm with a Population P System and its Application to Distribution System Reconfiguration

Gexiang Zhang[1], Miguel A. Gutiérrez-Naranjo[2], Yanhui Qin[3], Marian Gheorghe[4]

[1,3] School of Electrical Engineering,
Southwest Jiaotong University, Chengdu 610031, P.R. China
[1]zhgxdylan@126.com; [3]qinyanhui@gmail.com

[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, 41012, Spain
magutier@us.es

[4] Department of Computer Science,
University of Sheffield Regent Court,
Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk

Summary. This paper develops a membrane-inspired evolutionary algorithm, PSMA, which is designed by using a population P system and a quantum-inspired evolutionary algorithm (QIEA). We use a population P system with three cells to organize three types of QIEAs, where communications between cells are performed at the level of genes, instead of the level of individuals reported in the existing membrane algorithms in the literature. Knapsack problems are applied to discuss the parameter setting and to test the effectiveness of PSMA. Experimental results show that PSMA is superior to four representative QIEAs and our previous work with respect to the quality of solutions and the elapsed time. We also use PSMA to solve the optimal distribution system reconfiguration problem in power systems for minimizing the power loss.

Key words: Membrane computing; membrane-inspired evolutionary algorithm; population P system; distribution system reconfiguration

1 Introduction

According to the research development of interactions on membrane computing and evolutionary computation, two kinds of research topics, membrane-inspired evolutionary algorithms (MIEAs) and automated design of membrane computing models (ADMCMs), have been reported in the literature.

The automated synthesis of some types of membrane computing models or of a high level specification of them is envisaged to be obtained by applying various heuristic search methods. ADMCMs aim to circumvent the programmability issue of membrane-based models for complex systems. In this direction, Suzuki and Tanaka made the first attempts [20, 21] to introduce a genetic method to the *Artificial Cell Systems (ACS)* via a P system model called *Abstract Rewriting Systems on Multisets* [19, 22], a rewriting Membrane Computing model where the P systems have only one membrane. More recently, new attempts of using evolutionary algorithms to evolve P systems have been presented (see, e.g. [3, 5, 12, 24]). In [3], a nested evolutionary algorithm was used to tuning parameters of P system models. The automatic design of P systems for fulfilling an specific task was first discussed in [5], where genetic algorithms are used for finding simple P systems. In [5], the membrane structure is settled and the genetic evolution only corresponds to the set of rules. A *population* of P systems is considered and two genetic operations, *crossover* and *mutation* perform the evolution of the population. This work was extended from 4^2 to n^2 P systems in [12] by introducing a quantum-inspired evolutionary algorithm (QIEA), where the set of rules were encoded by a binary string and evolutionary operations (quantum-inspired gate (Q-gate) update) were performed on genotypic individuals (quantum-inspired bits (Q-bits)), instead of phenotypic individuals (binary bits) or evolution rules of P systems. The outstanding advantage of this approach is that the difficulty of designing evolutionary operators in the phenotypic space, such as crossover and mutation ones is effectively avoided. In [24], the design of P systems for generating languages and fitness functions were discussed.

A MIEA concentrates on generating new approximate algorithms for solving various optimization problems by using the hierarchical or network structures of membranes and rules of membrane systems, and the concepts and principles of meta-heuristic search methodologies [33, 34]. In [11, 17], a cell-like membrane system with a nested membrane structure (NMS) was used to combine with simulated annealing and genetic algorithms to solve traveling salesman problems and controller design problems for a marine diesel engine. In [31], a QIEA based on P systems (QEPS) was proposed by incorporating a one-level membrane structure (OLMS) with a QIEA. Knapsack problems were applied to verify that QEPS is superior to its counterpart method and OLMS has an advantage over NMS. The use of QEPS to solve sixty-five satisfiability problems with different complexities was discussed in [33]. In [34], the QEPS performance was improved by introducing a local search and the modified QEPS applied to analyze sixteen radar emitter signals. In [4,29], OLMS was integrated with differential evolution approaches and ant colony optimization to solve numeric optimization and travelling salesman problems. In [35], the use of a cell-like membrane system with active membranes to design a MIEA was designed for solving for combinatorial optimization problems. In [27], a MIEA was presented to solve the DNA sequence design problem, which has been proved to be NP-hard. In the above MIEAs, heuristic search methods, such as genetic algorithms, QIEA, differential evolution and ant colony optimiza-

tion, were considered as an independent subalgorithm inside each membrane. This idea was extended by the proposal of a membrane algorithm with quantum-inspired subalgorithm (MAQIS) in [30], where each membrane contains one component of the approach and all the components inside membranes cooperate together to produce offspring in a single evolutionary generation. The effectiveness of MAQIS was tested on knapsack problems and image sparse decomposition problems. It is worth pointing out that the analysis of the dynamic behavior of MIEAs in the process of evolution with respect to population diversity and convergence showed that MIEAs have better capabilities to balance exploration and exploitation than their corresponding optimization algorithms used [32, 36]. Until now MIEAs have been studied in conjunction with cell-like membrane systems with fixed membrane structures and by principally considering an evolutionary computing approach as a subalgorithm put inside a membrane. Further research topics might include cell-like membrane systems with active membranes, tissue-like membrane systems and population membrane systems for exploring more real-world applications of membrane computing.

In spite of the biological inspiration of membrane computing and evolutionary computation, in the literature there are only a few examples of papers bridging them. We continue to push this work forward. The main motivation of this work is to use a population membrane system to design a MIEA for distribution system reconfiguration. The algorithm called PSMA is designed by appropriately considering a population P system and three variants of QIEAs, where the communications between cells are performed at the level of genes, instead of the level of individuals reported in the existing membrane algorithms in the literature. This is the first attempt to apply a population P system to design an approximate optimization approach. Knapsack problems and distribution system reconfiguration are applied to test the effectiveness and the application of PSMA, respectively. Experimental results show that PSMA can obtain better solutions that four types of QIEAs and QEPS (a MIEA reported in [31]) and is competitive to five types of optimization algorithms for solving distribution system reconfiguration problems in power systems.

This paper is organized as follows. Section 2 introduces briefly QIEA and population P systems, and then describes PSMA in detail. Section 3 presents experiments conducted on knapsack problems for testing the PSMA performance. The application of PSMA to distribution system reconfiguration is discussed in Section 4. Concluding remarks follow in Section 5.

2 PSMA

PSMA uses the network framework of a population P system to organize the objects consisting of quantum-inspired bits (Q-bits) and classical bits, and rules made up of several quantum-inspired gate (Q-gate) evolutionary rules like in QIEA and evolution rules like in membrane systems. To clearly and concisely describe

PSMA, we first give brief introductions on QIEAs and population P systems, and then turn to a detailed presentation of the introduced MIEA.

2.1 QIEA

Inspired by concepts and principles of quantum computing such as quantum bits, quantum gates and a probabilistic observation, Han and Kim [9] proposed a new evolutionary algorithm, QIEA, for a classical computer instead of quantum one. In QIEA, a Q-bit representation is applied to describe individuals of a population; a Q-gate is introduced to generate the individuals at the next generation; a probabilistic observation is employed to link Q-bit representation with binary solutions [28]. A Q-bit is defined by a pair of numbers (α, β) represented as $[\alpha\ \beta]^T$, where $|\alpha|^2$ and $|\beta|^2$ are probabilities that the observation of the Q-bit will render a '0' or '1' state and $\xi = \arctan(\beta/\alpha)$ is the phase of the Q-bit [9,28]. Normalization requires that $|\alpha|^2 + |\beta|^2 = 1$. The evolution of QIEA depends on the operation of Q-gates on Q-bit individuals. The basic pseudocode algorithm for a QIEA is shown in Fig. 1 and a brief description for each step is as follows (here we just list the outline of QIEA algorithm and details will be provided in the algorithm description of PSMA (Section 2.3).).

```
    Begin
        t ← 1
(i)     Initialize Q(t)
        While (not termination condition) do
(ii)        Make P(t) by observing the states of Q(t)
(iii)       Evaluate P(t)
(iv)        Update Q(t) using Q-gates
(v)         Store the best solutions among P(t)
            t ← t + 1
        End
    End
```

Fig. 1. Pseudocode algorithm for a QIEA [9, 28]

(i) In the "initialize $Q(t)$" step, a population $Q(t)$ with n Q-bit individuals is generated, $Q(t)=\{\boldsymbol{q}_1^t, \boldsymbol{q}_2^t, \cdots, \boldsymbol{q}_n^t\}$, at generation t, where $\boldsymbol{q_i^t}(i = 1, 2, \cdots, n)$ is an arbitrary individual in $Q(t)$ and denoted as

$$\boldsymbol{q}_i^t = \begin{bmatrix} \alpha_{i1}^t | \alpha_{i2}^t | \cdots | \alpha_{il}^t \\ \beta_{i1}^t | \beta_{i2}^t | \cdots | \beta_{il}^t \end{bmatrix} \tag{1}$$

where l is the number of Q-bits, i.e., the string length of the Q-bit individual.

(ii) By observing the states $Q(t)$, binary solutions in $P(t)$, where $P(t) = \{\boldsymbol{x_1^t}, \boldsymbol{x_2^t}, \cdots, \boldsymbol{x_n^t}\}$ are produced at step t. According to the current probability, either $\left|\alpha_{ij}^t\right|^2$ or $\left|\beta_{ij}^t\right|^2$ of $\boldsymbol{q_i^t}$, $i = 1, 2, \cdots, n, j = 1, 2, \cdots, l$, a binary bit 0 or 1 is generated. Thus, l binary bits can construct a binary solution $\boldsymbol{x_i^t}(i = 1, 2, \cdots, n)$. More details can be referred to Step 2 *Observation* in the algorithm description of PSMA (Section 2.3).

(iii) Binary solutions $\boldsymbol{x_j^t}(j = 1, 2, \cdots, n)$ are evaluated and assigned fitness values with respect to a criterion.

(iv) In this step, Q-bit individuals in $Q(t)$ are updated by applying the current Q-gates. The details will be expounded in Step 5 *Q-gate update* in the algorithm description of PSMA (Section 2.3).

(v) The best solutions among $P(t)$ are selected and stored.

2.2 Population P Systems

A population P system is a special kind of tissue P systems except for two important differences that the structure can be dynamically changed by using bond making rules and cells are allowed to communicate indirectly by means of the environment [2].

A population P system with degree n is formally defined as follows [2]

$$\mathcal{P} = (V, \gamma, \alpha, \omega_e, C_1, C_2, \ldots, C_n, c_o),$$

where

(i) V is a finite alphabet of symbols called objects;

(ii) $\gamma = (\{1, 2, \ldots, n\}, E)$, with $E \subseteq \{\{i, j\} | 1 \leq i \neq j \leq n\}$, is a finite undirected graph;

(iii) α is a finite set of bond making rules $(i, x_1; x_2, j)$, with $x_1, x_2 \in V^*$, and $1 \leq i \neq j \leq n$;

(iv) $\omega_e \in V^*$ is a finite multiset of objects initially assigned to the environment;

(v) $C_i = (\omega_i, S_i, R_i)$, for each $1 \leq i \leq n$, with
   (a) $\omega_i \in V^*$ a finite multiset of objects,
   (b) S_i is a finite set of communication rules; each rule has one of the following forms: $(a; b, in)$, $(a; b, enter)$, $(b, exit)$, for $a \in V \bigcup \{\lambda\}, b \in V$,
   (c) R_i is a finite set of transformation rules of the form $a \to y$, for $a \in V$, and $y \in V^+$;

(vi) c_o is the (label of the) output cell, $1 \leq c_o \leq n$.

A population P system $\mathcal{P}$ is defined as a collection of n cells where each cell C_i corresponds in a one-to-one manner to a node i in a finite undirected graph γ, which defines the initial structure of the system. Cells are allowed to communicate alongside the edges of the graph γ, which are unordered pairs of the form $\{(i, j)\}$, with $1 \leq i \neq j \leq n$. The cells C_i, $1 \leq i \leq n$, are associated in a one-to-one manner with the set of nodes $\{1, 2, \ldots, n\}$. Each cell C_i gets assigned a finite multiset of objects ω_i, a finite set of communication rules S_i, and a finite set of transformation

rules R_i. Each set R_i contains rules of the form $x \to y$ that allow cell i to consume a multiset x in order to produce a new multiset y inside cell i. Communication rules in S_i of the form $(a; b, in)$ are instead used by cell i to receive objects from its neighboring cells if the object a is placed in the cell i. The rules of the forms $(a; b, enter)$ mean that objects from the environment can enter the cell i if an object a is present in it. The rules of the forms $(b, exit)$ allow the cell i to release an object b in the environment.

Cell capability of moving objects alongside the edges of the graph is influenced by particular bond making rules in α that allow cells to form new bonds. A bond making rule $(i, x_1; x_2, j)$ specifies that, in the presence of a multiset x_1 in the cell i and a multiset x_2 inside the cell j, a new bond can be created between the two cells. This means that a new edge $\{i, j\}$ can be added to the graph that currently defines the structure of the system. Thus the structure of a population P system can be dynamically changed in the process of evolution of the system.

A step of a computation in a population P system $\mathcal{P}$ is defined as being performed in two separate stages: the content of the cells is firstly modified by applying the communication rules in S_i, and the transformation rules in R_i, for all $1 \leq i \leq n$; the structure of the system is then modified by using the bond making rules in α. A successful computation in $\mathcal{P}$ is defined as a finite sequence of configurations from processing the initial multisets ω_i to the final state where the content of the cells cannot be modified anymore by means of some communication rules and transformation rules after a last bond making stage. The result is given by the number of objects that are placed inside the output cell c_o in the final configuration.

2.3 PSMA

In this subsection, we design PSMA by applying the dynamic network structure of a population P system with three cells and three representative variants of QIEAs that have good performance in terms of the investigations in [9, 10, 28, 37, 38]. Specifically, three QIEAs, QIEA02 [9], QIEA04 [10] and QIEA07 [38], are placed inside three cells of the population P system in a common environment. The objects consist of Q-bits and classical bits. The rules are composed of observation and Q-gate update rules of QIEAs, transformation rules in the population P system, evaluation rules for candidate solutions, communication rules for exchanging information between the three cells and bond making rules for modifying the structure of the system. Q-bits, organized as a Q-bit individual which is a special string of Q-bits, are processed as multisets of objects. Classical bits, which are obtained from their corresponding Q-bits by applying a probabilistic observation process, are arranged as a binary string and are treated with also as multisets of objects. Inside each cell, the processes of initialization, observation, evaluation and Q-gate update processes for producing offspring are performed independently. Information exchange between individuals are executed through communications between cells at the level of genes. In PSMA, a binary string corresponds a candidate solution

of a problem. The set of rules are responsible for evolving the system. The framework of the population P system used in PSMA is shown in Fig. 2, where ovals represent the cells and dashed lines indicate the links. The population P system can be described as the following construct

$$\mathcal{P} = (V, \gamma, \alpha, \omega_e, C_1, C_2, C_3, c_e),$$

where

(i) V is a finite alphabet that consists of all possible Q-bits and classical bits (*objects*)(It is worth noting that the alphabet used in this paper is finite because the number of possible Q-bits equals the product of the number of Q-bit individuals and the length of a Q-bit individual);
(ii) $\gamma = (\{1, 2, 3\}, E)$, with $E = \{(1, 2), (1, 3), (2, 3)\}$, is a finite undirected graph;
(iii) α is a finite set of bond making rules $(i, \lambda; \lambda, j)$ or $\emptyset$ if no new bond can be added;
(iv) $\omega_e = \lambda$;
(v) $C_i = (\omega_i, S_i, R_i)$, for each $1 \leq i \leq 3$, with
  (a) $\omega_i = \boldsymbol{q_1 q_2 \cdots q_{n_i}}$, where $\boldsymbol{q_i}, i = 1, 2, \cdots, n_i$, is a Q-bit individual as shown in (1); n_i is the number of individuals in cell C_i and satisfies $\sum_i^3 n_i = N$, where N is the total number of individuals in this system;
  (b) S_i is a finite set of communication rules; each rule has one of the following forms: $(\lambda; b, in)$, $(b, exit)$, for $b \in V$,
  (c) R_i is a finite set of transformation rules of the form $a \to y$, for $a \in V$, and $y \in V^+$;
(vi) c_e means that the result is collected in the environment.

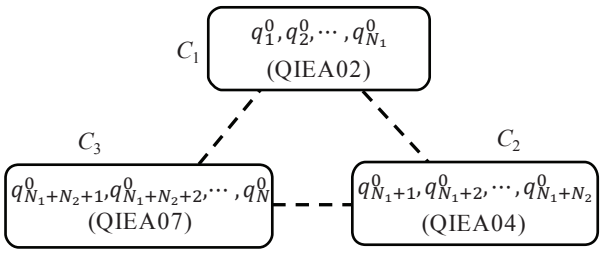


Fig. 2. The framework of the population P system involved in this paper

To clearly understand PSMA, in what follows we describe its algorithm step by step.

Step 1 *Initialization*: a membrane structure of a population P system with three cells in a common environment is created. An initial population with N individuals is generated. Each individual is composed of a certain number of Q-bits. The N individuals are randomly scattered across the three cells so that $n_i > 1$ and $\sum_i^3 n_i = N$, $i = 1, 2, 3$.

Step 2 *Observation*: a probabilistic observation process occurring in step (ii) of QIEA is applied to establish a link between genotypes and phenotypes, i.e., between Q-bits and classical bits. To be specific, as for the Q-bit $[\alpha\ \beta]^T$, if a random number r between 0 and 1 is less than $|\beta|^2$, i.e., $r < |\beta|^2$, the observed classical bit equals 1, otherwise, it is 0. Thus, given a Q-bit individual, we can get a corresponding binary solution. The observation process is shown in Fig. 3

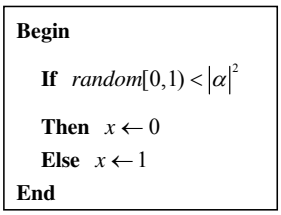


Fig. 3. Observation process in PSMA

Step 3 *Evaluation*: a specific criterion with respect to a problem is used to evaluate all the binary solutions obtained in Step 2. This step is identical with step (iii) of QIEA.

Step 4 *Communication*: Suppose that P_k represents the binary individuals obtained in Step 2 inside cell C_k, $P_k = \{\boldsymbol{x}_{\boldsymbol{1}}^{\boldsymbol{k}}, \boldsymbol{x}_{\boldsymbol{2}}^{\boldsymbol{k}}, \cdots, \boldsymbol{x}_{\boldsymbol{N}}^{\boldsymbol{k}}\}$, where $k = 1, 2, 3$; $\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{k}} = b_{i1}^k b_{i2}^k \cdots b_{im}^k$, where b_{ij}^k is a gene of $\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{k}}$ and m is the number of genes in an individual; the fitness of the individual $\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{k}}$ is $f(\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{k}})$. In this step, as for each gene b_{ij}^k in the binary individual $\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{k}}$, a random number r_c following a uniform distribution in the range [0, 1] is produced; if $r_c < p_c$, we randomly choose two binary individuals, $\boldsymbol{x}_{\boldsymbol{c_1}}^{\boldsymbol{k_1}}$ and $\boldsymbol{x}_{\boldsymbol{c_2}}^{\boldsymbol{k_2}}$, from the whole population (N individuals) except for the individual $\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{k}}$, where k_1 and k_2 are the labels of cells, k_1,k_2=1,2,3; p_c is a parameter denoting a communication rate and will be discussed in the next section. If $f(\boldsymbol{x}_{\boldsymbol{c_1}}^{\boldsymbol{k_1}})$ is better than $f(\boldsymbol{x}_{\boldsymbol{c_2}}^{\boldsymbol{k_2}})$, we use the gene $b_{c_1 j}^{k_1}$ to replace b_{ij}^k, otherwise, we use the gene $b_{c_2 j}^{k_2}$ to replace b_{ij}^k. Thus we can obtain another binary individual $\bar{\boldsymbol{x}}_{\boldsymbol{i}}^{\boldsymbol{k}}$ corresponding to $\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{k}}$. In the process of replacement, the values of k_1, k_2 and k decide what structure will be created and used to perform the communication for information exchange between cells k and k_1 or k_2. As for the values of k_1, k_2 and k, there are three cases: (1) $k_1 = k_2 = k$ means that no communication will be performed, i.e., the dashed lines in Fig. 2 do not work and the three cells are separate; (2) $k_1 = k_2 \neq k$ or $k_1 = k \neq k_2$ or $k_1 \neq k_2 = k$ means that communication is performed between two cells, i.e., only one of the dashed lines in Fig. 2 works and the communication rule $(\lambda; b, in)$ is performed between the two cells having the channel that works; (3) $k_1 \neq k_2 \neq k$ means that the three dashed lines in Fig. 2 work and the three cells communicate with each other. Thus the communication rule $(\lambda; b, in)$ is performed between each pair of cells.

Step 5 *Q-gate update*: transformation rules of the form $a \to y$ is utilized to evolve the objects in each of the three cells. The rules considered here are applied according to evolutionary mechanisms of QIEAs, instead of the semantics of P systems. The Q-gate update procedure

$$\begin{bmatrix}\alpha^{t+1}\\ \beta^{t+1}\end{bmatrix} = G(\theta)\begin{bmatrix}\alpha^{t}\\ \beta^{t}\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}\alpha^{t}\\ \beta^{t}\end{bmatrix} \tag{2}$$

is used to transform the current Q-bit $[\alpha^t\ \beta^t]^T$ into the corresponding Q-bit $\left[\alpha^{t+1}\ \beta^{t+1}\right]^T$ at generation t+1. The rotation angle θ in the Q-gate $G(\theta)$ in different cells has different definitions.

To be specific, in cell 1, the rotation angle is defined as $\theta = s(\alpha,\beta)\cdot\Delta\theta$, where $\Delta\theta$ is the value of θ determining the convergence speed of the algorithm and $s(\alpha,\beta)$ is the sign of θ deciding the search direction. The approach for looking up the rotation angle θ in [9] is shown in Tables 1, where $f(.)$ is the fitness function; α and β are the probabilities of the current Q-bit.

Table 1. Q-gate update approach in cell 1, where $f(.)$ is the fitness function, $\Delta\theta$ and $s(\alpha,\beta)$ are the value and the sign of θ, x and b are the bits of the binary individuals $\boldsymbol{x_i^1}$ and $\boldsymbol{x}_i^1$, respectively [9]

x	b	$f(\boldsymbol{x}) \geq f(\boldsymbol{b})$	$\Delta\theta$	$s(\alpha,\beta)$	
				$\alpha = 0$	$\beta = 0$
0	0	False	0	–	–
0	0	True	0	–	–
0	1	False	0.01π	+1	-1
0	1	True	0	–	–
1	0	False	0.01π	+1	-1
1	0	True	0	–	–
1	1	False	0	–	–
1	1	True	0	–	–

In cell 2, the Q-gate update procedure in (2) and the approach in Table 1 are firstly used. Then an additional process is applied to modify the Q-bit $\left[\alpha^{t+1}\ \beta^{t+1}\right]^T$. The modification method is as follows.

(i) If $|\alpha^{t+1}|^2 \leq \epsilon$ and $|\beta^{t+1}|^2 \geq 1-\epsilon$, $\left[\alpha^{t+1}\ \beta^{t+1}\right]^T = \left[\sqrt{\epsilon}\ \sqrt{1-\epsilon}\right]^T$;
(ii) If $|\alpha^{t+1}|^2 \geq 1-\epsilon$ and $|\beta^{t+1}|^2 \leq \epsilon$, $\left[\alpha^{t+1}\ \beta^{t+1}\right]^T = \left[\sqrt{1-\epsilon}\ \sqrt{\epsilon}\right]^T$.

According to the investigation in [10], the parameter ϵ is usually assigned as 0.01.

In cell 3, the approach for choosing the quantum rotation angle was defined by using the ratio of the probabilities of Q-bits [38]. The rotation angle θ is defined as

$$\theta = \theta_0 s(\alpha, \beta) f(\gamma_\alpha, \gamma_\beta) \tag{3}$$

where α and β represent the probabilities of a Q-bit; θ_0 is an initial rotation angle and is usually set to 0.05π; $s(\alpha, \beta)$ is a function determining the search direction of the algorithm; $f(\gamma_\alpha, \gamma_\beta)$ is a function of γ_α or γ_β, where $\gamma_\alpha = |\beta|/\alpha$, $\gamma_\beta = 1/\gamma_\alpha$. The values of $s(\alpha, \beta)$ and $f(\gamma_\alpha, \gamma_\beta)$ can be obtained in Table 2.

Table 2. Q-gate update approach in cell 3, where $f(.)$ is the fitness function, x and b are the bits of the binary individuals $\boldsymbol{x_i^1}$ and $\bar{\boldsymbol{x}}_i^1$, respectively [38]

x	b	$f(\boldsymbol{x}) \geq f(\boldsymbol{b})$	$s(\alpha, \beta)$			$f(\gamma_\alpha, \gamma_\beta)$
			$\alpha\beta \geq 0$	$\alpha\beta < 0$	$\alpha\beta = 0$	
0	0	false	-1	+1	±1	$\exp(-\gamma_\beta)$
0	0	true	-1	+1	±1	$\exp(-\gamma_\beta)$
0	1	false	+1	-1	±1	$\exp(-\gamma_\alpha)$
0	1	true	-1	+1	±1	$\exp(-\gamma_\beta)$
1	0	false	-1	+1	±1	$\exp(-\gamma_\beta)$
1	0	true	+1	-1	±1	$\exp(-\gamma_\alpha)$
1	1	false	+1	-1	±1	$\exp(-\gamma_\alpha)$
1	1	true	+1	-1	±1	$\exp(-\gamma_\alpha)$

Step 6 *Halting*: the algorithm stops when a prescribed number of evolutionary generations is attained.

Step 7 *Output*: the communication rule $(b, exit)$ is responsible for sending the best solutions out to the environment at the end of the computation. To be specific, each cell send the best solution inside it out to the environment at the end of the computation; thus there are three solutions coming from three cells in the environment; through a comparison, we collect the best one among the three solutions as the final solution of the computation.

3 Experiments

In this section, a well-known NP-hard combinatorial optimization problem, knapsack problem, is used to test the PSMA performance. The knapsack problem can be described as selecting from among various items those items that are most profitable, given that the knapsack has limited capacity [7, 9]. The knapsack problem is to select a subset from the given number of items so as to maximize the profit $f(x)$:

$$f(x) = \sum_{i=1}^{k} p_i x_i \tag{4}$$

subject to

$$\sum_{i=1}^{k} w_i x_i \leq C \tag{5}$$

where k is the number of items; p_i is the profit of the i-th item; w_i is the weight of the i-th item; C is the capacity of the given knapsack; and x_i is 0 or 1.

In the following experiments, strongly correlated sets of unsorted data are used

$$\omega_i = uniformly\, random[1, 50]$$

$$p_i = w_i + 25$$

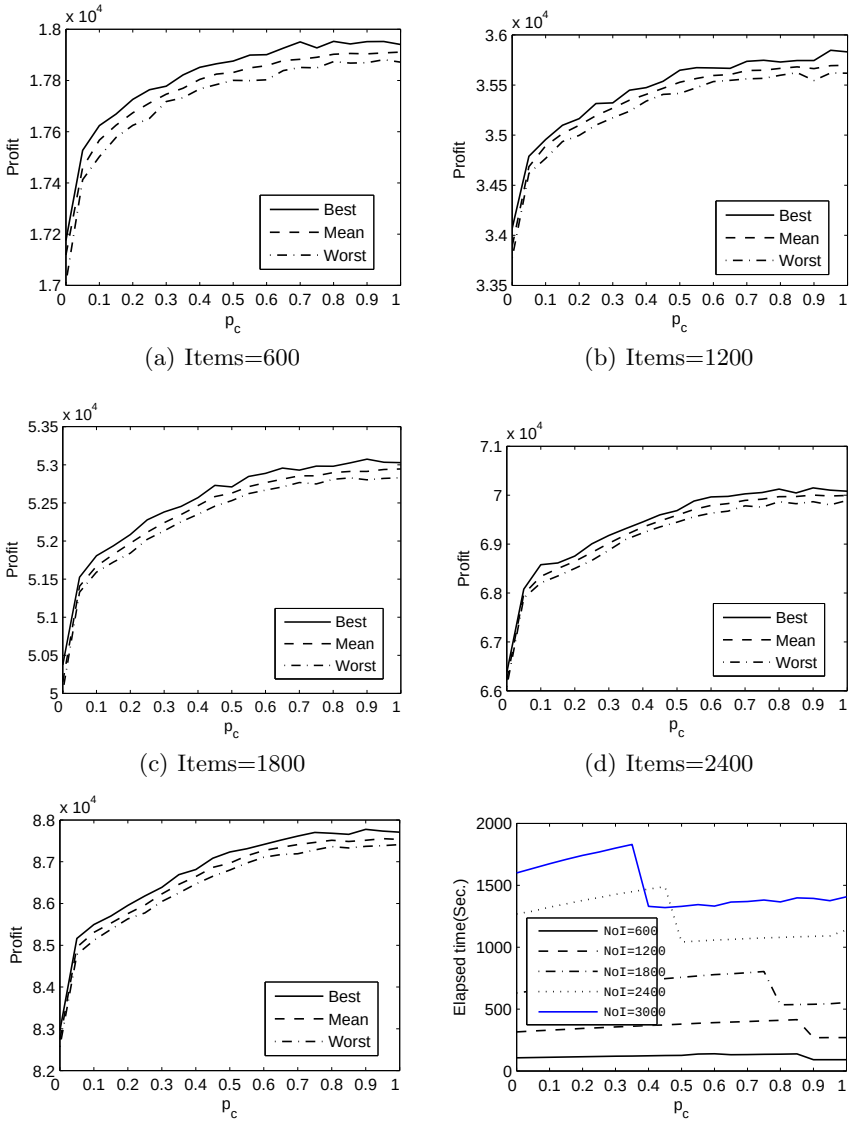
and the average knapsack capacity C is applied.

$$C = \frac{1}{2} \sum_{i=1}^{k} w_i \tag{6}$$

First of all, we use five knapsack problems with respective 600, 1200, 1600, 1800, 2400 and 3000 to discuss the choice of the parameter p_c in PSMA. The population size N=20. The the numbers, 20000, 30000, 40000, 60000 and 60000, of function evaluations are used as the stopping conditions for knapsack problems with 600, 1200, 1600, 1800, 2400 and 3000, respectively. Let p_c vary from 0 to 1 with interval 0.05, i.e., there are 21 cases. In the experiment, we perform 30 independent runs for each of 21 values of p_c of each knapsack problem. We record the best, mean and worst solutions over 30 runs and the elapsed time per run. Experimental results are shown in Fig. 4. It can be seen from these results that the parameter p_c could be assigned as the value ranged between 0.9 and 0.95 in terms of the quality of solutions and the elapsed time. Thus we set p_c to 0.9 in the following experiments.

To test the effectiveness of PSMA, Fifteen knapsack problems that have the items varied from 200 to 3000 items with interval 200 are used to conduct comparative experiments. Benchmark algorithms are considered to be composed of four types of QIEAs and the membrane algorithm QEPS in [31]. The four QIEAs include QIEA02 in [9], QIEA04 in [10], QIEA07 in [38] and QIEA08 [25]. 20 individuals are used in the six algorithms and 30 independent runs are performed for each of the 15 cases of each algorithm. The stopping condition for the six algorithms is set as follows: 20000 function evaluations for the first 4 knapsack problems; 30000 function evaluations for the 3 knapsack problems with 1000, 1200 and 1400; 40000 function evaluations for the 4 knapsack problems with 1600, 1800, 2000 and 2200; 60000 function evaluations for the last 4 knapsack problems. The best, mean and worst solutions over 30 independent runs and the elapsed time per run are recorded and listed in Tables 3 and 4.

As shown in Tables 3 and 4, we can conclude that PSMA is superior to QIEA02, QIEA04, QIEA07, QIEA08 and QEPS in terms of the the quality of best, mean and worst solutions and the elapsed time. To show that PSMA really outperforms



(a) Items=600 (b) Items=1200

(c) Items=1800 (d) Items=2400

(e) Items=3000 (f) Elapsed time per run (Sec.)

Fig. 4. Experimental results for p_c

Table 3. Experimental results of the first 8 knapsack problems. Best, Mean, Worst and Time represent the best, mean and worst solutions over 30 independent runs and the elapsed time per run, respectively (to be continued)

items		200	400	600	800	1000	1200	1400	1600
QIEA02	Best	5885	11650	17403	22940	28673	34399	39560	45277
	Mean	5786	11553	17173	22659	28333	33984	39149	44864
	Worst	5359	11396	16851	22010	27954	33424	38488	44423
	Time	24	48	72	96	182	221	259	413
QIEA04	Best	5749	11272	16561	21684	27024	32429	37329	42892
	Mean	5674	11081	16327	21499	26812	32210	37134	42596
	Worst	5627	10666	15680	20809	26524	31213	36028	42096
	Time	29	57	89	115	225	272	320	547
QIEA07	Best	5935	11850	17749	23390	29204	35099	40490	46403
	Mean	5893	11760	17627	23286	29080	34949	40267	46184
	Worst	5859	11700	17527	23139	28929	34822	40114	46002
	Time	26	52	78	104	197	249	402	420
QIEA08	Best	5456	10699	15734	20956	26073	31419	36384	41750
	Mean	5367	10615	15659	20747	25901	31244	36081	41499
	Worst	5325	10536	15591	20634	25775	31071	35942	41387
	Time	29	58	92	126	249	309	376	599
QEPS	Best	5959	11873	17702	23403	29531	35441	40886	47242
	Mean	5945	11837	17647	23257	29373	35292	40722	47018
	Worst	5909	11778	17575	23109	29198	35061	40364	46672
	Time	22	44	70	92	178	215	246	398
PSMA	Best	5984	11975	18000	23859	29822	35845	41412	47470
	Mean	5963	11965	17945	23782	29750	35782	41301	47365
	Worst	5959	11946	17902	23729	29687	35727	41214	47300
	Time	32	64	97	129	240	288	337	512

the other five algorithms, we go further to employ statistical techniques to analyze the behavior of the six algorithms over the 15 knapsack problems. Both parametric and non-parametric methods are considered. The parametric statistical analysis, also called single-problem analysis [6], is used to analyze whether there is a significant difference over one optimization problem between two algorithms. The non-parametric statistical test, also called multiple-problem analysis [6], is applied to compare different algorithms whose results represent average values for each problem. In Tables 5, a 95% confidence t-test is applied to check whether the mean solutions of the two pairs of algorithms, PSMA vs. QIEA02, QIEA04, QIEA07, QIEA08 and QEPS, are significantly different or not. Two non-parametric tests, Wilcoxon's and Friedman's tests, are employed to check whether there are signifi-

Table 4. Experimental results of the last 7 knapsack problems. Best, Mean, Worst and Time represent the best, mean and worst solutions over 30 independent runs and the elapsed time per run, respectively (continued)

items		1800	2000	2200	2400	2600	2800	3000
QIEA02	Best	50784	56453	61645	66683	72546	77511	83294
	Mean	50163	55879	61175	65984	71992	76734	82608
	Worst	49506	55129	59820	64981	71497	75924	82020
	Time	475	538	619	1056	1176	1310	1454
QIEA04	Best	47920	53276	58723	62952	68858	73355	79068
	Mean	47513	53018	58278	62523	68448	72938	78548
	Worst	46444	51889	56942	62250	66910	71360	76743
	Time	569	636	708	1268	1356	1448	1560
QIEA07	Best	51882	57579	63199	68351	74531	79471	85343
	Mean	51669	57414	62985	68093	74237	79215	85073
	Worst	51459	57277	62768	67932	73998	78685	84753
	Time	475	530	587	964	1049	1133	1222
QIEA08	Best	46507	52127	57221	61294	67228	71600	77142
	Mean	46293	51816	57008	61063	66950	71308	76867
	Worst	46155	51618	56811	60894	66817	71121	76709
	Time	702	815	983	1706	1950	2230	2515
QEPS	Best	52772	58775	64513	70402	76621	81918	88207
	Mean	52600	58543	64230	70015	76245	81486	87657
	Worst	52395	58065	63680	69726	75296	80683	87044
	Time	464	523	605	1051	1170	1289	1441
PSMA	Best	53201	59091	64955	70244	76569	81806	87899
	Mean	53071	58968	64785	70134	76442	81664	87740
	Worst	52982	58819	64669	70024	76311	81505	87565
	Time	576	643	709	1156	1253	1352	1451

cant differences between the two pairs of algorithms, PSMA vs. QIEA02, QIEA04, QIEA07, QIEA08 and QEPS. The level of significance considered is 0.05. The results of Wilcoxon's and Friedman's tests are shown in Table 6. In Tables 5 and 6, The symbols "+" and "–" represent significant difference and no significant difference, respectively.

As shown in Tables 5 and 6, the t-test results demonstrate that there are significant differences between the two pairs of algorithms, PSMA vs. QIEA02, QIEA04, QIEA07, QIEA08 and QEPS. The p-values of the Wilcoxon's and Friedman's tests in Table 6 are far smaller than the level of significance 0.05, which indicates that PSMA really outperforms QIEA02, QIEA04, QIEA07, QIEA08 and QEPS.

Table 5. The results of t-test for the algorithms in Tables 3 and 4. The symbols "+" and "–" represent significant difference and no significant difference, respectively

PSMA vs.	QIEA02	QIEA04	QIEA07	QIEA08	QEPS
200 items	3.41e-14 (+)	4.57e-49 (+)	3.12e-24 (+)	1.41e-65 (+)	7.83e-08 (+)
400 items	6.71e-40 (+)	2.81e-49 (+)	9.19e-38 (+)	1.93e-81 (+)	1.81e-33 (+)
600 items	7.30e-36 (+)	6.41e-53 (+)	7.29e-35 (+)	1.99e-92 (+)	1.72e-43 (+)
800 items	4.39e-38 (+)	1.66e-60 (+)	7.09e-47 (+)	6.87e-83 (+)	1.24e-44 (+)
1000 items	5.93e-44 (+)	3.49e-77 (+)	1.71e-48 (+)	4.70e-94 (+)	1.77e-28 (+)
1200 items	1.67e-43 (+)	8.17e-64 (+)	7.48e-50 (+)	5.28e-89 (+)	4.92e-38 (+)
1400 items	5.58e-49 (+)	6.68e-66 (+)	3.65e-51 (+)	1.06e-92 (+)	8.74e-33 (+)
1600 items	4.75e-54 (+)	6.57e-82 (+)	6.17e-53 (+)	2.12e-98 (+)	4.43e-21 (+)
1800 items	1.43e-48 (+)	1.32e-71 (+)	1.61e-59 (+)	3.02e-98 (+)	1.06e-32 (+)
2000 items	3.98e-50 (+)	5.76e-73 (+)	6.57e-63 (+)	1.23e-94 (+)	3.71e-19 (+)
2200 items	9.07e-51 (+)	2.02e-69 (+)	6.06e-59 (+)	4.94e-97 (+)	7.45e-22 (+)
2400 items	9.34e-50 (+)	8.21e-88 (+)	6.78e-63 (+)	7.97e-101 (+)	2.70e-03 (+)
2600 items	6.98e-62 (+)	3.20e-72 (+)	2.24e-62 (+)	4.07e-101 (+)	6.71e-04 (+)
2800 items	8.78e-59 (+)	3.91e-73 (+)	1.45e-58 (+)	7.47e-102 (+)	1.60e-03 (+)
3000 items	5.86e-64 (+)	9.18e-73 (+)	1.36e-64 (+)	1.08e-102 (+)	1.16e-01 (–)

Table 6. The p-values of Wilcoxon's and Friedman's tests for the algorithms in Tables 3 and 4. The symbol + represents significant difference

PSMA vs.	QIEA02	QIEA04	QIEA07	QIEA08	QEPS
Wilcoxon	6.1035e-5(+)	6.1035e-5(+)	6.1035e-5(+)	6.1035e-5(+)	6.1035e-5(+)
Friedman	0.0142(+)	0.0142 (+)	0.0142 (+)	0.0142 (+)	0.0142 (+)

4 Distribution System Reconfiguration

Power network reconfiguration is an important process in the improvement of operating conditions of a power system and in planning studies, service restoration and distribution automation when remote-controlled switches are employed [1, 15]. The optimal distribution system reconfiguration problem is to minimize the power loss of the system by changing the topology of distribution systems through altering the open/closed status of sectionalizing switches. Because there are many candidate-switching combinations in a distribution system, the distribution system reconfiguration is a complex combinatorial problem with a large number of integer and continuous variables and various constraints such as power flow equations, upper and lower bounds of nodal voltages, upper and lower bounds of line currents, feasible conditions in terms of network topology. As usual the problem can be formulated as a minimization cost function f [1, 23], i.e.,

$$\min f = \sum_{i=1}^{L} r_i \frac{P_i^2 + Q_i^2}{V_i^2} \tag{7}$$

Subject to

$$g(x) = 0 \tag{8}$$

$$V_{min} < V_n < V_{max} \tag{9}$$

$$I_i^{min} < I_i < I_i^{max} \tag{10}$$

$$\det(A) = 1 \text{ or } -1(\text{for radial systems}) \tag{11}$$

$$\det(A) = 0(\text{for not radial systems}) \tag{12}$$

where

f is the objective function (kW);
L is the number of branches;
P_i is the active power at sending end of branch i;
Q_i is the reactive power at sending end of branch i;
V_n is the voltage at node n;
I_i is the line current at branch i;
$g(x)$ is the power flow equations;
V_{min} and V_{max} are the lower and upper voltage limits, respectively;
I_i^{min} and I_i^{max} are the lower and upper current limits, respectively;
A is the bus incidence matrix;
r_i is the resistance of branch i.

The PSMA described above is used to solve the IEEE 33-bus and PG&E 69-bus distribution system reconfiguration problems. The IEEE 33-bus and PG&E 69-bus systems are shown in Fig. 5 and Fig. 6, respectively. Both of them are widely used as examples to test the performance of various optimization approaches. As shown in Fig. 5, the IEEE 33-bus system has 33 buses, 37 branches and 5 tie-lines. The normally open switches are 33, 34, 35, 36 and 37. The initial real power losses (before reconfiguration) are 202.68 kW. The PG&E 69-bus systems consists of 69 buses, 68 sectionalizing switches and 5 tie switches. The normally open switches are 69, 70, 71, 72 and 73. The initial real power losses (before reconfiguration) are 226.4419 kW. The algorithm for solving the distribution system reconfiguration problem by using PSMA is the same as the description in Section 2.3 except that Step 3 Evaluation considers (7) as the candidate solution criterion. In the experiment, PSMA uses 10 individuals as a population and 0.9 as the value of p_c. After 100 evolutionary generations, we obtain the optimal result reported in the literature. The experimental result of the IEEE 33-bus system is listed in Table 7, where results obtained by five optimization approaches, a heuristic approach [15], SA+TS [13], MTS [16], PSO [1] and ACO [23], reported in the recent literature, are also provided to be as a comparison. The experimental result from the PG&E 69-bus system is shown in Table 8, where ACS [8], HPSO [14], VSHDE [18] and ACO [23] are considered as benchmark optimization approaches.

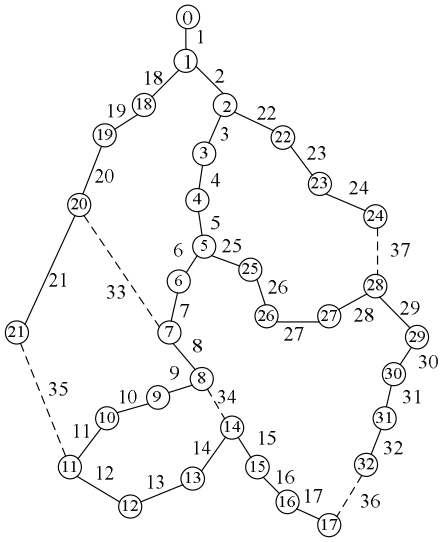


Fig. 5. IEEE 33-bus system

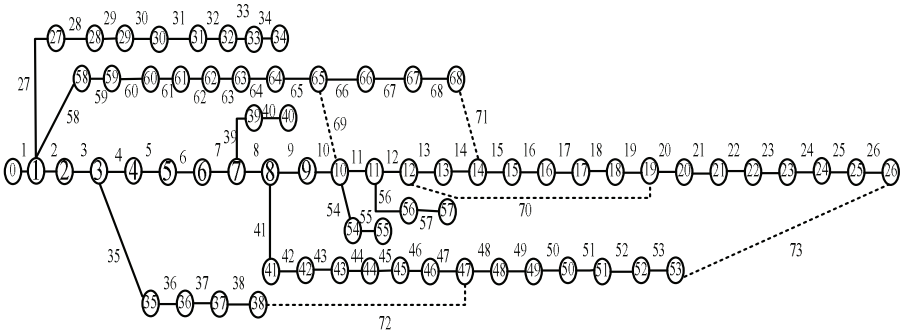


Fig. 6. PG&E 69-bus system

Table 7 shows that PSMA is competitive to the nine optimization approaches, a heuristic approach, SA+TS, MTS, PSO and ACO, due to the optimal solution obtained. The experimental results in Table 8 show that PSMA achieves lower real power losses and higher minimum node voltage than ACS, HPSO, VSHDE and IIGA. These results indicate that the better solution PSMA can obtain, the more complex the power system is.

Table 7. Results provided by PSMA for the IEEE 33-bus test system. MNV represents minimum node voltage

Methods	Optimal configuration	Real power loss (kW)	MNV (pu)
Before reconfiguration	33, 34, 35, 36, 37	202.68	0.9378
Heuristic approach [15]	7, 9, 14, 32, 37	139.55	0.9378
SA + TS [13]	7, 9, 14, 32, 37	139.55	0.9378
MTS [16]	7, 9, 14, 32, 37	139.55	0.9378
PSO [1]	7, 9, 14, 32, 37	139.55	0.9378
ACO [23]	7, 9, 14, 32, 37	139.55	0.9378
PSMA	7, 9, 14, 32, 37	139.55	0.9378

Table 8. Results provided by PSMA for the PG&E 69-bus test system. MNV represents minimum node voltage

Methods	Optimal configuration	Real power loss (kW)	MNV (pu)
Before reconfiguration	69, 70, 71, 72, 73	226.4419	0.9089
ACS [8]	61, 69, 14, 70, 55	99.519	0.943
HPSO [14]	69, 12, 14, 47, 50	99.6704	0.9428
VSHDE [18]	11, 24, 28, 43, 56	99.6252	0.9427
IIGA [26]	69, 14, 70, 47, 50	99.618	0.9427
PSMA	47, 12, 50, 14, 69	99.4944	0.9441

5 Conclusions

This paper is a continuous work on how to appropriately combine membrane computing models and evolutionary algorithms. This is the first attempt to use a population P system to design an approximate optimization algorithm. Extensively comparative experiments conducted on knapsack problems show that PSMA has a good performance with respect to the search capability and elapsed time. We also use PSMA to solve the distribution system reconfiguration problem in the area of power systems and experimental results are also attractive. Further work will focus on more and complex distribution system reconfiguration problems.

Acknowledgements

GZ and YQ acknowledges the support by the National Natural Science Foundation of China (61170016), the Program for New Century Excellent Talents in University (NCET- 11-0715) and SWJTU supported project (SWJTU12CX008), the Project-sponsored by SRF for ROCS, SEM, the Scientific and Technological Funds for Young Scientists of Sichuan (09ZQ026-040), the Fund for Candidates of Provincial Academic and Technical Leaders of Sichuan and the Fundamental Research Funds

for the Central Universities (SWJTU09ZT10). MAGN acknowledges the support of the project TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200. MG acknowledges the support of CNCSIS-UE-FISCSU project number PNII-IDEI 643/2008.

References

1. Abdelaziz, A., Mohammed, F., Mekhamer, S., Badr, M.: Distribution systems reconfiguration using a modified particle swarm optimization algorithm. Electric Power Systems Research 79(11), 1521–1530 (2009)
2. Bernardini, F., Gheorghe, M.: Population P systems. Journal of Universal Computer Science 10(5), 509–539 (2004)
3. Cao, H., Romero-Campero, F.J., Heeb, S., Cámara, M., Krasnogor, N.: Evolving cell models for systems and synthetic biology. Systems and Synthetic Biology 4(1), 55–84 (2010)
4. Cheng, J., Zhang, G., Zeng, X.: A novel membrane algorithm based on differential evolution for numerical optimization. International Journal of Unconventional Computing 7(3), 159–183 (2011)
5. Escuela, G., Gutiérrez-Naranjo, M.A.: An application of genetic algorithms to membrane computing. In: Martínez del Amor, M.A., Păun, Gh., Pérez Hurtado, I., Riscos-Núñez, A. (eds.) Eighth Brainstorming Week on Membrane Computing. pp. 101–108. Fénix Editora, Sevilla, Spain (2010)
6. García, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. Journal of Heuristics 15(6), 617–644 (2009)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
8. Ghorbani, M.A., Hosseinian, S.H., Vahidi B.: Application of ant colony system algorithm to distribution networks reconfiguration for loss reduction. In: Proceedings of the 11^{th} International Conference on Optimization of Electrical and Electronic Equipment, pp. 269–273 (2008).
9. Han, K.H., Kim, J.H.: Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE Transactions on Evolutionary Computation 6, 580–593 (2002)
10. Han, K.H., Kim, J.H.: Quantum-inspired evolutionary algorithms with a new termination criterion, H_ϵ gate, and two-phase scheme. IEEE Transactions on Evolutionary Computation 8(2), 156–169 (2004)
11. Huang, L., Suh, I.H.: Controller design for a marine diesel engine using membrane computing. International Journal of Innovative Computing, Information and Control 5(4), 899–912 (2009)
12. Huang, X.L., Zhang, G.X., Rong, H.N., Ipate, F.: Evolutionary design of a simple membrane system. In: Gheorghe, M., Păun, G., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) International Conference on Membrane Computing. Lecture Notes in Computer Science, vol. 7184, pp. 203–214. Springer (2011)

13. Jeon, Y.J., Kim, J.C.: Application of simulated annealing and tabu search for loss minimization in distribution systems. International Journal of Electrical Power & Energy Systems 26(1), 9–18 (2004)
14. Li, Z.K., Chen, X.Y., Yu, K., Sun, Y., Liu, H.M.: A hybrid particle swarm optimization approach for distribution network reconfiguration problem. In: Proceedings of Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, pp. 1–7 (2008).
15. Martín, J.A., Gil, A.J.: A new heuristic approach for distribution systems loss reduction. Electric Power Systems Research 78(11), 1953–1958 (2008)
16. Mekhamer, S., Abdelaziz, A., Mohammed, F., Badr, M.: A new intelligent optimization technique for distribution systems reconfiguration. In: Proceedings of the 12^{th} International Middle-East Power System Conference, 2008. MEPCON 2008. pp. 397–401. IEEE (2008)
17. Nishida, T.Y.: Membrane algorithm with brownian subalgorithm and genetic subalgorithm. International Journal of Foundations of Computer Science 18(6), 1353–1360 (2007)
18. Nournejad, F., Kazemzade, R., Yazdankhah, A.S.: A multiobjective evolutionary algorithm for distribution system reconfiguration. In: Proceedings of the 16_{th} Conference on Electrical Power Distribution Networks, pp. 1–7 (2011).
19. Suzuki, Y., Fujiwara, Y., Takabayashi, J., Tanaka, H.: Artificial life applications of a class of P systems: Abstract rewriting systems on multisets. In: Calude, C., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Workshop on Multiset Processing. Lecture Notes in Computer Science, vol. 2235, pp. 299–346. Springer, Berlin Heidelberg (2001)
20. Suzuki, Y., Tanaka, H.: Chemical evolution among artificial proto-cells. In: Bedau, M.A., McCaskill, J.S., Rasmussen, S. (eds.) Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life. pp. 54–63. MIT Press, Cambridge, MA, USA (2000)
21. Suzuki, Y., Tanaka, H.: Computational living systems based on an abstract chemical system. In: Proceedings of the 2000 Congress on Evolutionary Computation. pp. 1369–1376. IEEE (2000)
22. Suzuki, Y., Tanaka, H.: Modeling p53 signaling pathways by using multiset processing. In: Ciobanu, G., Păun, G., Pérez-Jiménez, M.J. (eds.) Applications of Membrane Computing, pp. 203–214. Natural Computing Series, Springer Berlin Heidelberg (2006)
23. Swarnkar, A., Gupta, N., Niazi, K.: Efficient reconfiguration of distribution systems using ant colony optimization adapted by graph theory. In: Power and Energy Society General Meeting, 2011 IEEE. pp. 1–8. IEEE (2011)
24. Tudose, C., Lefticaru, R., Ipate, F.: Using genetic algorithms and model checking for P systems automatic design. In: Pelta, D.A., Krasnogor, N., Dumitrescu, D., Chira, C., Lung, R.I. (eds.) NICSO. Studies in Computational Intelligence, vol. 387, pp. 285–302. Springer (2011)
25. Vlachogiannis, J., Lee, K.: Quantum-inspired evolutionary algorithm for real and reactive power dispatch. IEEE Transactions on Power Systems 23(4), 1627–1636 (2008)
26. Wang, C.X., Zhao, A.J., Dong, H., Li, Z.J.: An improved immune genetic algorithm for distribution network reconfiguration. In: Proceedings of International Conference on Information Management, Innovation Management and Industrial Engineering, 218–223 (2009).

27. Xiao, J.H., Zhang, X.Y., Xu, J.: A membrane evolutionary algorithm for DNA sequence design in DNA computing. Chinese Science Bulletin 57(6), 698–706 (2012)
28. Zhang, G.X.: Quantum-inspired evolutionary algorithms: a survey and empirical study. Journal of Heuristics 17(3), 303–351 (2011)
29. Zhang, G.X., Cheng, J.X., Gheorghe, M.: A membrane-inspired approximate algorithm for traveling salesman problems. Romanian Journal of Information Science and Technology 14(1), 3–19 (2011)
30. Zhang, G.X., Li, Y.Q., Gheorghe, M.: A membrane algorithm with quantum-inspired subalgorithms and its application to image processing, Natural Computing DOI: 10.1007/s11047-012-9320-2 (2012)
31. Zhang, G.X., Gheorghe, M., Wu, C.Z.: A quantum-inspired evolutionary algorithm based on P systems for knapsack problem. Fundamenta Informaticae 87(1), 93–116 (2008)
32. Zhang, G.X., Liu, C.X., Gheorghe, M.: Diversity and convergence analysis of membrane algorithms. In: Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE. pp. 596–603 (2010)
33. Zhang, G.X., Liu, C.X., Gheorghe, M., Ipate, F.: Solving satisfiability problems with membrane algorithms. In: Fourth International Conference on Bio-Inspired Computing, 2009. BIC-TA '09. pp. 29–36 (2009)
34. Zhang, G.X., Liu, C.X., Rong, H.N.: Analyzing radar emitter signals with membrane algorithms. Mathematical and Computer Modelling 52(11-12), 1997–2010 (2010)
35. Zhang, G.X., Liu, C.X., Gheorghe, M., Ipate, F.: An approximate algorithm using P systems with active membranes. Mathematics and Computers in Simulation (2012), Available: `http://staffwww.dcs.shef.ac.uk/people/M.Gheorghe/research/paperlist.html`
36. Zhang, G.X., Gheorghe, M., Cheng, J.X.: Dynamic behavior analysis of membrane algorithms. MATCH Communications in Mathematical and in Computer Chemistry (2012), Accepted paper.
37. Zhang, H., Zhang, G.X., Rong, H.N., Cheng, J.X.: Comparisons of quantum rotation gates in quantum-inspired evolutionary algorithms. In: Sixth International Conference on Natural Computation (ICNC), 2010. pp. 2306–2310. IEEE (2010)
38. Zhang, R., Gao, H.: Improved quantum evolutionary algorithm for combinatorial optimization problem. In: International Conference on Machine Learning and Cybernetics, 2007. pp. 3501–3505. IEEE (2007)

Author Index