Sixth Brainstorming Week on Membrane Computing

Sevilla, February 4–February 8, 2008

**Daniel Díaz-Pernil, Carmen Graciani,
Miguel Angel Gutiérrez-Naranjo, Gheorghe Păun,
Ignacio Pérez-Hurtado, Agustín Riscos-Núñez**

Editors

Sixth Brainstorming Week on Membrane Computing

Sevilla, February 4–February 8, 2008

**Daniel Díaz-Pernil, Carmen Graciani,
Miguel Angel Gutiérrez-Naranjo, Gheorghe Păun,
Ignacio Pérez-Hurtado, Agustín Riscos-Núñez**

Editors

RGNC REPORT 01/2008

**Research Group on Natural Computing
Sevilla University**

Fénix Editora, Sevilla, 2008

ISBN: 978-84-612-4429-4
Depósito Legal: SE-????–08
Edita: Fénix Editora
C/Patricio Sáenz 13, 1 – B
41004 Sevilla
fenixeditora1@telefonica.net
Telf. 954 41 29 91

Preface

This volume contains the papers emerged from the Sixth Brainstorming Week on Membrane Computing (BWMC), held in Sevilla, from February 4 to February 8, 2008, 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 the next four editions took place in Sevilla at the beginning of February 2004, 2005, 2006, and 2007, respectively.

In the style of previous meetings in this series, the sixth BWMC was again a period of active interaction among the participants, with the emphasis on exchanging ideas and cooperation, with only a few ("provocative", of any length between 5 and 55 minutes, depending on the interaction with audience) presentations scheduled in the first days of the meeting and with most of the time devoted to the joint work. The efficiency of this type of meetings was again proved to be very high and the present volume proves this assertion.

Among the features of the 2008 BWMC, we mention the usual high number of participants, the presentation of a PhD thesis in membrane computing (this was the case also in the last years), the presentation of new results, and, especially, of new applications.

The papers included in this volume, 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.

A selection of the papers from these volumes will be considered for publication in a special issues of *Fundamenta Informaticae*. After the first BWMC, a special issue of *Natural Computing* – volume 2, number 3, 2003, and a special issue of *New Generation Computing* – volume 22, number 4, 2004, were published; papers from the second BWMC have appeared in a special issue of *Journal of Universal Computer Science* – volume 10, number 5, 2004, as well as in a special issue

of *Soft Computing* – volume 9, number 5, 2005; a selection of papers written during the third BWMC have appeared in a special issue of *International Journal of Foundations of Computer Science* – volume 17, number 1, 2006); after the fourth BWMC a special issue of *Theoretical Computer Science* was edited – volume 372, numbers 2-3, 2007; after the fifth edition, a special issue of *International Journal of Unconventional Computing* was edited – now in press. Other papers elaborated during the sixth 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 Milano web page `http://psystems.disco.unimib.it` (with a mirror in China, at `http://bmc.hust.edu.cn/psystems`); soon, possibly before the publication of this volume, the page will have a new address, `http://ppage.psystems.eu`, a new look and a new host – Vienna.

The Sixth Brainstorming Week on Membrane Computing was dedicated to the memory of Nadia Busi, one of the most active researchers in membrane computing, a friendly, highly creative, and reliable member of our community, who passed away last year.

***

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

1. Ardelean Ioan I., Institute of Biology of the Romanian Academy, Romania, `ioan.ardelean@ibiol.ro`
2. Balbontín-Noval Delia, University of Sevilla, Spain, `delia@us.es`
3. Beyreder Markus, Technical University Wien, Austria, `e9526745@stud3.tuwien.ac.at`
4. Binder Aneta, Technical University Wien, Austria, `ani@logic.at`
5. Castellini Alberto, University of Verona, Italy, `alb.caste@gmail.com`
6. Ceterchi Rodica, University of Bucharest, Romania, `rceterchi@gmail.com`
7. Colomer M. Angels, University of Lleida, Spain, `Colomer@matematica.UdL.es`
8. Cordón Franco Andrés, University of Sevilla, Spain, `acordon@us.es`
9. Díaz-Pernil Daniel, University of Sevilla, Spain, `sbdani@us.es`
10. Ferretti Claudio, University of Milano-Bicocca, Italy, `ferretti@disco.unimib.it`
11. Franco Giuditta, University of Verona, Italy, `franco@sci.univr.it`
12. Freund Rudolf, Technical University Wien, Austria, `rudi@emcc.at, rudi@logic.at`

13. Frisco Pierluigi, Heriot-Watt University, United Kingdom,
pier@macs.hw.ac.uk
14. Gálvez-Santisteban Manuel Angel, University of Sevilla, Spain,
mangalsan@alum.us.es
15. Gheorghe Marian, University of Sheffield, United Kingdom,
marian@dcs.shef.ac.uk
16. Graciani Carmen, University of Sevilla, Spain,
cgdiaz@us.es
17. Gutiérrez-Naranjo Miguel Angel, University of Sevilla, Spain,
magutier@us.es
18. Hitzler Pascal, Univertity of Karlsruhe, Germany,
hitzler@aifb.uni-karlsruhe.de
19. Ishdorj Tseren-Onolt, Abo Akademi, Finland,
tishdorj@abo.fi
20. Leporati Alberto, University of Milano-Bicocca, Italy,
leporati@disco.unimib.it
21. Maggiolo-Schettini Andrea, University of Pisa, Italy,
maggiolo@di.unipi.it
22. Mauri Giancarlo, University of Milano-Bicocca, Italy,
mauri@disco.unimib.it
23. Milazzo Paolo, University of Pisa, Italy,
milazzo@di.unipi.it
24. Mira-Mira José, U.N.E.D., Spain,
jmira@dia.uned.es
25. Murphy Niall, NUI Maynooth, Ireland,
nmurphy@cs.nuim.ie
26. Obtułowicz Adam, Polish Academy of Sciences, Poland,
A.Obtulowicz@impan.gov.pl
27. Păun Gheorghe, Institute of Mathematics of the Romanian Academy, Romania, and University of Sevilla, Spain,
george.paun@imar.ro, gpaun@us.es
28. Pérez-Hurtado Ignacio, University of Sevilla, Spain,
perezh@us.es
29. Pérez-Jiménez Mario de Jesús, University of Sevilla, Spain,
marper@us.es
30. Ramírez-Martínez Daniel, University of Sevilla, Spain,
thebluebishop@gmail.com
31. Reyes-Jurado Fernando, University of Sevilla, Spain,
donepi@gmail.com
32. Riscos-Núñez Agustín, University of Sevilla, Spain,
ariscosn@us.es
33. Rivero-Gil Elena, University of Sevilla, Spain,
elen.rg@gmail.com

34. Rodrigues de Alcântara Júnior, Oséas, University of Sevilla, Spain, oseas@live.com
35. Romero-Campero Francisco José, University of Sevilla, Spain, fran@us.es
36. Romero Jiménez Alvaro, University of Sevilla, Spain, Alvaro.Romero@cs.us.es
37. Roselló Francesc, University of Balearic Islands, Spain, cesc.rossello@uib.es
38. Sempere José María, Polytechnical University of Valencia, Spain, jsempere@dsic.upv.es
39. Tomescu Alexandru I., University of Bucharest, Romania, alexandru.tomescu@gmail.com
40. Zandron Claudio, University of Milano-Bicocca, Italy, zandron@disco.unimib.it

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 Investigación TIN2006–13425 del Ministerio de Educación y Ciencia of Spain, (ii) Grupo de Investigación en Computación Natural (PAI TIC 193) de la Junta de Andalucía, (iii) Proyecto de Excelencia TIC–581 de la Junta de Andalucía, (iv) III Plan Propio de la Universidad de Sevilla, as well as by the Department of Computer Science and Artificial Intelligence from Sevilla University.

Gheorghe Păun
Mario de Jesús Pérez-Jiménez
(Sevilla, April 10, 2008)

Contents

A Biological Perspective on Sorting with P Systems

Ioan I. Ardelean[1,2], Rodica Ceterchi[⋆3], Alexandru Ioan Tomescu[3]

[1] Institute of Biology, Romanian Academy
Centre of Microbiology, Splaiul Independentei 296
P.O. Box 56-53, Bucharest 060031, Romania
[2] Ovidius University, Constanta, Romania
[3] Faculty of Mathematics and Computer Science, University of Bucharest
Academiei 14, RO-010014, Bucharest, Romania
E-mails: ioan.ardelean@ibiol.ro, rceterchi@gmail.com,
alexandru.tomescu@gmail.com

Summary. The aim of this contribution is to argue that the processes occurring in biological membranes in bacteria are also important as natural examples of communication between membranes, which, in the formal framework of P systems, leads (among other things) to simulations of sorting operations.

1 Introduction

Sorting is one of the most studied problem in Computer Science, as it has a wide range of applications, including sequential and parallel algorithms. Static sorting algorithms have been developed and proposed also in the P systems area. Among the first approaches, made independently, we mention [6] and [10], [11]. The problem of sorting with P systems occupies Chapter 8, [1], of the monograph [12].

The aim of this contribution is to show that the processes occurring in biological membranes in bacteria (cell membrane, external membrane or intracellular vesicles) which are essential for cell life being, are also important as natural examples of sorting processes occurring in bacteria. The argument is presented in Section 2. Accordingly, we have abstracted a formal model for a comparator of two values, which uses 3 membranes and communication rules between them. From the formal point of view, an immediate generalization follows, for N arbitrary values, whose biological feasibility remains to be investigated, and is probably heavily dependent on the value of N. Along the lines of previous work on sorting with membranes, a system is proposed to sort 4 values, by simulating a sorting network with 4 wires. This is the content of Section 3.

⋆ Corresponding author

2 Biochemical Reactions

Biochemical reactions in living cells occur following precise rules and laws; one rule shows that in a given chemical reaction there is a given numerical proportion between the reactants.

Respiration is the biological process that allows the cells (from bacteria to humans) to obtain energy. In short, respiration promotes a flux of electrons from electron donors to a final electron acceptor, which in most cases is molecular oxygen. The ability of many bacteria to use molecular oxygen as final electron acceptor in their respiration is provided by the work of an enzyme named: citocrom c oxidase which catalyzes the following equations:

$$O_{2\ in} + 4H^+_{in} + 4H^+_{in} + 4e^-_{within} = 2H_2O + 4H^+_{out}$$

The subscript "in" means on the inner face of the membrane, "out" the outer face of the membrane while "within" simply means within membrane.

Thus, during the last step of respiration shortly presented above, water is formed from molecular oxygen, protons ($4H^+$) and electrons($4e^-$). 4 protons are simultaneously transferred across membrane from inside to outside the cell contributing to energy conservation. Apart from its biological significance, the function of cy tocrom c oxidase could offer to P system scientists an example of a new type of developmental rule more complex that those already taken into account [18]. In a general formulation this rule is:

$$A_{in} + B_{in} + C_{within} = D_{in} + B_{out}$$

Moreover, coefficients before the symbols could be of help in establishing whether or not the function of cytocrom c oxidase could be used for sorting.

The overall process of photosynthesis as it occurs in cyanobacteria (as well as in algae and plants) consists in using electrons from water to ultimately reduce carbon dioxide thus forming substances such as carbohydrates. This process is essential for the life on Earth, being the main energy source for almost all living cells, including humans, the only source of molecular oxygen needed for respiration (and many oxygen-consuming related activities) as well as a huge carbon dioxide-consuming process.

The first major event in photosynthesis is the splitting of water at the expense of light energy to molecular oxygen, protons and electrons, which occurs at the level of intracitoplasmatoic vesicles called thylakoids. To be more precise, we focus on cyanobacteria which are Gram-negative bacteria, and we recall a few structural aspects of these bacteria which are relevant for sorting.

In Gram-negative bacteria, apart from the cell membrane (CM) covering the cytoplasm, there is a second membrane, called external membrane (EM), because it is located at the exterior of the cell membrane; the space between these two membranes is called periplasmic space.

The two membranes, EM and CM with a structure described by the fluid mosaic model, have different chemical composition and different particularization of the functions. When it comes to the transport of ions and molecules across them there are some important differences. Inogranic ions for example, can pass through the EM while there are special proteins and mechanisms controlling their passage across CM (see below).

Apart from CM and EM in some bacteria inside the cell there are some intracellular membranes (IM) organized in very tiny vesicles, these structures being associated with specific metabolic functions, photosynthesis being the most important.

Notwithstanding its biological significance, the splitting of water could offer to P system scientists an example of a new type of developmental rule more complex that those already taken into account, with applications, for example, in sorting with P systems. In photosynthesis, there is a movement of molecules and ions across the cytoplasmic membrane (skin membrane) the intrathylakoidal space to cytoplasm which could be interesting to be studied from the point of view of sorting.

For P systems symport and antiport rules are nice examples of how bacterial cells manage the developmental rules [18, 19]. In Bacteria these processes are needed to transport useful substances inside the cell and to transport outside the cell toxic substances, thus maintaining intracellular composition stable in a changing environment.

Another important transport system used by bacteria, is TRAP (tripartite ATP independent periplasmic) transporters [13]. Mechanistically TRAP systems are symporters transporting across CM protons and one solute (glutamate, for example). However, the significant differences with respect to a pure symport system, it that the transport of the solute takes place only when a specific periplasmic protein to bind the solute is present. So, there is the control of the transport of solute, carried out by a specific periplasmic protein that does not cross the CM! For P systems, the transport of a solute, chemical species controlled by another component, not passing through the membrane could be important for in vitro implementation of sorting.

The incorporation of different active (mainly) protein molecules in artificial membranes opens the possibility to move objects across these membranes. In our opinion, these experiments (with antiporters, symporters or any other active molecule biologically produced or chemically synthesized, but arranged in an appropriate way within the artificial membrane) could be useful for P systems for molecule sorting. Furthermore, in artificial membranes one could incorporate molecules which function as molecular logic gates such as those active in respiration [5]. Moreover, very recent results show that it is possible to improve the structure of artificial vesicle membranes by coating hollow polyelectrolyte capsules with biological interfaces such as phospholipid membrane and proteins [15], a step toward an artificial cell assembly [16]. The results in artificial membrane

research support the hope that they are appropriate tools for sorting experiments proposed in this contribution.

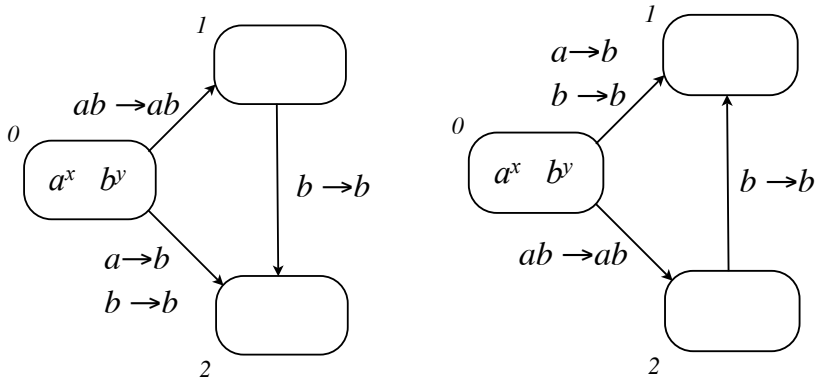
3 The Abstract Model for Sorting

Comparison-based sorting has been previously addressed in the field of P systems [1, 9]. Although it is well known that such sequential sorting algorithms require at least $N \log N$ comparisons to sort N items, performing many comparisons in parallel can reduce the sorting time. For example, the bitonic sorting algorithm proposed by Batcher [7] has complexity $O(\log^2 N)$. The intrinsic parallelism of P systems leads to a natural adaptation of classical parallel algorithms, which has been exploited in [9]. The P system consisted of a 2D-mesh of $\sqrt{N} \times \sqrt{N}$ membranes which were used to route values, but also to compare values.

We are concerned in this section with the crucial step of constructing a comparator of two values, which can serve as a building block for a P system which can sort N values. However, we do this by keeping in mind the biological processes and biochemical reactions illustrated in Section 2.

The formalism we adopt is that of a P system with dynamic communication [8, 10], along the same general lines as the model proposed in [9]. The computation takes place according to a finite sequence R_μ of pairs $[E, rules]$. The rules are applied on the set E of directed edges between membranes.

The comparator P system we propose is illustrated in Figure 2.



(a) Increasing comparator (b) Decreasing comparator

Fig. 1. A P system which sorts numbers x and y codified as occurrences of symbols a and b in membrane 0.

The formal definition of the P system which sorts ascending (Figure 1(a)) is the following:

$$\Pi =< V = \{a, b\}, \langle [a^x, b^y]_1, []_2, []_3 \rangle,$$
$$R_\mu = [\{(0,1)\}, rules(0,1) = \{ab \to ab\}]\cdot$$
$$\cdot[\{(0,2),(1,2)\}, rules(0,2) = \{a \to b, b \to b\}, rules(1,2) = \{b \to b\}] > .$$

At the beginning of the computation, x copies of a and y copies of b are loaded in membrane 0. The first pair of R_μ sends $\min(x, y)$ copies of ab into membrane 1. In the next step, the b symbols from membrane 1 are sent to membrane 2, while the remaining $\max(x, y) - \min(x, y)$ symbols of membrane 0 are rewritten to b and sent to membrane 2. At this point we have obtained $\min(x, y)$ as the number of occurrences of symbol a in membrane 1, and $\max(x, y)$ as number of occurrences of symbol b in membrane 2.

In our opinion the biological implementation probably could be done by the use of artificial lipid membranes in which appropriate membrane transporters (symporters, antiporters, ABC transporters, etc) have been included.

In the following, we put forward the design of a biological experiment which could in vitro implement such a comparator. The only modification of the proposed P system is that on edge $(0, 2)$ the rules are no longer rewriting rules, but simply: $a \to a,\ b \to b$. In this case, we still obtain the maximum in membrane 2, except that now it is codified as the total number of occurrences of both symbols a and b.

The proposal comprises the following biological-biochemical-biophysical steps:

i) in membrane 0 a specific type of enzyme which links one occurrence of a with one occurrence of b, thus forming one occurrence of ab; the process is repeated until the number of ab occurrences equals the $\min(x, y)$;
ii) in membrane 0 a uniporter protein transports all the occurrences of ab in membrane 1;
iii) in membrane 1, a specific enzyme split all the occurrences of ab in a and b;
iv) in membrane 1, a membrane carrier, an antiporter, sends all the occurrences of b in membrane 2;
iv')concurrently with step iv), in membrane 0 another membrane carrier, sends all remaining, unpaired occurrences of a or b in membrane 2, where there are already the bs arrived from membrane 1.

The particular biochemical nature of a and b, and transporters is in work. However is has to be said that the spatial relationships between these three types of membranes is important for computation and sorting.

A first generalization of this comparator to sort N numbers can be obtained by using $N + 1$ membranes. In membrane 0, number x_i is codified as numbers of occurrences of symbol a_i, with $1 \leq i \leq N$. In the first N steps, only one of the edges linking membranes 0 with 1, until 0 with N is active. The edge between membranes

0 and 1 contains the rule $a_1 \dots a_N \to a_1 \dots a_N$, the edge between membranes 0 and 2 contains all rules $\{a_1 \dots a_{i-1} a_{i+1} \dots a_N \to a_2 \dots a_N \mid 1 \leq i \leq N\}$, and so on. Finally, on the edge linking membrane 0 and membrane N the rules are $\{a_i \to a_N \mid 1 \ \leq i \leq N\}$. In the meantime, we also send from membrane 0 down to membrane N all symbols $a_2 \dots a_N$, $a_3 \dots a_N$, and so on, the only symbol being sent between membranes $N-1$ and N being a_N. Having specified how to generate the rules attached to edges, the sequence of active edges has length N and is the following:

$$\{(0,1)\}, \{(0,2),(1,2)\}, \{(0,3),(2,3)\}, \cdots \{(0,N),(N-1,N)\}.$$

In the end, after N steps, the sequence of numbers codified as occurrences of symbols in membranes 1 to N is increasing.

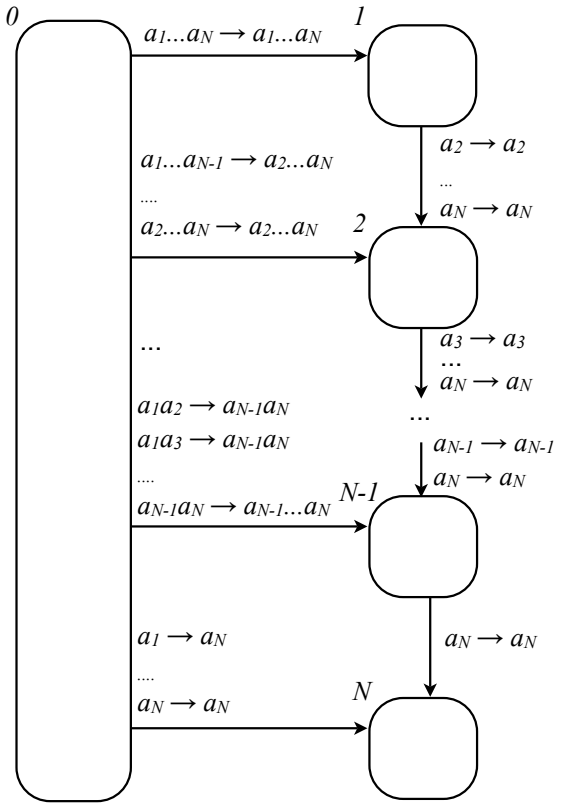


Fig. 2. A P system which sorts N numbers codified as occurrences of symbols $a_1 \dots a_N$ in membrane 0.

As mentioned before, one of the fastest parallel sorting algorithm is the bitonic sorting network. Following [14] it is customary to represent a network as an ordered set of N *lines* (wires) connected by a set of compare-exchange devices (*comparators*, for brevity). A comparator has two input terminals, a and b, and produces two output terminals c and d. If the comparator is increasing, then $c = \min(a, b)$ and $d = \max(a, b)$, while if the comparator is decreasing, $c = \max(a, b)$ and $d = \min(a, b)$. A bitonic sorting network for $N = 4$ is represented in Figure 3.

As we have built comparators of two elements, one can think of replacing each comparator of the bitonic network with the equivalent P system, and then connect the P systems according to the topology of the network. The rules according to these communication edges simply send all the symbols from the output membranes of the previous comparators to membrane 0 of the following one.

The P system described in Figure 4 has the following formal presentation:

$$\Pi_4 =< \{a, b, c, d\}, [a^x c^z]_0, []_1, []_2, [b^y d^t]_3, []_4, []_5, \cdots, []_{17}, R_\mu > .$$

The sequence R_μ of pairs $[graph, rules]$ is given below, where by $\cdot$ we denote sequential composition, and pairs grouped in the same set act in parallel.

$$R_\mu = \{[(0,1), ac \to ac][(3,5), bd \to bd]\}\cdot$$

$$\cdot\{[(1,2), c \to c], [(0,2), a \to c, c \to c], [(3,4), b \to d, d \to d], [(5,4), d \to d]\}\cdot$$

$$\cdot\{[(1,6), a \to a], [(2,9), c \to c], [(4,6), d \to d], [(5,9), b \to b]\}\cdot$$

$$\cdot\{[(6,7), ad \to ad], [(9,10), bc \to bc]\}\cdot$$

$$\cdot\{[(6,8), a \to d, d \to d], [(7,8), d \to d], [(9,11), b \to c, c \to c], [(10,11), c \to c]\}\cdot$$

$$\cdot\{[(7,12), a \to a], [(8,15), d \to d], [(10,12), b \to b], [(11,15), c \to c]\}\cdot$$

$$\cdot\{[(12,13), ab \to ab], [(15,16), cd \to cd]\}\cdot$$

$$\cdot\{[(12,14), a \to b, b \to b], [(13,14), b \to b], [(15,17), c \to d, d \to d], [(16,17), d \to d]\}.$$

Thus, after 8 steps, the four numbers $\{x, y, z, t\}$ are sorted in ascending order in membranes labeled $\{13, 14, 16, 17\}$, codified with symbols $\{a, b, c, d\}$ respectively.

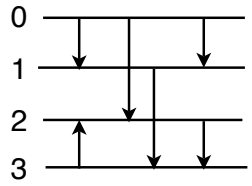


Fig. 3. A bitonic network of size $N = 4$.

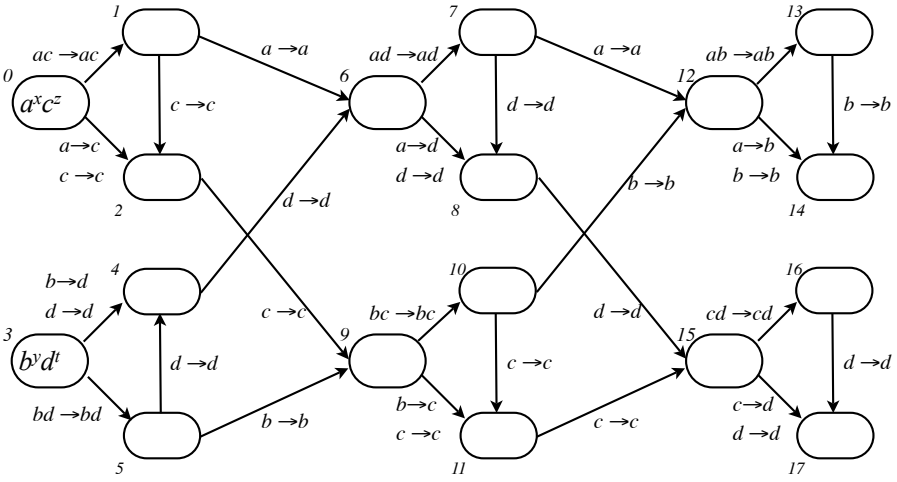


Fig. 4. A P system obtained from the bitonic sorting network of size 4 (Figure 3), in which each comparator has been replaced by a corresponding P system.

References

1. A. Alhazov, D. Sburlan: Static sorting P systems, Chapter 8 in Applications of Membrane Computing (G. Ciobanu, Gh. Păun, M.J. Pérez Jiménez Eds.), Springer, 2005.
2. I.I Ardelean: The relevance of cell membranes for P systems. General aspects, Fundamenta Informaticae, 49, 1-3, pp. 35-43, 2002.
3. I.I. Ardelean: Molecular biology of bacteria and its relevance for P systems, Membrane Computing, LNCS 2597 (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), pp. 1-19, 2003.
4. I.I. Ardelean, Biological roots and applications of P systems. Further suggestions, H.J. Hoogeboom et al. (Eds.), LNCS 4361, pp. 117, 2006.
5. I.I. Ardelean, D. Besozzi, C. Manara: Aerobic respirations a bio-logic circuit containing molecular logic gates. Pre-Proc. of Fifth Workshop on Membrane Computing, WMC5 (G. Mauri, Gh. Păun, C. Zandron, eds.), Universita di Milano-Bicocca, June 14-16, pp. 119125, 2004.
6. J.J. Arulanandham: Implementing bead–sort with P systems, Unconventional Models of Computation 2002 (C.S. Calude, M.J. Dinneen, F. Peper Eds.), LNCS 2509, pp. 115-125, 2002.
7. K. Batcher: Sorting networks and their applications, Proc. of the AFIPS Spring Joint Computing Conf. 32, p. 307-314, 1968.
8. R. Ceterchi, M.J. Pérez Jiménez: On two-dimensional mesh networks and their simulation with P systems, LNCS 3365, pp. 259-277, 2005.
9. R. Ceterchi, M.J. Pérez Jiménez, A.I. Tomescu: Simulating the bitonic sort using P systems, G. Eleftherakis et al. (Eds.), WMC8 2007, LNCS 4860, pp. 172-192, 2007.
10. R. Ceterchi, C. Martín-Vide: Dynamic P systems, LNCS 2597, pp. 146-186, 2003.
11. R. Ceterchi, C. Martín-Vide: P Systems with communication for static sorting, GRLMC Report 26 (M. Cavaliere, C. Martín–Vide, Gh. Păun, eds.), Rovira i Virgili University, Tarragona, 2003.

12. G. Ciobanu, Gh. Păun, M.J. Pérez Jiménez (Eds.): Applications of Membrane Computing, Springer, 2006.
13. D.J.Kelly, G.H. Thomas: The tripartite ATP-independent periplasmic (TRAP) transporters of bacteria and archaea, FEMS Microbiol. Rev. 25, pp. 404-424, 2001.
14. D.E. Knuth: The art of computer programming, volume 3: sorting and searching, second ed. Redwood City, CA: Addison Wesley Longman, 1998.
15. S.E. Moya, J.L. Toca-Herrera: From hollow shells to artificial cells: biointerface engineering on polyelectrolyte capsules. J. Nanosci. Nanotechnol., vol. 6, pp. 19, 2006.
16. V. Noireaux, A. Libchaber: A vesicle bioreactor as a step toward an artificial cell assembly. PNAS, 101, pp. 1766917674, 2004.
17. A. Ottova, H.T. Tien: The 40th anniversary of bilayer lipid membrane research. Bioelectrochemistry, 56, pp. 171173, 2002.
18. Gh. Păun: Computing with membranes (P systems): solving SAT in linear time, Proc. Rom. Acad., series A, 1, pp. 59-63, 2000.
19. Gh. Păun: From cells to computers: computing with membranes (P systems), Biosystems, 59, pp.139-158, 2001.

Membrane Proteins as Maxwell's Demons and Their Significance for P Systems

Ioan I. Ardelean[1,2], Cristina Moisescu[1], Mihaela Marilena Lăzăroaie[1]

[1] Institute of Biology of the Romanian Academy
Centre of Microbiology
296 Splaiul Independentei, P.O. Box 56-53, Bucharest 060031, Romania
[2] Ovidius University
124 Bulevardul Mamaia, Constanta, Romania
E-mails: ioan.ardelean@ibiol.ro, cristina.moisescu@ibiol.ro, mihaela.lazaroaie@ibiol.ro

Summary. The aim of these notes is to contribute to the dialog between P systems and Biological Sciences focussing on membrane proteins involved either in iron transport inside the bacterial cells or in elimination outside the bacterial cells of substances which are dangerous for the cell. The ability of these membrane proteins to behave as Maxwell's demon, gate keeper, or as "a being who can see the individual molecules" could be important for P systems as examples of discrete processes which could be modeled by discrete mathematics and as real molecular objects for *in vitro* implementation of P systems.

1 Introduction

In the framework of the dialog between P systems and Biological Sciences, the aim of this communication is to develop the already increasing interest in Biological Sciences for membrane proteins acting as devices commonly known as Maxwell's demons (Hopfer, 2002; Otsuka and Nozawa, 1998).

The device originally suggested by Maxwell to separate molecules (actually low- and high-speed gas molecules) into different compartments has as a crucial element an intelligent gate keeper (Hopfer, 2002), called by Maxwell himself "a being, who can see the individual molecules" (Maxwell, 1871), a being which was called later on Maxwell's demon. The mechanism was originally described as follows: "Now let us suppose that such a vessel is divided into two portions, A and B, by a division in which there is a small hole, and that a being, who can see the individual molecules, opens and closes this hole, so as to allow only the swifter molecules to pass from A to B, and only the slower molecules to pass from B to A (Maxwell, 1871).

In this paper, in order to contribute to the dialog between P systems and Biological Sciences, we will focus on membrane proteins involved in iron transport

inside the bacterial cells and on other membrane proteins involved in elimination outside the bacterial cells of substances which are dangerous for the cell (antibiotics and solvents), the so-called multidrug resistance (MDR) efflux systems.

We also put forward the question if Maxwell could be seen as a precursor of membrane computing because he imagined nanosized devices working at the boundary between two compartments.

These membrane proteins involved in iron transport inside the bacterial cells and in the elimination outside the bacterial cells of substances which are dangerous for the cell (antibiotics and solvents) could have significance for P systems in the followings:

a) as examples of discrete processes which could be modeled by discrete mathematics;
b) as real molecular objects for *in vitro* implementation of P systems.

2 Membrane Proteins Involved in Iron Transport Inside the Bacterial Cell

Iron is a crucial microelement in microbial metabolism, playing a vital role in many important biological processes such as respiration and biomineralization. Biomineralization, the process by which organisms transforms soluble substances into mineral crystals, plays a significant role in environmental iron cycling, the magnetization of sediments and thus the geologic record, and in the use of biomarkers as microbial fossils (Bazylinski et al., 2007; Simmons et al., 2004; Stolz, 1990; Petersen et al., 1986). Magnetotactic bacteria (MTB) can be considered as a model system for biomineralization of iron oxide and sulfide nanocrystals produced by living organisms. MTB are a fascinating group of microorganisms (Blakemore, 1979; Schüler and Baeuerlein, 1998; Ignat et al., 2007; Ardelean et al., 2008; Logofătu et al., 2008), which exhibit the peculiar ability to orient themselves along the magnetic field lines of Earth's magnetic field. The sensitivity of MTB to the Earth's magnetic field arises from the fact that the bacteria precipitate within their cells chains of lipid membrane-enclosed crystals of magnetic minerals magnetite (Fe_3O_4), greigite (Fe_3S_4), or both, referred to as "magnetosomes", which serve as a navigational device for spatial orientation in marine and freshwater habitats. In MTB, a magnetic particle synthesis system was proposed (Mann et al., 1990), which involves the following discrete processes occurring across and within biological membranes: (i) uptake of iron, (ii) transport of iron to the cytoplasm and across the magnetic particle membrane, (iii) precipitation of hydrated ferric oxide within vesicles, and (iv) phase transformation of the amorphous iron phase to magnetite, during both nucleation and surface-controlled growth.

Recent molecular studies have postulated the steps of bacterial magnetic particles (BMP) synthesis in *Magnetospirillum magneticum* strain AMB-1 (Okamura et al., 2001; Matsunaga et al., 2000; Nakamura et al., 1995a). The presentation of these steps in some detail could help the mathematicians and informaticians

to deeper accommodate with the progresses at molecular level in this hot topic of nowadays Microbiology and, hopefully, to find inspiration for other application of these proteins in P systems, than those proposed by us, pure microbiologists, in this communication.

1. The first event of BMP synthesis is the formation of vesicles (magnetosome membrane), which further argues the role of membranes in biomineralization. Okamura et al. (2001) identified a 16 kDa protein, called Mms16, which was the most abundant of the magnetosome specific proteins in *M. magneticum* AMB-1. The Mms16 protein was confirmed to be expressed only on the BMP membrane, thus Okamura and coworkers were the first to report the experimental function of a BMP-specific protein. Their results also suggested that this novel Mms16 protein, specifically localized on the magnetic particle membrane, is a GTPase, being able to split GTP, a compound rich in useful energy for the cell. The proposed mechanism supposes that the Mms16 first binds with the cytoplasmic membrane. This binding serves to prime the invagination of the cytoplasmic membrane for the intracellular vesicles formation, which will become the future BMP membrane. Acyl-CoA and GTP hydrolysis might be required during vesicle budding. Therefore, another magnetosome specific protein, called MpsA, a homolog of an acyl-CoA carboxylase (transferase) containing a CoA-binding motif, is also considered to be involved in this process (Matsunaga et al., 2000) functioned as mediator for BMP membrane invagination (Matsunaga et al., 2000), but its exact function remains unclear.

2. The second process in BMP synthesis is iron transport (see Figure 1). Iron exists in two redox states: the reduced Fe^{2+} ferrous state and the oxidized Fe^{3+} ferric form. In natural environments, iron predominantly occurs as ferric iron (Fe^{3+}) under aerobiosis. Fe^{3+} as iron hydroxide is poorly soluble in aqueous solution, rendering it basically unavailable for the cells (Neilands, 1981). Under anaerobiosis, reducing or acidic conditions, the iron equilibrium shifts from the ferric Fe^{3+} to the ferrous Fe^{2+} form that is more easily available for microorganisms. Thus, several aspects of the general system needs to be made clearer, for example, whether the ferric or ferrous ion is taken up and transported and which proteins control the reactions in each stage. The studies of Suzuki et al. (2006) showed that a robust ferrous ion-uptake system coupled to magnetosome synthesis exist within *M. magneticum* AMB-1. It appears that in *M. magneticum* strain AMB-1 ferric iron is reduced on the cell surface, taken as ferrous iron into the cytoplasm, transported into the BMP vesicle, and finally oxidized to produce magnetite (Arakaki et al., 2003) which is an interesting system to be modeled within the framework of membrane computing. The later studies of Suzuki et al. (2007) also revealed that the activity of this ferrous ion-uptake system is modulated by a cytoplasmic ATPase (gene product of ORF4 operon), which accelerates the uptake of ferrous ions through the cell membrane into the cytoplasm by energizing cell membrane ferrous ion transporters FeoAB, Tpd and Ftr1 (see Figure 1).

There are several families of proteins involved in iron transport in Prokaryotes. These families are grouped in two main groups: ferrous ion (Fe^{2+}) transporters,

and ferric ion (Fe^{3+}) transporters. From the first group, the following genes (and consequently the proteins encoded by these genes) are present in *M. magneticum* AMB-1: *ftr1, tpd, feoA*, and *feoB*. From the later group, the *cirA, fepA, fepC, tonB, exbB, exbD, tolQ, napA, napB*, and *napC* genes (respectively proteins) are present in *M. magneticum* AMB-1 (Suzuki et al., 2006).

The *ftr, tpd,* and *feo* genes are known to be expressed under low-oxygen conditions when ferrous iron remains stable and predominates over ferric iron (Andrews et al., 2003; Dubbels et al., 2004; Felice et al., 2005; Kammler et al., 1993; Marlovits et al., 2002). The studies of Suzuki et al. (2006) also revealed that in *M. magneticum* AMB-1 the ferrous iron transport system is triggered under reducing conditions (with low-oxygen levels) these results being consistent with the microaerobic culture conditions in which the cells were grown. On the other hand, ferric ions transport genes, which include *fepA, tonB, exbB*, and *exbD* (Andrews et al., 2003), were downregulated under iron-rich conditions. Additionally, higher transcript levels of nitrate reductase (amb2686, amb2687, amb2690) and ferric reductase (amb3335) genes were obtained under iron-rich, magnetosome-forming conditions (Suzuki et al., 2006).

3. The last process is crystallization of magnetite within the intracellular vesicles (magnetosome membranes). Several proteins appear to be required for magnetite crystallization and the first reported protein was MagA, isolated from *M. magneticum* AMB-1 (Nakamura et al., 1995a). Internal localization analysis of the MagA protein indicated that, unlike Mms16 protein, MagA is localized on both cytoplasmic membrane (Nakamura et al., 1995b), and BMP membrane and showed iron transport activity. Interestingly, MagA topology is inversely oriented between the cytoplasmic membrane and the BMP membrane (Nakamura et al., 1995a; 1995b), something which would occur if magnetosomes were formed by membrane invagination. MagA appears to function for iron efflux in the former and iron influx in the latter. The number of MagA molecules per magnetosome volume is much larger than per cell volume as calculated from the total amount of expressed MagA (Nakamura et al., 1995b). This makes the quantity of effluxed iron by MagA on the cytoplasmic membrane negligible. The iron-uptake activity of MagA was determined using inverted vesicles prepared from fragmented membrane-expressing MagA protein in *E. coli*. Addition of ATP initiated the accumulation of ferrous ions in the vesicles. The ions were released by the addition of carbonyl cyanide m-chlorophenylhydrazone (CCCP), also known as protonophore (Nakamura et al., 1995a). This activity was also observed under an artificial proton gradient without ATP. These results suggest that MagA protein is a proton-driving H^+/ Fe^{3+} antiporter (Matsunaga et al., 2000). MagA protein may play roles in transporting Fe^{3+} to the vesicle to grow up to BMPs and the alkalization of the inside of vesicles due to its H^+/ Fe^{3+} antiporter function.

Furthermore, Mms6 protein isolated from BMP membrane was shown to function for the crystallization of ferric and ferrous ions under anaerobic conditions (Arakaki et al., 2003). Magnetite crystallization is inorganically derived, as demonstrated by Frankel et al. (1983), but Mms6 directly binds ferric iron and regulates

crystallization and morphology during magnetite formation in *M. magneticum* AMB-1 (Arakaki et al., 2003), acting as an organic matrix for crystal formation.

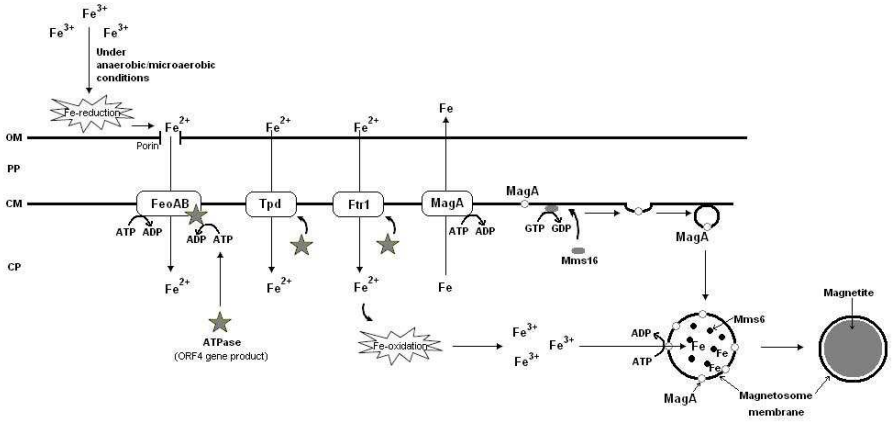


Fig. 1. Schematic representation of the possible ferrous ion uptake system during magnetosome synthesis within *M. magneticum* AMB-1 strain (adapted after Suzuki et al., 2007; Matsunaga et al., 2004). The Mms16 protein attaches to the CM initiating the intracellular vesicle formation (first step in magnetosome synthesis). Under the microaerobic growth conditions, the extracellular Fe^{3+} is reduced on the cell surface to Fe^{2+}, thus making the iron available for the microorganism. An ATPase (the ORF4 gene product) present in the cell cytoplasm, accelerates the uptake of the Fe^{2+} ions through the cellular membrane into the cytoplasm, by energizing cell membrane Fe^{2+} transporters FeoAB, Tpd, and Ftr1. The transported Fe^{2+} is then oxidized to Fe^{3+} in the cytoplasm, and transported into the intracellular vesicles (the future magnetosome membrane) by the MagA, a H^+/Fe^{3+} antiporter protein. The Mms6 protein present inside the magnetosme vesicles binds Fe^{3+}, and acts as an organic matrix for magnetite crystal formation in *M. magneticum* AMB-1. OM, outer membrane; PP, periplasm; CM, cytoplasmic membrane; CP, cytoplasm.

3 Multidrug Resistance (MDR) Efflux Systems

The efflux of the hydrocarbons by multidrug resistance (MDR) efflux systems is the most important mechanism of hydrocarbons tolerance in bacteria used for maintaining the hydrocarbons concentration in the cell under its equilibrium level. Together with MDR efflux systems microorganisms can use other mechanisms to resist toxic hydrocarbons such as: metabolism of toxic hydrocarbons, which can contribute to their transformation into nontoxic compounds; rigidification of the cell membrane via alteration of the phospholipids composition; alterations in the cell surface that make the cells less permeable; formation of vesicles that remove the

solvent from the cell surface; and efflux of hydrocarbons in an energy-dependent process (Ramos et al., 2002; Segura et al., 1999, 2007).

MDR efflux systems catalyze the active extrusion of many structurally and functionally related and unrelated compounds from the bacterial cytoplasm (or internal membrane) to the external medium (Segura et al., 1999; Ramos et al., 2002). Some of the substrates of these MDR pumps are hydrocarbons that do not resemble any of the known natural substrates that these cells may have encountered during evolution. The data available indicate that it is the physical characteristics of the compounds (e.g., charge, hydrophobicity or amphipathicity), the Van der Waals interactions they establish with active sites and effectors pockets, and the flexibility of these sites in the target proteins that determine the specificity of these multidrug efflux systems (Neyfakh, 2001).

Multidrug efflux systems have been the subject of recent reviews, and five main families of MDR transporters have been identified in bacteria. These include: 1) the major facilitator superfamily (MFS); 2) the ATP-binding cassette (ABC) family; 3) the resistance nodulation and cell division (RND) family, which is part of the larger RND permease superfamily; 4) the small multidrug resistance (SMR) family; and 5) the multidrug and toxic compound extrusion (MATE) family. The most well characterized representatives of these families from Gram-positive and Gram-negative bacteria are shown in Figure 2A and 2B, respectively. Small multidrug resistance (SMR) and multidrug and toxic compound extrusion (MATE) look structurally similar with the major facilitator superfamily (MFS) but are designed as distinct families, based on size (i.e., SMR) or phylogenetic diversity (i.e., MATE). All of these transporters catalyze active drug efflux and therefore require energy, mostly in the form of proton motif force (i.e., utilize the H^+ or Na^+ transmembrane electrochemical gradient for pumping the antibiotics or other compounds from the inner to the outer space of the cell), or in the form of ATP (i.e., utilize the release of phosphate bond-energy by ATP hydrolysis for pumping the antibiotics or other compounds from the inner to the outer space of the cell) (Putman et al., 2000; Schweizer, 2003).

The efflux pumps for antibiotics and hydrocarbons work with exceptional efficiency in Gram-negative bacteria due to synergistic action of cytoplasmic membrane with outer membrane. In Gram-positive bacteria, the efflux pumps move the substrate across just one membrane. This is rather inefficient, as they have to compete with the rapid spontaneous influx of the lipophilic molecule back into the cytoplasm. A high rate of efflux is therefore required to produce significant levels of resistance. The efflux pumps in the Gram-negative bacteria traverse both the cytoplasmic and outer membranes. As the outer membrane is composed largely of lipopolysaccharides (LPS), it has different permeability properties compared to the membrane of Gram-positive bacteria. The membrane of Gram-negative bacteria allows the penetration of lipophilic molecules but at a rate 50-100 times slower than the phospholipid bilayer of Gram-positive bacteria. The decrease in penetration of lipophilic molecules is responsible for the intrinsic resistance of Gram-negatives to certain antibiotics and hydrocarbons. Hydrophilic molecules enter in the mem-

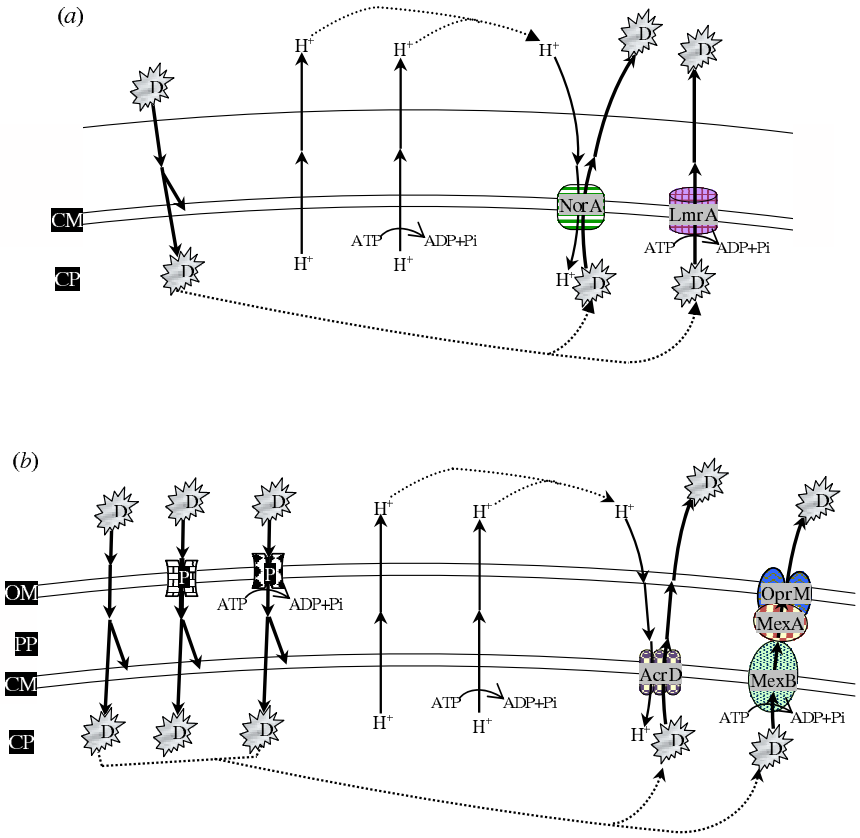


Fig. 2. Schematic illustration of the main types of bacterial drug efflux pumps in Gram-positive (a) and Gram-negative (b) bacteria. (a) Illustrated are NorA, a member of the major facilitator superfamily (MFS); LmrA, a member of the ATP-binding cassette (ABC) family; (b) AcrD and MexAB-OprM, two memers of resistance nodulation and cell division (RND) family. D, drug; OM, outer membrane; PP, periplasm; CM, cytoplasmic membrane; CP, cytoplasm; P, porin.

brane of Gram-negative cells through porins, but in the presence of hydrophilic molecules (e.g., antibiotics, hydrocarbons) or when efflux mechanisms are induced, a decrease in the number of porins in the membrane is also seen. This leads to decreased penetration of the hydrophilic molecules (Nikaido, 1998, 2003).

The intellectual framework of metabolic engineering is built upon the integration of biological information in an attempt to induce higher order principles that govern cell behavior. As such, metabolic engineering and the emerging field of systems biology share an over-arching emphasis on revealing general biological principles from the analysis of the regulation and activity of biological networks ranging from gene sequence to gene expression to metabolic flux (Stephanopoulos

and Gill, 2000). The engineering of new traits or re-engineering of existing traits, which is dependent upon such biological networks, is a major thrust of metabolic engineering. Specifically, metabolic engineering involves the modification of the genetic makeup of an organism in an attempt to re-direct cell behavior in a specific manner. This might involve, for example, engineering increased or decreased expression of a gene that is thought to influence production of a valuable chemical product. While such an example may appear to be straightforward, it is complicated by the fact that metabolism forms a network of chemical reactions that are mutually interdependent and that incommensurately influence overall network activity, which itself influences the relative fitness of an organism in a particular environment. Therefore, any attempt to engineer flux through a specific pathway in an organism can result in secondary effects that may include a reduction in the overall fitness of the organism and, as a result, reduce the attractiveness of the engineering strategy). Antibiotic resistance provides numerous examples of how nature has approached this problem. In particular, antibiotic resistance exemplifies how bacteria routinely develop new phenotypes through combinations of creative and hard to predict mechanisms and the importance that environment plays in selection and maintenance of such phenotypes. As such, it serves as a model to elucidate underlying evolutionary mechanisms that might be applied to the development of future metabolic engineering efforts (Bailey, 1991; Bailey, 1999; Bonomo and Gill, 2005).

In our opinion, the development of future metabolic engineering of MDR efflux systems would benefit if they would be studied in the framework of P systems which could deeply take into account the discrete nature of these proteins working at/within the biological membranes

References

1. S.C. Andrews, A.K. Robinson, F. Rodriguez-Quinones: Bacterial iron homeostasis. *FEMS Microbiol. Rev.*, 27 (2003), 215–237.
2. A. Arakaki, J. Webb, T. Matsunaga: A novel protein tightly bound to bacterial magnetic particles in *Magnetospirillum magneticum* strain AMB-1. *J. Biol. Chem.*, 278 (2003), 8745–8750.
3. I. Ardelean, C. Moisescu, D.R. Popoviciu: Magnetotactic bacteria and their potential for terraformation. Accepted for publication in: *From Fossils to Astrobiology*, Springer, Series: *Cellular Origin, Life in Extreme Habitats and Astrobiology*, Eds. Joseph Seckbach and Maud M. Walsh.
4. J.E. Bailey: Toward a science of metabolic engineering. *Science*, 252 (1991), 1668–1675.
5. J.E. Bailey: Lessons from metabolic engineering for functional genomics and drug discovery. *Nat. Biotechnol.*, 17 (1999), 616–618.
6. D.A. Bazylinski, R.B. Frankel, K.O. Konhauser: Modes of biomineralization of magnetite by microbes. *Geomicrobiol. J.*, 24 (2007), 465–475.
7. R.P. Blakemore, D. Maratea, R.S. Wolfe: Isolation and pure culture of a freshwater magnetic spirillum in chemically defined medium. *J. Bacteriol.*, 140 (1979), 720–729.

8. J. Bonomo, R.T. Gill: Antibiotic resistance as a model for strain engineering. *Computers and Chemical Engineering*, 29 (2005), 509–517.
9. B.L. Dubbels, A.A. DiSpirito, J.D. Morton, J.D. Semrau, J.N. Neto, D.A. Bazylinski: Evidence for a copper-dependent iron transport system in the marine, magnetotactic bacterium strain MV-1. *Microbiology*, 150 (2004), 2931–2945.
10. M.R. Felice, I. De Domenico, L. Li, D.M. Ward, B. Bartok, G. Musci, J. Kaplan: Post-transcriptional regulation of the yeast high affinity iron transport system. *J. Biol. Chem.*, 280 (2005), 22181–22190.
11. R.B. Frankel, R.P. Blakemore, A.L. Mackay: Fe3O4 precipitation in magnetotactic bacteria. *Biochim. Biophys. Acta,* 763 (1983), 147–159.
12. U. Hopfer: A Maxwell:s demon type of membrane transport: Possibility for active transport by ABC-type transporters? *J. theor. Biol.*, 214 (2002), 539–547.
13. M. Ignat, G. Zarnescu, S. Soldan, I. Ardelean, C. Moisescu: Magneto-mechanic model of the magnetotactic bacteria. Applications in the microacuator field. *J. Optoel. Advanced Materials*, 4 (2007), 1169–1171.
14. M. Kammler, C. Schon, K. Hantke: Characterization of the ferrous iron uptake system of Escherichia coli. *J. Bacteriol.*, 175 (1993), 6212–6219.
15. P.C. Logofătu, I. Ardelean, D. Apostol, C. Moisescu, B. Ioniţă: Determination of the magnetic moment and geometrical dimensions of the magnetotactic bacteria using an optical scattering method. Accepted for publication in *Journal of Applied Physics.*
16. S. Mann, N.H.C. Sparks, R.G. Board: Magnetotactic bacteria: microbiology, biomineralization, palaeomagnetism and biotechnology. *Adv. Microbiol. Physiol.*, 31 (1990), 125–181.
17. T.C. Marlovits, W. Haase, C. Herrmann, S.G. Aller, V.M. Unger: The membrane protein FeoB contains an intramolecular G protein essential for Fe(II) uptake in bacteria. *Proc. Natl. Acad. Sci. USA*, 99 (2002), 16243–16248.
18. T. Matsunaga, Y. Okamura, T. Tanaka: Biotechnological application of nano-scale engineered bacterial magnetic particles. *J. Mater. Chem.*, 14 (2004), 2099–2105.
19. T. Matsunaga, N. Tsujimura, Y. Okamura, H. Takeyama: Cloning and characterization of a gene mpsA encoding a protein associated with intracellular magnetic particles from *Magnetospirillum sp.* strain AMB-1, *Biochem. Biophys. Res. Commun.*, 268 (2000), 932–937.
20. J.C. Maxwell: *Theory of Heat.* 1871, New York, Dover. Reprinted (2001).
21. C. Nakamura, J.G. Burgess, K. Sode, T. Matsunaga: An iron-regulated gene, magA, encoding an iron transport protein of Magnetospirillum magneticum strain AMB-1. *J. Biol. Chem.*, 270 (1995a), 28392–28396.
22. C. Nakamura, T. Kikuchi, J.G. Burgess, T. Matsunaga: Iron regulated expression and membrane localization of the magA protein in *Magnetospirillum magneticum* strain AMB-1. *J. Biochem.*, 118 (1995b), 23–27.
23. J.B. Neilands: Iron absorption and transport in microorganisms. *Annu. Rev. Nutr.*, 1 (1981), 27–46.
24. A.A. Neyfakh: The ostensible paradox of multidrug recognition. *J. Mol. Microbiol. Biotechnol.*, 3 (2001), 151–154.
25. H. Nikaido: Multiple antibiotic resistance and efflux. *Curr. Opin. Microbiol.*, 1 (1998), 516–523.
26. H. Nikaido: Molecular basis of bacterial outer membrane permeability revisited. *Microbiol. Mol. Biol. Rev.*, 67 (2003), 593–656.
27. Y. Okamura, H. Takeyama, T. Matsunaga: A magnetosome-specific GTPase from the magnetic bacterium *Magnetospirillum magneticum*AMB-1. *J. Biol. Chem.*, 276 (2001), 48183–48188.

28. J. Otsuka, Y. Nozawa: Self-reproducing system can behave as Maxwell's demon: Theoretical illustration under prebiotic conditions. *J. Theor. Biol.*, 194 (1998), 205–221.
29. N. Petersen, T. von Dobeneck, H. Vali: Fossil bacterial magnetite in deep-sea sediments from the South Atlantic Ocean. *Nature*, 320 (1986), 611–615.
30. M. Putman, H.W. van Veen, W.N. Konings: Molecular properties of bacterial multidrug transporters. *Microbiol. Mol. Biol. Rev.*, 64 (2000), 672–693.
31. J.L. Ramos, E. Duque, M.T. Gallegos, P. Godoy, M.I. Ramos-Gonzalez, A. Rojas, W. Teran, A. Segura: Mechanisms of solvent tolerance in Gram-negative bacteria. *Annu. Rev. Microbiol.*, 56 (2002), 743–768.
32. D. Schüler, E. Baeuerlein: Dynamics of iron uptake and Fe3O4 biomineralization during aerobic and microaerobic growth of Magnetospirillum gryphiswaldense. *J. Bacteriol.*, 180 (1998), 159–162.
33. H.P. Schweizer: Efflux as a mechanism of resistance to antimicrobials in *Pseudomonas aeruginosa* and related bacteria: unanswered questions. *Genet. Mol. Res.*, 2 (2003), 48–62.
34. A. Segura, E. Duque, G. Mosqueda, J.L. Ramos, F. Junker: Multiple responses of Gram-negative bacteria to organic solvents. *Environ. Microbiol.*, 1 (1999), 191–198.
35. A. Segura, A. Hurtado, B. Rivera, M.M. Lăzăroaie: Isolation of new toluene-tolerant marine strains of bacteria and characterization of their solvent-tolerance properties. *Journal of Applied Microbiology* (OnlineEarly Articles), (2007), Blackwell Publishing, doi: 10.1111/j.1365-2672.2007.03666.x.
36. S.L. Simmons, S.M. Sievert, R.B. Frankel, D.A. Bazylinski, K.J. Edwards: Spatiotemporal distribution of marine magnetotactic bacteria in a seasonally stratified coastal pond. *Appl. Environ. Microbiol.*, 70 (2004), 6230–6239.
37. G. Stephanopoulos, R.T. Gill: After a decade of progress, an expanded role for metabolic engineering. *Advances in Biochemical Engineering*, 73 (2000), 1–8.
38. J.F. Stolz: Distribution of phototrophic microbes in the stratified microbial community at Laguna Figueroa, Baja California, Mexico. *BioSystems*, 23 (1990), 345–357.
39. T. Suzuki, Y. Okamura, R.J. Calugay, H. Takeyama, T. Matsunaga: Global gene expression analysis of iron inducible genes in Magnetospirillum magneticum AMB-1. *J. Bacteriol.*, 188 (2006), 2275–2279.
40. T. Suzuki, Y. Okamura, A. Arakaki, H. Takeyama, T. Matsunaga: Cytoplasmic ATPase involved in ferrous ion uptake from magnetotactic bacterium *Magnetospirillum magneticum* AMB-1. *FEBS Letters*, 581 (2007), 3443–3448.

Towards a P Systems Normal Form Preserving Step-by-step Behavior

Roberto Barbuti[1], Andrea Maggiolo-Schettini[1], Paolo Milazzo[1], Simone Tini[2]

[1] Dipartimento di Informatica, Università di Pisa
Largo Pontecorvo 3, 56127 Pisa, Italy
[2] Dip. di Scienze della Cultura, Politiche e dell'Informazione
Università dell'Insubria
Via Valleggio 11, 22100 Como, Italy

Summary. Starting from a compositional operational semantics of transition P Systems we have previously defined, we face the problem of developing an axiomatization that is sound and complete with respect to some behavioural equivalence. To achieve this goal, we propose to transform the systems into a unique normal form which preserves the semantics. As a first step, we introduce axioms which allow the transformation of membrane structures with no dissolving rules into flat membranes. We discuss the problems which arise when dissolving rules are allowed and we suggest possible solutions. We leave as future work the further step that leads to the wanted normal form.

1 Introduction

We have recently defined a compositional operational semantics of P Systems as a labeled transition system (LTS) [2]. The class of P Systems we have considered are the so called *transition P Systems* with dissolving rules and without restrictions on evolution rules. In the definition of the semantics, P Systems are seen as *reactive systems*, namely as systems that can receive stimuli from an environment and can react to these stimuli, possibly by sending some reply back to the environment. In particular, membranes are seen as the entities that can receive stimuli, in terms of objects, from an environment. The environment of a membrane can be another membrane containing it and having some rules which send objects into it. As an environment of a membrane we consider also other membranes possibly contained in it and having some rules which send objects out. The objects received by a membrane from the environment could enable the application of some rules of the membrane that eventually could send some objects back to the environment, namely to outer and inner membranes.

The LTS we have defined allows us to observe the behavior of membranes in terms of objects sent to and received from inner and external membranes. A state of the LTS is a configuration of the considered P System, and a transition

from a state to another describes an execution step of the P System, in which rules are applied according to maximal parallelism in all the membranes of the system. Transitions are labeled with the multiset of objects received from the environment, the multiset of objects sent to outer membranes, and the multiset of objects sent to inner membranes in the described execution step. Other information carried by labels is needed to build the LTS in a compositional way. This means that the semantics of a complex system can be inferred from the semantics of its components.

In [2] we have proved that some well–known behavioural equivalences such as trace equivalence and bisimulation defined on our LTS are congruences. This means that if we can prove that a membrane system that is a component of some bigger system is behaviourally equivalent to another membrane system, then the former can be replaced with the latter in the bigger system without changing the global behaviour. In other words, there exists no environment in which the bigger system with the original component reacts differently to stimuli with respect to the same system with the replaced component.

Behavioural equivalences are powerful analysis tools as they allow us to compare the behaviours of two systems and to verify properties of a system by assessing the equivalence between such a system and another one known to satisfy those properties. However, proving behavioural equivalence is not easy because the semantics of a system often consists of infinite states and infinite transitions. For this reason, it is usually important to find an axiomatization of some behavioural equivalence, namely a sound and complete characterization of the equivalence in terms of axioms on the syntax of systems. In this way the equivalence between two systems could be proved by showing that there exists a sequence of applications of such axioms that transform one system into the other. This allows the proof of the equivalence to be performed without considering the (possibly complex) semantics of the compared systems, and this usually favors the development of tools for the comparison of systems.

We would like to define a sound and complete axiomatization of the semantics we have given in [2]. It is not easy to prove soundness and completeness of an axiomatization, namely that axioms relate behaviorally equivalent systems and that all behaviourally equivalent systems are in the relation characterized by the axioms. In particular, completeness proof is usually difficult. What can help to prove this result is a notion of normal form to which all the considered systems can be reduced. This could allow the set of axioms to be split into two subsets: one consisting of the axioms that can be used to bring the systems into their normal form and the other consisting of axioms that relate systems in normal form. This, in turn, could allow the proof of completeness to be simplified by considering only the axioms in the second set.

In this paper we perform the first steps towards the definition of a normal form for P Systems preserving behavioural equivalences. In particular, we face the problem of determining the membrane structure of the normal form of a system. We start by considering P Systems without dissolving rules, and we show, by

giving a few axioms, that any P System in this class can be transformed into an equivalent P System consisting of one only membrane (a *flat* system). In order to obtain this result we slightly enrich the membranes of a P System, namely we associate with each membrane an *interface*, that is a set of objects that are allowed to be received by the membrane from the environment.

We also discuss the problem of considering P Systems containing dissolving rules. We show that in this case it is no longer possible to find an equivalent flat form, but we discuss how an alternative "standard" form can be reached.

Related work

Operational semantics for P Systems have been proposed in [1, 5, 6, 8]. All these semantics are not compositional and have no notion of observable behavior. In fact, they have not been defined with the aim of developing behavioural equivalences. In particular, [1] aims at simplifying the development of an interpreter of P Systems proved to be correct, [5] aims at proving the decidability of the divergence problem for the considered variant of P Systems, [6] aims at describing the causal dependencies occurring between applications of rules of a P System, and in [8] a formal framework is proposed to describe a large number of variants of P Systems.

The flattening result we obtain by considering P Systems without dissolving rules is similar to the result given in [3], where a notion of *computational encoding* is introduced and used to show that n–PBR Systems (PBR Systems with $n > 0$ membranes) can be simulated by 0–PBR Systems (PBR Systems with no membranes). We refer the reader to [4] for an introduction to PBR Systems. The difference between the result given in [3] and ours is that the axioms we give to transform a P System into its flat form is proved to preserve our compositional semantics, hence it is sound with respect to any behavioural equivalence. The flat system we obtain can replace the original one in any bigger system without changing the global behaviour.

Another normal form of P Systems is introduced in [9], where it is shown that any P System of grade k (namely, in which the depth of the membrane nesting tree is k) consisting of a composition of n membranes can be reduced to an equivalent P System of grade 2 with the same number of membranes. In this case the P System in normal form is equivalent to the original one in the sense that it can generate the same language, where a word of the language is the concatenation of the objects sent outside the skin membrane during the execution of the system. This means that the original system and the one in normal form can be considered as equivalent even if one of the two performs additional steps in which no objects are sent out of the skin. The notion of equivalence we consider here, instead, is stronger. In fact, in order to ensure that a system in normal form can always replace its original system in any context, we need to require that the two systems are step–by–step equivalent.

2 The P Algebra: Syntax and Semantics

In this section we recall the *P Algebra*, the algebraic notation of P Systems we have introduced in [2]. Constants of the P Algebra correspond to single objects or single evolution rules, and they can be composed into membrane systems by using operations of union, containment in a membrane, juxtaposition of membranes, and so on. Terms of the P Algebra are the states of the LTS.

We assume that objects belong to an alphabet V, and we assume the usual string notation to represent multisets of objects. For instance, to represent $\{a, a, b, b, c\}$ we may write either $aabbc$, or a^2b^2c, or $(ab)^2c$. We denote multiset (and set) union as string concatenation, hence we write u_1u_2 for $u_1 \cup u_2$. For the sake of readability, we shall write $u \to v_h v_o\{v_{l_i}\}$ for the generic non–dissolving evolution rule $u \to (v_h, here)(v_o, out)(v_1, in_{l_1}) \dots (v_n, in_{l_n})$, and $u \to v_h v_o\{v_{l_i}\}\delta$ for the similar generic dissolving evolution rule.

The abstract syntax of the P Algebra is defined as follows.

Definition 1 (P Algebra). *The abstract syntax of* membrane contents c, membranes m, *and* membrane systems ms *is given by the following grammar, where* l *ranges over* $\mathbb{N}$ *and* a *over the set of object names* V*:*

$$\begin{aligned} c &::= (\varnothing, \varnothing) \mid (u \to v_h v_o\{v_{l_i}\}, \varnothing) \mid (u \to v_h v_o\{v_{l_i}\}\delta, \varnothing) \mid (\varnothing, a) \mid c \cup c \\ m &::= [_l\, c\,]_l \\ ms &::= m \mid ms \mid ms \mid \mu(m, ms) \mid \mathsf{v} \end{aligned}$$

A membrane content c represents a pair $(\mathcal{R}, u)$, where $\mathcal{R}$ is a set of evolution rules and u is a multiset of objects. A membrane content is obtained through the union operation $\_ \cup \_$ from constants representing single evolution rules and constants representing single objects, and can be plugged into a membrane with label l by means of the operation $[_l\, \_\,]_l$ of membranes m. As a consequence, given a membrane content c representing the pair $(\mathcal{R}, u)$ and $l \in \mathbb{N}$, $[_l\, c\,]_l$ represents the membrane having l as label, $\mathcal{R}$ as evolution rules and u as objects. For the sake of simplicity, we shall usually write $(\mathcal{R}_1\mathcal{R}_2, u_1u_2)$ for $(\mathcal{R}_1\, u_1) \cup (\mathcal{R}_2, u_2)$, $[_l\, u\,]_l$ for $[_l\, (\varnothing, u)\,]_l$ and $[_l\, u_1 \to v_{h1}v_{o1}\{v_{l_i1}\}, \dots, u_1 \to v_{hn}v_{on}\{v_{l_in}\}, u\,]_l$ for $[_l\, (\mathcal{R}, u)\,]_l$ if $\mathcal{R} = \{u_1 \to v_{h1}v_{o1}\{v_{l_i1}\}, \dots, u_1 \to v_{hn}v_{on}\{v_{l_in}\}\}$.

Membrane systems ms have the following meaning: $ms_1 \mid ms_2$ represents the juxtaposition of ms_1 and ms_2, $\mu(m, ms)$ represents the hierarchical composition of m and ms, namely the containment of ms in m, and v represents the *dissolved membrane*. Juxtaposition is used to group sibling membranes, namely membranes all having the same parent in a membrane structure. This operation allows hierarchical composition μ to be defined as a binary operator on a single membrane (the parent) and a juxtaposition of membrane (all the children) rather than on $n + 1$ membranes, for any possible number of children n. Finally, the dissolved membrane v will be used in the definition of the LTS to denote the state of a membrane after the application of one of its dissolving rules.

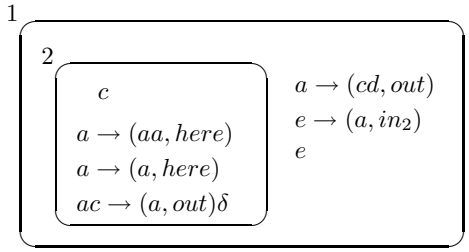


Fig. 1. An example of P System that may send out of the skin membrane a multiset of objects $c^n d^n$ for any $n \in \mathbb{N}$.

As an example, the P System shown in Figure 1 corresponds to the following membrane system:

$$\mu(\, [_1\, a \to (cd, out)\,,\, e \to (a, in_2)\,,\, e\,]_1\,, \\ [_2\, a \to (aa, here)\,,\, a \to (a, here)\,,\, ac \to (a, out)\delta\,,\, c\,]_2\,)\,.$$

Moreover, a P System similar to the one shown in Figure 1, but in which membrane 1 contains also a membrane with label 3 containing, in turn, an object a and no rules, corresponds to the following membrane system:

$$\mu(\, [_1\, a \to (cd, out)\,,\, e \to (a, in_2)\,,\, e\,]_1\,, \\ [_2\, a \to (aa, here)\,,\, a \to (a, here)\,,\, ac \to (a, out)\delta\,,\, c\,]_2 \;\mid\; [_3\, a\,]_3\,)\,.$$

The semantics of the P Algebra is given in terms of an LTS, namely a triple $(\mathcal{S}, \mathcal{L}, \{\xrightarrow{\ell} \mid \ell \in \mathcal{L}\})$, where $\mathcal{S}$ is a set of *states*, $\mathcal{L}$ is a set of *labels*, and $\xrightarrow{\ell} \subseteq \mathcal{S} \times \mathcal{S}$ is a *transition relation* for each $\ell \in \mathcal{L}$. As usual, we write $s \xrightarrow{\ell} s'$ for $(s, s') \in \xrightarrow{\ell}$. LTS labels can be of the following forms:

- $(u, U, v, v', M, I, O^{\uparrow}, O^{\downarrow})$, describing a computation step performed by a membrane content c, where:
  - u is the multiset of objects consumed by the application of evolution rules in c, as it results from the composition, by means of $\_ \cup \_$, of the constants representing these evolution rules.
  - U is the set of multisets of objects corresponding to the left hand sides of the evolution rules in c.
  - v is the multiset of objects in c offered for the application of the evolution rules, as it results from the composition, by means of $\_ \cup \_$, of the constants representing these objects. When operation $[_l\ \_\ ]_l$ is applied to c, it is required that v and u coincide.
  - v' is the multiset of objects in c that are not used to apply any evolution rule and, therefore, are not consumed, as it results from the composition, by means of $\_ \cup \_$, of the constants representing these objects. When operation

$[_{l}\_]_{l}$ is applied to c, it is required that no multiset in U is contained in v', thus implying that no evolution rule in c can be further applied by exploiting the available objects. This constraint is mandatory to ensure maximal parallelism.
  - M contains a membrane label l if some evolution rule in c is not applied since its firing would imply sending objects to some child membrane labeled l, but no child membrane labeled l exists. When the operation μ is applied to $([_{l'}c]_{l'}, ms)$, for any membrane system ms and membrane label l', it is required that l is not a membrane in ms.
  - I is the multiset of objects received as input from the parent membrane and from the child membranes.
  - $O^{\uparrow}$ is the multiset of objects sent as an output to the parent membrane.
  - $O^{\downarrow}$ is a set of pairs (l_i, v_{l_i}) describing the multiset of objects sent as an output to each child membrane l_i.
- $(M, \mathcal{I}, O^{\uparrow}, O^{\downarrow})$, describing a computation step performed by a membrane system ms, where: $\mathcal{I}$ is a set of pairs (l_i, v_{l_i}) describing the multiset of objects received as an input by each membrane l_i in ms, and M, $O^{\uparrow}$ and $O^{\downarrow}$ are as in the previous case.

Components I, $O^{\downarrow}$, $O^{\uparrow}$ in labels of the first form, and components $\mathcal{I}$, $O^{\downarrow}$, $O^{\uparrow}$ in labels of the second form, describe the input/output behavior of P Algebra terms, namely what is usually considered to be the observable behavior. Labels of the first form are more complex since u, U, v, v' are needed to infer the behavior of membrane contents compositionally. For the same reason M is used in both forms of labels.

Now, LTS transitions are defined through SOS transition rules of the form $\frac{premises}{conclusion}$, where the premises are a set of transitions, and the conclusion is a transition. Intuitively, SOS transition rules permit us to infer moves of P Algebra terms from moves of their subterms. Rules of the semantics are given in Appendix A.

3 Flattening Systems without Dissolving Rules

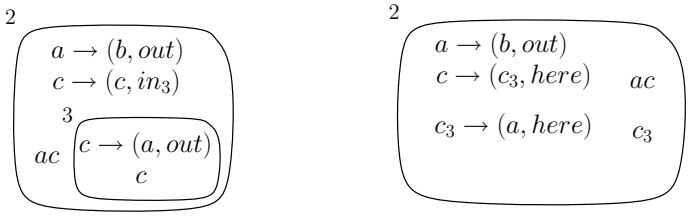


Fig. 2. An example of flattening of a P System without dissolving rules.

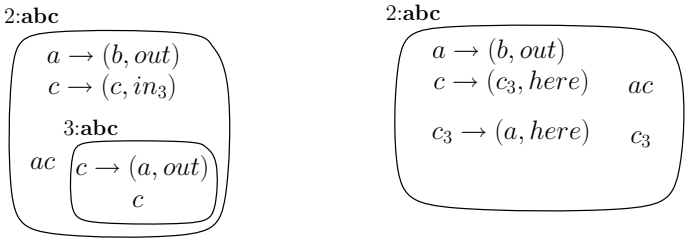


Fig. 3. An example of flattening of a P System without dissolving rules in which membranes are enriched with interfaces.

As shown in [3], any transition P System with a fixed membrane structure (i.e. without dissolving rules) can be reduced to a flat form in which the membrane structure consists only of one membrane. If we assume that membrane labels of a P System are unique, this result can be obtained by moving objects and rules of inner membranes into the external membrane, after suitable renaming. An example of application of this technique is shown in Figure 2. However, the behaviour of the flat membrane is the same as the behaviour of the original membrane structure only under the assumption that the membrane cannot receive any object from the external environment. In fact, if the external environment could send to the flat membrane an object that is the renaming of some object originally in an inner membrane, this could enable the application of some rules among those that have been added to the external membrane by the flattening technique. In the example of the figure, if the environment could send an object c_3 inside the membrane on the right, this would enable the application of rule $c_3 \to (a, here)$ which would result, after one more step, in the output of a b that would not be sent out by the original system.

To solve this problem we consider a slightly extended variant of P Systems in which each membrane is enriched with an interface, namely a set of objects representing the only objects that can be received from the environment. This means that if in the environment of a membrane there is a rule willing to send into it some objects that are not in the corresponding interface, then such a rule will never be applicable. Note that this extension is rather conservative, namely it is always possible to find a set of objects large enough to ensure that the behaviour of a P System extended with interfaces is the same as the intended behaviour of the original P System. As an example, in Figure 3 we extend the P Systems of Figure 2 with interfaces (placed together with membrane labels), and we obtain that in this case the behaviour of the two systems is really the same, as now the environment cannot send c_3 into the external membrane.

Now we formally define the syntax of the P Algebra extended with interfaces on membranes. The main difference with respect to the original syntax, given in Definition 1, is that the operation $[_l \, \_ \,]_l$ of membranes m is extended with a set of objects i. Moreover, since we aim at introducing a notion of flat membrane,

namely a membrane which cannot have inner membranes, we extend the syntax of the P Algebra also with a new operation $[\![_l \_]\!]^i_l$ denoting a membrane that cannot be used as the first operand of a $\mu(\_,\_)$ operation.

Definition 2 (P Algebra with Interfaces). *The abstract syntax of* membrane contents c, membranes m, *and* membrane systems ms *is given by the following grammar, where* l *ranges over* $\mathbb{N}$, *a over* V *and* $i \subseteq V$*:*

$$\begin{aligned} c &::= (\varnothing,\varnothing) \mid (u \to v_h v_o\{v_{l_i}\},\varnothing) \mid (u \to v_h v_o\{v_{l_i}\}\delta,\varnothing) \mid (\varnothing,a) \mid c \cup c \\ m &::= [_l\, c\,]^i_l \\ ms &::= m \mid ms \mid ms \mid \mu(m,ms) \mid [\![_l\, c\,]\!]^i_l \mid \mathsf{v} \end{aligned}$$

We also give the formal definition of the semantics of the P Algebra extended with interfaces on membranes. The main difference with respect to the original semantics is that the objects that can be received as an input by a membrane must belong to the interface of the membrane itself. Moreover, the new semantics has also to describe the behaviour of the new operation $[\![_l \_]\!]^i_l$. Formally, the SOS rules of the semantics of the P Algebra with Interfaces can be defined by starting from those given in Appendix A. In order to describe the behaviour of interfaces we replace rules $(m1)$ and $(m2)$ with the following four rules:

$$\frac{x \xrightarrow[u,U,u,v']{M,\varnothing,O^\uparrow,O^\downarrow} y \quad \delta \notin O^\uparrow}{[_l\, x\,]^i_l \xrightarrow{M,\varnothing,O^\uparrow,O^\downarrow} [_l\, y\,]^i_l} \qquad (m1')$$

$$\frac{x \xrightarrow[u,U,u,v']{M,I,O^\uparrow,O^\downarrow} y \quad \delta \notin O^\uparrow \quad \mathsf{Set}(I) \subseteq i \quad I \neq \varnothing}{[_l\, x\,]^i_l \xrightarrow{M,(l,I),O^\uparrow,O^\downarrow} [_l\, y\,]^i_l} \qquad (m1'')$$

$$\frac{x \xrightarrow[u,U,u,v']{M,\varnothing,O^\uparrow,O^\downarrow} y \quad \delta \in O^\uparrow}{[_l\, x\,]^i_l \xrightarrow{M,\varnothing,O^\uparrow,O^\downarrow} \mathsf{v}} \qquad (m2')$$

$$\frac{x \xrightarrow[u,U,u,v']{M,I,O^\uparrow,O^\downarrow} y \quad \delta \in O^\uparrow \quad \mathsf{Set}(I) \subseteq i \quad I \neq \varnothing}{[_l\, x\,]^i_l \xrightarrow{M,(l,I),O^\uparrow,O^\downarrow} \mathsf{v}} \qquad (m2'')$$

where $\mathsf{Set}(I)$ is the *underlying set* of multiset I, namely the set of all the objects occurring in I. The rules $(m1')$ and $(m1'')$ replace the old rule $(m1)$. Here we distinguish the case $I = \varnothing$ and the case $I \neq \varnothing$, since we prefer to have $\varnothing$ instead of $(l,\varnothing)$ in the label component showing inputs received from the environment. Finally, in order to describe the behaviour of flat membranes we add the following four rules:

$$\frac{x \xrightarrow[u,U,u,v']{M,\varnothing,O^{\uparrow},\varnothing} y \quad \delta \notin O^{\uparrow}}{[\![_l\, x\,]\!]^i_l \xrightarrow{M,\varnothing,O^{\uparrow},\varnothing} [\![_l\, y\,]\!]^i_l} \qquad (fm1')$$

$$\frac{x \xrightarrow[u,U,u,v']{M,I,O^{\uparrow},\varnothing} y \quad \delta \notin O^{\uparrow} \quad \mathsf{Set}(I) \subseteq i \quad I \neq \varnothing}{[\![_l\, x\,]\!]^i_l \xrightarrow{M,(l,I),O^{\uparrow},\varnothing} [\![_l\, y\,]\!]^i_l} \qquad (fm1'')$$

$$\frac{x \xrightarrow[u,U,u,v']{M,\varnothing,O^{\uparrow},\varnothing} y \quad \delta \in O^{\uparrow}}{[\![_l\, x\,]\!]^i_l \xrightarrow{M,\varnothing,O^{\uparrow},\varnothing} \mathsf{v}} \qquad (fm2')$$

$$\frac{x \xrightarrow[u,U,u,v']{M,I,O^{\uparrow},\varnothing} y \quad \delta \in O^{\uparrow} \quad \mathsf{Set}(I) \subseteq i \quad I \neq \varnothing}{[\![_l\, x\,]\!]^i_l \xrightarrow{M,(l,I),O^{\uparrow},\varnothing} \mathsf{v}} \qquad (fm2'')$$

Notice that rules for $[\![_l\, x\,]\!]^i_l$ require that the multiset of objects sent to inner membranes (fourth component of the label) is empty.

The new semantics rules, similarly to all the other rules of the semantics, respect the constraints of the well–known *de Simone* format [7] which ensures that all the behavioural equivalences considered in [2] are congruences.

Now, the flattening technique for systems without dissolving rules can be expressed be means of axioms. We first give some basic axioms on the commutativity and associativity of the operations of the P Algebra and on simple properties of membranes with empty interfaces and of flat membranes.

Definition 3 (Basic axioms). *The* basic axioms *are the following:*

$$c_1 \cup c_2 = c_2 \cup c_1 \qquad (\cup_1)$$
$$c_1 \cup (c_2 \cup c_3) = (c_1 \cup c_2) \cup c_3 \qquad (\cup_2)$$

$$ms_1 \mid ms_2 = ms_2 \mid ms_1 \qquad (|_1)$$
$$ms_1 \mid (ms_2 \mid ms_3) = ms_1 \mid (ms_2 \mid ms_3) \qquad (|_2)$$

$$[_{l_1}\, c\,]^{\varnothing}_{l_1} = [_{l_2}\, c\,]^{\varnothing}_{l_2} \qquad (if1)$$
$$ms = ms \mid [_l\, \mathcal{R}, \varnothing\,]^{\varnothing}_l \qquad (if2)$$

$$[\![_{l_1}\, c\,]\!]^{\varnothing}_{l_1} = [\![_{l_2}\, c\,]\!]^{\varnothing}_{l_2} \qquad (if3)$$
$$ms = ms \mid [\![_l\, \mathcal{R}, \varnothing\,]\!]^{\varnothing}_l \qquad (if4)$$

$$[\![_{l_1}\, c\,]\!]^i_{l_1} = \mu\left([_{l_1}\, c\,]^i_{l_1}\,,\, [_{l_2}\, \mathcal{R}, \varnothing\,]^{\varnothing}_{l_2}\right) \qquad (fm1)$$
$$[\![_{l_1}\, c\,]\!]^i_{l_1} = \mu\left([_{l_1}\, c\,]^i_{l_1}\,,\, [\![_{l_2}\, \mathcal{R}, \varnothing\,]\!]^{\varnothing}_{l_2}\right) \qquad (fm2)$$

The first four axioms state commutativity and associativity of union of membrane contents and juxtaposition of membrane systems. Axiom $(if1)$ states that if a membrane has empty interface then its name l_1 can be changed into l_2. The reason is that l_1 cannot receive any object from any outer membrane, and any evolution rule sending objects to l_1 is never applicable. Axioms $(if2)$ and $(fm1)$ state that a membrane with no object and with empty interface can be juxtaposed with any membrane system or inserted inside another membrane, since its rules are never applicable. Axioms $(if3), (if4)$ and $(fm2)$ are the same as $(if1), (if2)$ and $(fm1)$, respectively, but dealing with flat membranes.

The flattening technique we are going to define is based on renaming of objects. In the example of Figure 3, the object c contained in membrane 3 is renamed into c_3 when it is moved to membrane 2 in order to distinguish it from the other object c that occurs in membrane 2. Consequently, rules of membranes 2 and 3 have to be modified before merging them in membrane 2 resulting from flattening. For this reason, we define two functions FlatIn and $\mathsf{FlatOut}$. The former gives the result of the renaming of the rules of the membrane that is removed by the flattening. The latter gives the result of the renaming of the rules of the membrane which contains the one that is removed. In order to avoid ambiguities, in the definitions of FlatIn and $\mathsf{FlatOut}$ we shall use the notation $u \to (v_h, here)(v_o, out)(v_{l_1}, in_{l_1}) \ldots (v_{l_n}, in_{l_n})$ for evolution rules rather than the more compact notation $u \to v_h v_o\{v_{l_i}\}$.

We assume that the alphabet V is partitioned as follows: $V = \overline{V} \cup (\bigcup_{L \in \mathbb{N}^+} V_L)$, where $\overline{V}$ is the set of all objects without subscripts and V_L is the set obtained by adding $L \in \mathbb{N}^+$ as a subscript to each object of $\overline{V}$. In other words, if $\overline{V} = a, b, c, \ldots$, then $V_1 = a_1, b_1, c_1, \ldots$, $V_{1\cdot 2} = a_{1\cdot 2}, b_{1\cdot 2}, c_{1\cdot 2}, \ldots$ and so on. Moreover, let $\mathsf{Rid}(u, l)$ denote the multiset obtained by replacing each occurrence of each object a_L in u with an occurrence of object $a_{L\cdot l}$. For example, $\mathsf{Rid}(aabccc, 3) = a_3 a_3 b_3 c_3 c_3 c_3 = a_3^2 b_3 c_3^3$, and $\mathsf{Rid}(aa_1 bb_2, 3) = a_3 a_{1\cdot 3} b_3 b_{2\cdot 3}$. The functions FlatIn and $\mathsf{FlatOut}$ are defined as follows:

$$\mathsf{FlatIn}\,(u \to (v_h, here)(v_o, out)(v_{l_1}, in_{l_1}) \ldots (v_{l_n}, in_{l_n})\,,\, l) = \\ \mathsf{Rid}(u, l) \to (\mathsf{Rid}(v_h, l) v_o, here)(\varnothing, out)(v_{l_1}, in_{l_1}) \ldots (v_{l_n}, in_{l_n})$$

$$\mathsf{FlatOut}\,(u \to (v_h, here)(v_o, out)(v_{l_1}, in_{l_1}) \ldots (v_{l_i}, in_{l_i}) \ldots (v_{l_n}, in_{l_n})\,,\, l_i) = \\ u \to (v_h \mathsf{Rid}(v_{l_i}, l_i), here)(v_o, out)(v_{l_1}, in_{l_1}) \ldots (\varnothing, in_{l_i}) \ldots (v_{l_n}, in_{l_n})$$

Both FlatIn and $\mathsf{FlatOut}$ take a rule and a membrane label as arguments, and give a new rule as result. In both cases the membrane label represents the label of the membrane that is removed by the flattening. In the first case such a label (denoted l) should not occur in the evolution rule, as the rule is assumed to be one of those of the inner membrane involved in the flattening. In the second case the label certainly occurs in the evolution rule (in fact it is denoted l_i) as the rule is assumed to be one of those of the outer membrane involved in the flattening. With abuse of notation we shall write $\mathsf{FlatIn}(\mathcal{R}, l)$ for $\{\mathsf{FlatIn}(r, l) \,|\, r \in \mathcal{R}\}$, and $\mathsf{FlatOut}(\mathcal{R}, l)$ for $\{\mathsf{FlatOut}(r, l) \,|\, r \in \mathcal{R}\}$.

Now, the flattening technique is expressed by means of the following axioms.

Definition 4 (Flattening axioms). *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be sets of evolution rules containing no dissolving rule, and let $\mathcal{R}_1$ and u_1 contain no objects in $V_{L\cdot l_2}$. The* flattening axioms *are the following:*

$$\frac{ms \neq \mathsf{v} \quad V_{L\cdot l_2} \cap i_1 = \varnothing \quad \mathsf{FlatOut}(\mathcal{R}_1, l_2) = \mathcal{R}_1' \quad \mathsf{FlatIn}(\mathcal{R}_2, l_2) = \mathcal{R}_2'}{\mu\left([_{l_1}\, \mathcal{R}_1, u_1\,]_{l_1}^{i_1}\,,\, [_{l_2}\, \mathcal{R}_2, u_2\,]_{l_2}^{i_2} \mid ms\right) = \mu\left([_{l_1}\, \mathcal{R}_1'\mathcal{R}_2', u_1\mathsf{Rid}(u_2, l_2)\,]_{l_1}^{i_1}\,,\, ms\right)} \qquad (f1)$$

$$\frac{ms \neq \mathsf{v} \quad V_{L\cdot l_2} \cap i_1 = \varnothing \quad \mathsf{FlatOut}(\mathcal{R}_1, l_2) = \mathcal{R}_1' \quad \mathsf{FlatIn}(\mathcal{R}_2, l_2) = \mathcal{R}_2'}{\mu\left([_{l_1}\, \mathcal{R}_1, u_1\,]_{l_1}^{i_1}\,,\, [\![_{l_2}\, \mathcal{R}_2, u_2\,]\!]_{l_2}^{i_2} \mid ms\right) = \mu\left([_{l_1}\, \mathcal{R}_1'\mathcal{R}_2', u_1\mathsf{Rid}(u_2, l_2)\,]_{l_1}^{i_1}\,,\, ms\right)} \qquad (f2)$$

$$\frac{V_{L\cdot l_2} \cap i_1 = \varnothing \quad \mathsf{FlatOut}(\mathcal{R}_1, l_2) = \mathcal{R}_1' \quad \mathsf{FlatIn}(\mathcal{R}_2, l_2) = \mathcal{R}_2'}{\mu\left([_{l_1}\, \mathcal{R}_1, u_1\,]_{l_1}^{i_1}\,,\, [_{l_2}\, \mathcal{R}_2, u_2\,]_{l_2}^{i_2}\right) = [\![_{l_1}\, \mathcal{R}_1'\mathcal{R}_2'\,,\, u_1\mathsf{Rid}(u_2, l_2)\,]\!]_{l_1}^{i_1}} \qquad (f3)$$

$$\frac{V_{L\cdot l_2} \cap i_1 = \varnothing \quad \mathsf{FlatOut}(\mathcal{R}_1, l_2) = \mathcal{R}_1' \quad \mathsf{FlatIn}(\mathcal{R}_2, l_2) = \mathcal{R}_2'}{\mu\left([_{l_1}\, \mathcal{R}_1, u_1\,]_{l_1}^{i_1}\,,\, [\![_{l_2}\, \mathcal{R}_2, u_2\,]\!]_{l_2}^{i_2}\right) = [\![_{l_1}\, \mathcal{R}_1'\mathcal{R}_2'\,,\, u_1\mathsf{Rid}(u_2, l_2)\,]\!]_{l_1}^{i_1}} \qquad (f4)$$

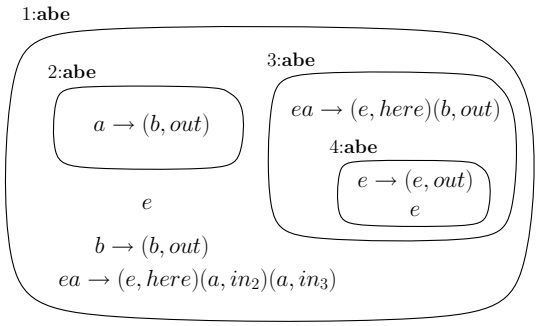


Fig. 4. An example of P System.

As an example, let us consider the P System in Figure 4 which corresponds to the P Algebra term $t = \mu(m_1\,,\, m_2 \mid \mu(m_3, m_4))$ where:

$$m_1 = [_1\, \mathcal{R}_1, e\,]_1^{abe} \text{ with } \mathcal{R}_1 = \{ea \to (e, here)(a, in_2)(a, in_3)\,,\, b \to (b, out)\}$$
$$m_2 = [_2\, \mathcal{R}_2, \varnothing\,]_2^{abe} \text{ with } \mathcal{R}_2 = \{a \to (b, out)\}$$
$$m_3 = [_3\, \mathcal{R}_3, \varnothing\,]_3^{abe} \text{ with } \mathcal{R}_3 = \{ea \to (e, here)(b, out)\}$$
$$m_4 = [_4\, \mathcal{R}_4, e\,]_4^{abe} \text{ with } \mathcal{R}_4 = \{e \to (e, out)\}\ .$$

Now, we have that

$$\begin{aligned}\mu(m_3, m_4) &\overset{(f3)}{=} [\![_3\, \mathsf{FlatOut}(\mathcal{R}_3, 4)\mathsf{FlatIn}(\mathcal{R}_4, 4)\,,\, \varnothing\mathsf{Rid}(e, 4)\,]\!]^{abe}_3 \\ &= [\![_3\, ea \to (e, here)(b, out)\,,\, e_4 \to (e, here)\,,\, e_4\,]\!]^{abe}_3\,.\end{aligned}$$

Let us denote with fm_3 the flat membrane we have obtained. Now, we can go on applying axioms as follows:

$$\begin{aligned}t &= \mu(m_1\,,\, m_2 \mid \mu(m_3, m_4)) = \mu(m_1\,,\, m_2 \mid fm_3) \\ &\overset{(f1)}{=} \mu(\,[\!_1\, \mathsf{FlatOut}(\mathcal{R}_1, 2)\mathsf{FlatIn}(\mathcal{R}_2, 2)\,,\, e\mathsf{Rid}(\varnothing, 2)\,]\!]^{abe}_1\,, \\ &\qquad [\![_3\, ea \to (e, here)(b, out)\,,\, e_4 \to (e, here)\,,\, e_4\,]\!]^{abe}_3\,) \\ &= \mu(\,[\!_1\, ea \to (ea_2, here)(a, in_3)\,,\, b \to (b, out)\,,\, a_2 \to (b, here)\,,\, e\,]\!]^{abe}_1\,, \\ &\qquad [\![_3\, ea \to (e, here)(b, out)\,,\, e_4 \to (e, here)\,,\, e_4\,]\!]^{abe}_3\,) \\ &\overset{(f4)}{=} [\![_1\, ea \to (ea_2a_3, here)\,,\, b \to (b, out)\,,\, a_2 \to (b, here)\,, \\ &\qquad e_3a_3 \to (e_3, here)(b, out)\,,\, e_{43} \to (e_3, here)\,,\, ee_{43}\,]\!]^{abe}_1\,.\end{aligned}$$

Proposition 1 (soundness). *The portions of the LTS that are rooted in terms equated by axioms* $(f1)$*–*$(f4)$ *are isomorphic.*

Proof. We start with the proof for $(f1)$. We prove that the portion of the LTS rooted in $\mu\left([\!_{l_1}\, \mathcal{R}_1, u_1\,]\!]^{i_1}_{l_1}\,,\, [\!_{l_2}\, \mathcal{R}_2, u_2\,]\!]^{i_2}_{l_2} \mid ms\right)$ is isomorphic to a part of the portion of the LTS rooted in $\mu\left([\!_{l_1}\, \mathcal{R}'_1\mathcal{R}'_2, u_1\mathsf{Rid}(u_2, l_2)\,]\!]^{i_1}_{l_1}\,,\, ms\right)$. More precisely, we prove that, given any transition $\mu\left([\!_{l_1}\, \mathcal{R}_1, u_1\,]\!]^{i_1}_{l_1}\,,\, [\!_{l_2}\, \mathcal{R}_2, u_2\,]\!]^{i_2}_{l_2} \mid ms\right) \xrightarrow{l} t$, for any term t, then there is a transition $\mu\left([\!_{l_1}\, \mathcal{R}'_1\mathcal{R}'_2, u_1\mathsf{Rid}(u_2, l_2)\,]\!]^{i_1}_{l_1}\,,\, ms\right) \xrightarrow{l} t'$ such that t and t' are equated by the same axiom $(f1)$.

Take any transition from $\mu\left([\!_{l_1}\, \mathcal{R}_1, u_1\,]\!]^{i_1}_{l_1}\,,\, [\!_{l_2}\, \mathcal{R}_2, u_2\,]\!]^{i_2}_{l_2} \mid ms\right)$. Such a transition must be inferred from a transition of each of its three components. These three transitions have the following shape:

$$[\!_{l_1}\, \mathcal{R}_1, u_1\,]\!]^{i_1}_{l_1} \xrightarrow{M_1, \{(l_1, I_1)\}, O_1^{\uparrow}, O_1^{\downarrow}} [\!_{l_1}\, \mathcal{R}_1, u'_1\,]\!]^{i_1}_{l_1} \tag{1}$$

$$[\!_{l_2}\, \mathcal{R}_2, u_2\,]\!]^{i_2}_{l_2} \xrightarrow{M_2, \{(l_2, I_2)\}, O_2^{\uparrow}, \varnothing} [\!_{l_2}\, \mathcal{R}_2, u'_2\,]\!]^{i_2}_{l_2} \tag{2}$$

$$ms \xrightarrow{M, \mathcal{I}, O^{\uparrow}, \varnothing} ms' \tag{3}$$

Then, transitions (2) and (3) originate transition

$$[\!_{l_2}\, \mathcal{R}_2, u_2\,]\!]^{i_2}_{l_2} \mid ms \xrightarrow{\varnothing, \{(l_2, I_2)\}\mathcal{I}, O_2^{\uparrow}O^{\uparrow}, \varnothing} [\!_{l_2}\, \mathcal{R}_2, u'_2\,]\!]^{i_2}_{l_2} \mid ms' \tag{4}$$

by semantic rule $(jux1)$. The transition from $\mu\left([\!_{l_1}\, \mathcal{R}_1, u_1\,]\!]^{i_1}_{l_1}, [\!_{l_2}\, \mathcal{R}_2, u_2\,]\!]^{i_2}_{l_2} \mid ms\right)$ is inferred through semantic rule $(h1)$ from (1) and (4) and takes the shape:

$$\mu\left(\left[_{l_1}\,\mathcal{R}_1, u_1\,\right]^{i_1}_{l_1}\,,\,\left[_{l_2}\,\mathcal{R}_2, u_2\,\right]^{i_2}_{l_2} \mid ms\right) \xrightarrow{\varnothing,\{(l_1,I_1\setminus(O^{\uparrow}O_2^{\uparrow})\},O_1^{\uparrow},\varnothing} \mu\left(\left[_{l_1}\,\mathcal{R}_1, u_1'\,\right]^{i_1}_{l_1}\,,\,\left[_{l_2}\,\mathcal{R}_2, u_2'\,\right]^{i_2}_{l_2} \mid ms'\right) \quad (5)$$

provided that:

1. $O_1^{\downarrow} \doteq \mathcal{I} \cup \{(l_2, I_2)\}$;
2. $O^{\uparrow}O_2^{\uparrow} \subseteq I_1$.

Note that the first constraint above implies that there exists some $O_1'^{\downarrow}$ such that

$$O_1^{\downarrow} = \{(l_2, I_2)\} \cup O_1'^{\downarrow} \quad (6)$$

Now, (1) can be inferred through $(m1')$ or $(m1'')$. The two cases are similar, let us assume the case $(m1')$. Analogously, let us assume that (2) is inferred through semantic rule $(m2')$. The two originating transitions have the shape:

$$(\mathcal{R}_1, u_1) \xrightarrow[v_1,U_1,v_1,v_1']{M_1,I_1,O_1^{\uparrow},\{(l_2,I_2)\}\cup O_1'^{\downarrow}} (\mathcal{R}_1, u_1') \quad (7)$$

$$(\mathcal{R}_2, u_2) \xrightarrow[v_2,U_2,v_2,v_2']{M_2,I_2,O_2^{\uparrow},\varnothing} (\mathcal{R}_2, u_2') \quad (8)$$

for suitable values $v_1, U_1, v_1', v_2, U_2, v_2'$.
Notice that this implies that $\mathsf{Set}(I_1) \subseteq i_1$. From (7) we infer

$$(\mathcal{R}_1', u_1) \xrightarrow[v_1,U_1,v_1,v_1']{M_1,I_1^{\uparrow},O_1^{\uparrow},O_1'^{\downarrow}} (\mathcal{R}_1', u_1' \setminus (\mathsf{Rid}(I_2, l_2))) \quad (9)$$

By removing input $O_2^{\uparrow}$ we infer

$$(\mathcal{R}_1', u_1) \xrightarrow[v_1,U_1,v_1,v_1']{M_1,I_1\setminus O_2^{\uparrow},O_1^{\uparrow},O_1'^{\downarrow}} (\mathcal{R}_1', u_1' \setminus (O_2^{\uparrow}\mathsf{Rid}(I_2, l_2))) \quad (10)$$

From (8) we infer:

$$(\mathcal{R}_2', \mathsf{Rid}(u_2, l_2)) \xrightarrow[\hat{v_2},\hat{U_2},\hat{v_2},\hat{v_2'}]{M_2,\varnothing,\varnothing,\varnothing} (\mathcal{R}_2', \mathsf{Rid}(u_2' \setminus I_2, l_2)O_2^{\uparrow}) \quad (11)$$

where $\hat{v_2}, \hat{U_2}, \hat{v_2}$ denote $\mathsf{Rid}(v_2, l_2)$, $\mathsf{Rid}(U_2, l_2)$, $\mathsf{Rid}(v_2', l_2)$, respectively.
Through semantic rule $(u1)$, from (10) and (11) we infer

$$(\mathcal{R}_1'\mathcal{R}_2', u_1\mathsf{Rid}(u_2, l_2)) \xrightarrow[v_1\hat{v_2},U_1\oplus\hat{U_2},v_1\hat{v_2},v_1'\hat{v_2'}]{M_1M_2,I_1\setminus O_2^{\uparrow},O_1^{\uparrow},O_1'^{\downarrow}} (\mathcal{R}_1'\mathcal{R}_2', u_1'\mathsf{Rid}(u_2', l_2)) \quad (12)$$

By applying semantic rule $(m1)$, which is applicable since $\mathsf{Set}(I_1) \subseteq i_1$, we infer:

$$[_{l_1}\, \mathcal{R}_1'\mathcal{R}_2', u_1\mathsf{Rid}(u_2,l_2)\,]_{l_1}^{i_1} \xrightarrow{M_1M_2,\{(l_1,I_1\setminus O_2^{\uparrow})\},O_1^{\uparrow},O_1'^{\downarrow}} [_{l_1}\, \mathcal{R}_1'\mathcal{R}_2', u_1'\mathsf{Rid}(u_2',l_2)\,]_{l_1}^{i_1} \quad (13)$$

We already know that $O_1^{\downarrow} \backsimeq \mathcal{I} \cup \{(l_2, I_2)\}$, $O^{\uparrow}O_2^{\uparrow} \subseteq I_1$ and $O_1^{\downarrow} = \{(l_2, I_2)\} \cup O_1'^{\downarrow}$. Therefore, $O_1'^{\downarrow} \backsimeq \mathcal{I}$ and $O^{\uparrow} \subseteq I_1 \setminus O_2^{\uparrow}$. So, we can apply the semantic rule $(h1)$ to infer that (3) and (13) originate

$$\mu\left([_{l_1}\, \mathcal{R}_1'\mathcal{R}_2', u_1\mathsf{Rid}(u_2,l_2)\,]_{l_1}^{i_1},\, ms\right) \xrightarrow{\varnothing,\{(l_1,I_1\setminus O^{\uparrow}O_2^{\uparrow})\},O_1^{\uparrow},\varnothing} \mu\left([_{l_1}\, \mathcal{R}_1'\mathcal{R}_2', u_1'\mathsf{Rid}(u_2',l_2)\,]_{l_1}^{i_1},\, ms'\right). \quad (14)$$

Summarizing, we have proved that if we take any transition from term $\mu\left([_{l_1}\, \mathcal{R}_1, u_1\,]_{l_1}^{i_1},\, [_{l_2}\, \mathcal{R}_2, u_2\,]_{l_2}^{i_2} \mid ms\right)$ we have a corresponding transition from term $\mu\left([_{l_1}\, \mathcal{R}_1'\mathcal{R}_2', u_1\mathsf{Rid}(u_2,l_2)\,]_{l_1}^{i_1},\, ms\right)$, where the two transitions have the same label and have terms related by axiom $(f1)$ in the right side.

The converse is similar, with the use of premise $V_{L\cdot l_2} \cap i_1 = \varnothing$.

The proof of the case of axiom $(f2)$ is the same as the one of axiom $(f1)$, but for minor differences in transition labels. Moreover, the cases of axioms $(f3)$ and $(f4)$ are analogous to the case of $(f1)$ and $(f2)$, respectively, thanks to basic axioms that allow us to rewrite the term $[\![_{l_1}\, \mathcal{R}_1'\mathcal{R}_2',\, u_1\mathsf{Rid}(u_2,l_2)\,]\!]_{l_1}^{i_1}$ as the term $\mu\left([_{l_1}\, \mathcal{R}_1'\mathcal{R}_2',\, u_1\mathsf{Rid}(u_2,l_2)\,]_{l_1}^{i_1},\, [_{l}\, \varnothing\,]_{l}^{\varnothing}\right)$ and $\mu\left([_{l_1}\, \mathcal{R}_1, u_1\,]_{l_1}^{i_1},\, [\![_{l_2}\, \mathcal{R}_2, u_2\,]\!]_{l_2}^{i_2}\right)$ as the term $\mu\left([_{l_1}\, \mathcal{R}_1, u_1\,]_{l_1}^{i_1},\, [\![_{l_2}\, \mathcal{R}_2, u_2\,]\!]_{l_2}^{i_2} \mid [_{l}\, \varnothing\,]_{l}^{\varnothing}\right)$.

Theorem 1. *Any membrane system $\mu(m, ms)$ with multisets of objects and evolution rules built over $\overline{V}$, without dissolving rules and with membranes having unique labels, can be reduced to an equivalent flat membrane.*

Proof. This can be proved by induction on the number of membrane nodes in the membrane nesting tree of $\mu(m, ms)$. If the membrane nesting tree contains two membranes, namely $\mu(m, ms)$ is either $\mu([_{l_1}\, c_1\,]_{l_1}, [_{l_2}\, c_2\,]_{l_2})$ or $\mu([_{l_1}\, c_1\,]_{l_1}, [\![_{l_2}\, c_2\,]\!]_{l_2})$, then the proof follows immediately from axioms $(f3)$ and $(f4)$. If the membrane nesting tree contains more that two membranes, then the proof can be done by resorting to the induction hypothesis after applying one of the axioms $(f1), (f2), (f3)$ and $(f4)$ to one of the leaves of the tree.

Since the size of a term is always finite (and consequently the membrane nesting tree is finite) the flat form is reached after a finite number of steps. The facts that no dissolving rules are present, that multisets of objects and evolution rules are built by using objects from $\overline{V}$ and that membranes are labeled with unique labels, ensure that the assumptions and the premises of the axioms are always satisfied. Finally, Proposition 1 ensures that all the applications of the flattening axioms preserve the behaviour, hence the behaviour of the final flat membrane is equivalent to the one of the original membrane system.

4 Problems that Arise with Dissolving Rules and Possible Solutions

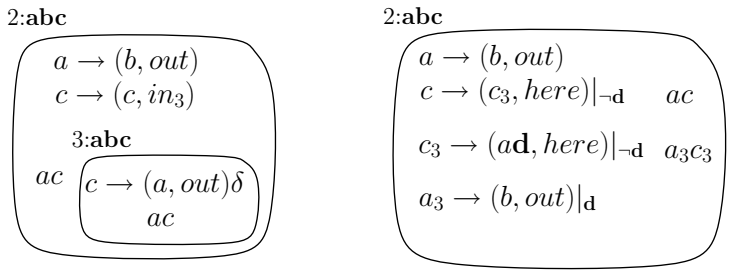


Fig. 5. Example of flattening in which the inner membrane contains a dissolving rule.

In the membrane obtained by our flattening technique, dissolution of a membrane contained in another one has to be simulated. In general, when a dissolving rule is applied in a membrane, we have that (i) the objects of such a membrane become immediately available to the outer membrane, (ii) rules of such a membrane disappear, and (iii) rules of the outer membrane which send objects to the membrane that has been dissolved become no longer applicable.

One possible way of simulating dissolution is by replacing δ with a special object $\mathbf{d}$ in every dissolving rule and using such a special object as a promoter or inhibitor of some rules obtained by the flattening (after extending the syntax and the semantics of the P Algebra to deal with promoters and inhibitors). This would allow (ii) and (iii) to be simulated by using $\mathbf{d}$ as an inhibitor of those rules obtained by the flattening and corresponding to the rules of the dissolved membrane and to the rules sending objects to the dissolved membranes. Moreover, (i) can be simulated by defining the flattening in such a way that the rules of the outer membrane are copied with the objects they consume renamed, in order to allow such rules to be applied to the objects representing the objects of the dissolved membrane. These new rules should have $\mathbf{d}$ has a promoter.

We give a simple example of flattening with dissolution of inner membranes in Figure 5. Here, the rule causing dissolution of membrane 3 is rewritten into a new rule having objects renamed as described in the previous section and producing $\mathbf{d}$. Now, both the rule originally in 2 and sending objects to 3, and the rule originally in 3 require that $\mathbf{d}$ has not yet been produced. Moreover, a new rule promoted by $\mathbf{d}$ has been introduced to simulate that the objects originally in membrane 3 are available in membrane 2 after its dissolution.

The flattening technique explained in the previous section cannot be applied if the outer membrane contains a dissolving rule. As an example, let us consider Fig. 6, where flattening is applied. We can provide a context in which the original membrane system and the flat membrane behave differently (see Fig. 7). The point

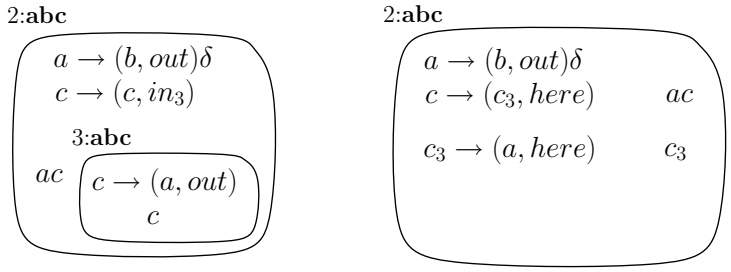


Fig. 6. Example of flattening in which the outer membrane contains a dissolving rule.

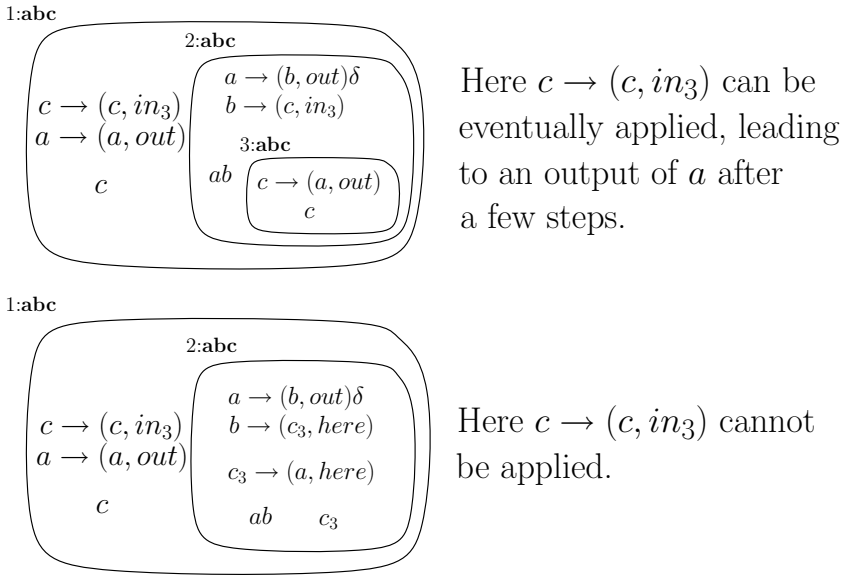


Fig. 7. Example of context in which the two membranes systems in Figure 6 behave differently.

is that the rule of membrane 1 sending object c to membrane 3 can be eventually applied if and only if membrane 3 still exists after the dissolution of membrane 2. In this case the only possible solution is to avoid flattening of membrane structures in which the outermost membrane can be dissolved. As a consequence, a general normal form for P Systems will have two shapes:

- If the external membrane of a membrane structure cannot be dissolved its normal form is a single flat membrane that cannot be dissolved.
- If the external membrane of a membrane structure can be dissolved its normal form is a structure consisting only of membranes that can be dissolved, but for innermost membranes that might be non–dissolvable.

5 Conclusions and Future Work

We have faced the problem of defining a flattening technique for P Systems defined by means of axioms on terms of the algebra of such systems we have introduced in [2], the P Algebra, and preserving the semantics. We have formally defined such a technique in the case of P Systems without dissolving rules. This has required extending the syntax and the semantics of the P Algebra with a notion of interface and with a notion of flat membrane, defining some axioms and proving that these axioms preserve the semantics. We have discussed the problems that arise when dissolving rules are taken into account, and we have proposed some possible solutions to these problems.

Our long term aim is to define a normal form of P System. In order to reach the normal form of a P System, in addition to apply our flattening technique we would also need to transform rules and objects of such a system into some minimal form. We believe that, given two systems in normal form, it will be possible to check their equivalence as follows:

- if they are both flat, they should contain the same rules and objects, up to a suitable renaming;
- if they are both non flat because the external membrane contain a dissolving rule (see Section 4), they should have the same membrane structure of equivalent membranes.

Acknowledgments

This research has been partially supported by MiUR PRIN 2006 Project "Biologically Inspired Systems and Calculi and their Applications (BISCA)". We thank Pierluigi Frisco for interesting discussions.

References

1. O. Andrei, G. Ciobanu, D. Lucanu. A Rewriting Logic Framework for Operational Semantics of Membrane Systems. Theoret. Comp. Sci. 373 (2007) 163–181.
2. R. Barbuti, A. Maggiolo–Schettini, P. Milazzo, S.Tini. Compositional Semantics and Behavioral Equivalences for P Systems. Theoret. Comput. Sci., in press.
3. L. Bianco, V. Manca. Encoding–Decoding Transitional Systems for Classes of P Systems. Workshop on Membrane Computing (WMC 2005), LNCS 3850, pp. 134–143, Springer, 2006.
4. L. Bianco, F. Fontana, G. Franco, V. Manca. P Systems for Biological Dynamics. In: Applications of Membrane Computing, Springer, Berlin, 2006.
5. N. Busi. Using Well–structured Transition Systems to Decide Divergence for Catalytic P Systems. Theoret. Comput. Sci. 372 (2007) 125–135.
6. N. Busi. Causality in Membrane Systems. Workshop on Membrane Computing (WMC 2007), LNCS 4860, pp. 160–171, Springer, 2007.

7. R. de Simone. High Level Synchronization Devices in Meije-SCCS. Theoret. Comput. Sci. 37, pp. 245–267, 1985.
8. R. Freund, S. Verlan. A Formal Framework for Static (Tissue) P Systems. Workshop on Membrane Computing (WMC 2007), LNCS 4860, pp. 271–284, Springer, 2007.
9. I. Petre. A Normal Form for P-Systems. Bulletin of the EATCS 67 (1999) 165–172.

A Rules of the Operational Semantics

In this section we recall the rules of the operational sematics of the P Algebra given in [2].

A.1 Rules for membrane contents

$$\frac{I \in V^* \quad n \in \mathbb{N}}{(u \to v_h v_o\{v_{l_i}\}, \varnothing) \xrightarrow[u^n,\{u\},\varnothing,\varnothing]{\varnothing,I,v_o^n,\{(l_i,v_{l_i}^n)\}} (u \to v_h v_o\{v_{l_i}\}, I v_h^n)} \qquad (mc1_n)$$

$$\frac{I \in V^* \quad n \in \mathbb{N} \quad n > 0}{(u \to v_h v_o\{v_{l_i}\}\delta, \varnothing) \xrightarrow[u^n,\{u\},\varnothing,\varnothing]{\varnothing,I,I v_o^n v_h^n \delta,\{(l_i,v_{l_i}^n)\}} \mathsf{v}} \qquad (mc2_n)$$

$$\frac{I \in V^*}{(u \to v_h v_o\{v_{l_i}\}\delta, \varnothing) \xrightarrow[\varnothing,\{u\},\varnothing,\varnothing]{\varnothing,I,\varnothing,\varnothing} (u \to v_h v_o\{v_{l_i}\}\delta, I)} \qquad (mc3)$$

$$\frac{I \in V^* \quad M \subseteq \mathsf{Labels}(\{v_{l_i}\}) \quad M \neq \varnothing}{(u \to v_h v_o\{v_{l_i}\}, \varnothing) \xrightarrow[\varnothing,\varnothing,\varnothing,\varnothing]{M,I,\varnothing,\varnothing} (u \to v_h v_o\{v_{l_i}\}, I)} \qquad (mc4)$$

$$\frac{I \in V^* \quad M \subseteq \mathsf{Labels}(\{v_{l_i}\}) \quad M \neq \varnothing}{(u \to v_h v_o\{v_{l_i}\}\delta, \varnothing) \xrightarrow[\varnothing,\varnothing,\varnothing,\varnothing]{M,I,\varnothing,\varnothing} (u \to v_h v_o\{v_{l_i}\}\delta, I)} \qquad (mc5)$$

$$\frac{I \in V^*}{(\varnothing, a) \xrightarrow[\varnothing,\varnothing,a,\varnothing]{\varnothing,I,\varnothing,\varnothing} (\varnothing, I)} \qquad (mc6) \qquad \frac{I \in V^*}{(\varnothing, a) \xrightarrow[\varnothing,\varnothing,\varnothing,a]{\varnothing,I,\varnothing,\varnothing} (\varnothing, Ia)} \qquad (mc7)$$

$$\frac{I \in V^*}{(\varnothing, \varnothing) \xrightarrow[\varnothing,\varnothing,\varnothing,\varnothing]{\varnothing,I,\varnothing,\varnothing} (\varnothing, I)} \qquad (mc8)$$

A.2 Rules for union of membrane contents

$$\frac{x_1 \xrightarrow[u_1,U_1,v_1,v_1']{M_1,I_1,O_1^\uparrow,O_1^\downarrow} y_1 \quad x_2 \xrightarrow[u_2,U_2,v_2,v_2']{M_2,I_2,O_2^\uparrow,O_2^\downarrow} y_2 \quad \begin{array}{c} M_1M_2 \cap \mathsf{Labels}(O_1^\downarrow \cup_{\mathbb{N}} O_2^\downarrow) = \varnothing \\ v_1'v_2' \not\vdash U_1 \oplus U_2 \quad \delta \notin O_1^\uparrow O_2^\uparrow \end{array}}{x_1 \cup x_2 \xrightarrow[u_1u_2,U_1\oplus U_2,v_1v_2,v_1'v_2']{M_1M_2,I_1I_2,O_1^\uparrow O_2^\uparrow,O_1^\downarrow \cup_{\mathbb{N}} O_2^\downarrow} y_1 \cup y_2} \quad (u1)$$

$$\frac{x_1 \xrightarrow[u_1,U_1,v_1,v_1']{M_1,I_1,O_1^\uparrow,O_1^\downarrow} y_1 \quad x_2 \xrightarrow[u_2,U_2,v_2,v_2']{M_2,I_2,O_2^\uparrow,O_2^\downarrow} y_2 \quad \begin{array}{c} M_1M_2 \cap \mathsf{Labels}(O_1^\downarrow \cup_{\mathbb{N}} O_2^\downarrow) = \varnothing \\ v_1'v_2' \not\vdash U_1 \oplus U_2 \quad \delta \in O_1^\uparrow \quad \delta \notin O_2^\uparrow \end{array}}{x_1 \cup x_2 \xrightarrow[u_1u_2,U_1\oplus U_2,v_1v_2,v_1'v_2']{M_1M_2,I_1I_2,O_1^\uparrow O_2^\uparrow \mathsf{Objects}(y_2),O_1^\downarrow \cup_{\mathbb{N}} O_2^\downarrow} \mathsf{v}} \quad (u2)$$

$$\frac{x_1 \xrightarrow[u_1,U_1,v_1,v_1']{M_1,I_1,O_1^\uparrow,O_1^\downarrow} y_1 \quad x_2 \xrightarrow[u_2,U_2,v_2,v_2']{M_2,I_2,O_2^\uparrow,O_2^\downarrow} y_2 \quad \begin{array}{c} M_1M_2 \cap \mathsf{Labels}(O_1^\downarrow \cup_{\mathbb{N}} O_2^\downarrow) = \varnothing \\ v_1'v_2' \not\vdash U_1 \oplus U_2 \quad \delta \in O_1^\uparrow \cap O_2^\uparrow \end{array}}{x_1 \cup x_2 \xrightarrow[u_1u_2,U_1\oplus U_2,v_1v_2,v_1'v_2']{M_1M_2,I_1I_2,O_1^\uparrow O_2^\uparrow,O_1^\downarrow \cup_{\mathbb{N}} O_2^\downarrow} \mathsf{v}} \quad (u3)$$

A.3 Rules for single membranes and juxtaposition of membranes

$$\frac{x \xrightarrow[u,U,u,v']{M,I,O^\uparrow,O^\downarrow} y \quad \delta \notin O^\uparrow}{[_l\, x\,]_l \xrightarrow{M,\{(l,I)\},O^\uparrow,O^\downarrow} [_l\, y\,]_l} \quad (m1) \qquad \frac{x \xrightarrow[u,U,u,v']{M,I,O^\uparrow,O^\downarrow} y \quad \delta \in O^\uparrow}{[_l\, x\,]_l \xrightarrow{M,\{(l,I)\},O^\uparrow,O^\downarrow} \mathsf{v}} \quad (m2)$$

$$\frac{x_1 \xrightarrow{M_1,\mathcal{I}_1,O_1^\uparrow,\varnothing} y_1 \quad x_2 \xrightarrow{M_2,\mathcal{I}_2,O_2^\uparrow,\varnothing} y_2 \quad \delta \notin O_1^\uparrow O_2^\uparrow}{x_1|x_2 \xrightarrow{\varnothing,\mathcal{I}_1\mathcal{I}_2,O_1^\uparrow O_2^\uparrow,\varnothing} y_1|y_2} \quad (jux1)$$

$$\frac{x_1 \xrightarrow{M_1,\mathcal{I}_1,O_1^\uparrow,\varnothing} y_1 \quad x_2 \xrightarrow{M_2,\mathcal{I}_2,O_2^\uparrow,\varnothing} y_2 \quad \delta \in O_1^\uparrow, \delta \notin O_2^\uparrow}{x_1|x_2 \xrightarrow{\varnothing,\mathcal{I}_1\mathcal{I}_2,(O_1^\uparrow O_2^\uparrow)-\delta,\varnothing} y_2} \quad (jux2)$$

$$\frac{x_1 \xrightarrow{M_1,\mathcal{I}_1,O_1^\uparrow,\varnothing} y_1 \quad x_2 \xrightarrow{M_2,\mathcal{I}_2,O_2^\uparrow,\varnothing} y_2 \quad \delta \in O_1^\uparrow \cap O_2^\uparrow}{x_1|x_2 \xrightarrow{\varnothing,\mathcal{I}_1\mathcal{I}_2,(O_1^\uparrow O_2^\uparrow),\varnothing} \mathsf{v}} \quad (jux3)$$

A.4 Rules for hierarchy of membranes

$$\frac{x_1 \xrightarrow{M_1,\{(l_1,I_1)\},O_1^{\uparrow},O_1^{\downarrow}} y_1 \quad x_2 \xrightarrow{M_2,\mathcal{I}_2,O_2^{\uparrow},\varnothing} y_2 \qquad \begin{array}{c} O_1^{\downarrow} \backsimeq \mathcal{I}_2 \quad O_2^{\uparrow} \subseteq I_1 \\ M_1 \cap \mathsf{Labels}(\mathcal{I}_2) = \varnothing \quad \delta \notin O_1^{\uparrow} O_2^{\uparrow} \end{array}}{\mu(x_1,x_2) \xrightarrow{\varnothing,(l_1,I_1 \setminus O_2^{\uparrow}),O_1^{\uparrow},\varnothing} \mu(y_1,y_2)} \quad (h1)$$

$$\frac{x_1 \xrightarrow{M_1,\{(l_1,I_1)\},O_1^{\uparrow},O_1^{\downarrow}} y_1 \quad x_2 \xrightarrow{M_2,\mathcal{I}_2,O_2^{\uparrow},\varnothing} y_2 \qquad \begin{array}{c} O_1^{\downarrow} \backsimeq \mathcal{I}_2 \quad O_2^{\uparrow} \subseteq I_1 \quad \delta \in O_1^{\uparrow} \\ M_1 \cap \mathsf{Labels}(\mathcal{I}_2) = \varnothing \quad \delta \notin O_2^{\uparrow} \end{array}}{\mu(x_1,x_2) \xrightarrow{\varnothing,\{(l_1,I_1 \setminus O_2^{\uparrow})\},O_1^{\uparrow}-\delta,\varnothing} y_2} \quad (h2)$$

$$\frac{x_1 \xrightarrow{M_1,\{(l_1,I_1)\},O_1^{\uparrow},O_1^{\downarrow}} y_1 \quad x_2 \xrightarrow{M_2,\mathcal{I}_2,O_2^{\uparrow},\varnothing} y_2 \qquad \begin{array}{c} O_1^{\downarrow} \backsimeq \mathcal{I}_2 \quad O_2^{\uparrow} - \delta \subseteq I_1 \quad \delta \notin O_1^{\uparrow} \\ M_1 \cap \mathsf{Labels}(\mathcal{I}_2) = \varnothing \quad \delta \in O_2^{\uparrow} \end{array}}{\mu(x_1,x_2) \xrightarrow{\varnothing,\{(l_1,I_1 \setminus O_2^{\uparrow})\},O_1^{\uparrow},\varnothing} y_1} \quad (h3)$$

$$\frac{x_1 \xrightarrow{M_1,\{(l_1,I_1)\},O_1^{\uparrow},O_1^{\downarrow}} y_1 \quad x_2 \xrightarrow{M_2,\mathcal{I}_2,O_2^{\uparrow},\varnothing} y_2 \qquad \begin{array}{c} O_1^{\downarrow} \backsimeq \mathcal{I}_2 \quad O_2^{\uparrow} - \delta \subseteq I_1 \\ M_1 \cap \mathsf{Labels}(\mathcal{I}_2) = \varnothing \quad \delta \in O_1^{\uparrow} \cap O_2^{\uparrow} \end{array}}{\mu(x_1,x_2) \xrightarrow{\varnothing,\{(l_1,I_1 \setminus O_2^{\uparrow})\},O_1^{\uparrow},\varnothing} \mathsf{v}} \quad (h4)$$

(Tissue) P Systems Using Non-cooperative Rules Without Halting Conditions

Markus Beyreder, Rudolf Freund

Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9, A-1040 Wien, Austria
E-mails: {markus,rudi}@emcc.at

Summary. We consider (tissue) P systems using non-cooperative rules, but considering computations without halting conditions. As results of a computation we take the contents of a specified output membrane/cell in each derivation step, no matter whether this computation will ever halt or not, eventually taking only results completely consisting of terminal objects only. The computational power of (tissue) P systems using non-cooperative rules turns out to be equivalent to that of (E)0L systems.

1 Introduction

In contrast to the original model of P systems introduced in [5], in this paper we only consider non-cooperative rules. Moreover, as results of a computation we take the contents of a specified output membrane in each derivation step, no matter whether this computation will ever halt or not, eventually taking only results completely consisting of terminal objects. In every derivation step, we apply the traditional maximal parallelism. Other derivation modes could be considered, too, but, for example, applying the sequential derivation mode would not allow us to go beyond context-free languages. As the model defined in this paper we shall take the more general one of tissue P systems (where the communication structure of the system is an arbitrary graph, e.g., see [4], [2]), which as a specific subvariant includes the original model of membrane systems if the communication structure allows for arranging the cells in a hierarchical tree structure.

The motivation to consider this specific variant of tissue P systems came during the Sixth Brainstorming Week in Sevilla 2008 when discussing the ideas presented in [3] with the authors Miguel Gutiérrez-Naranjo and Mario Pérez-Jiménez. They consider the evolution of deterministic (tissue) P systems with simple (i.e., non-cooperative) rules and aim to find a mathematically sound representation of such systems in order to deduce their behaviour and, on the other hand, to find suitable corresponding P systems for a given mathematical system with specific behaviour. Whereas in that paper only deterministic P systems are considered, which allows

for a mathematical representation like for deterministic 0L systems, and as well real values for the coefficients assigned to the symbols are allowed, in this paper we restrict ourselves to the non-negative integer coefficients commonly used in traditional variants of (tissue) P systems.

We shall prove that the computational power of extended tissue P systems using non-cooperative rules is equivalent to that of E0L systems when taking all results appearing in the specified output cell consisting of terminal objects only.

The present paper is organized as follows. Section 2 briefly recalls the notations commonly used in membrane computing and the few notions of formal language theory that will be used in the rest of the paper; in particular, we report the definition of (extended) Lindenmayer systems. Section 3 is dedicated to the definition of tissue P systems with non-cooperative rules working in the maximally parallel derivation mode. The computational power of these classes of (extended) tissue P systems is then investigated in Section 4 in comparison with the power of the corresponding classes of (extended) Lindenmayer systems. Some further remarks and directions for future research are discussed in the last section.

2 Preliminaries

We here recall some basic notions concerning the notations commonly used in membrane computing (we refer to [6] for further details and to [9] for the actual state of the art in the area of P systems) and the few notions of formal language theory we need in the rest of the paper (see, for example, [8] and [1], as well as [7] for the mathematical theory of L systems).

An alphabet is a finite non-empty set of abstract symbols. Given an alphabet V, by V^* we denote the set of all possible strings over V, including the empty string λ. The length of a string $x \in V^*$ is denoted by $|x|$ and, for each $a \in V$, $|x|_a$ denotes the number of occurrences of the symbol a in x. A multiset over V is a mapping $M : V \longrightarrow \mathbb{N}$ such that $M(a)$ defines the multiplicity of a in the multiset M ($\mathbb{N}$ denotes the set of non-negative integers). Such a multiset can be represented by a string $a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \in V^*$ and by all its permutations, with $a_j \in V$, $M(a_j) \geq 0$, $1 \leq j \leq n$. In other words, we can say that each string $x \in V^*$ identifies a finite multiset over V defined by $M_x = \{(a, |x|_a) \mid a \in V\}$. Ordering the symbols in V in a specific way, i.e., $(a_1, \dots, a_n)$ such that $\{a_1, \dots, a_n\} = V$, we get a Parikh vector $\left(|x|_{a_1}, \dots, |x|_{a_n}\right)$ associated with x. The set of all multisets over V is denoted by M_V, the set of all Parikh vectors by $Ps\,(V^*)$. In the following, we shall not distinguish between multisets and the corresponding Parikh vectors. Given two multisets x and y, with $x, y \in V^*$, we say that the multiset x includes the multiset y, or the multiset y is included in the multiset x, and we write $x \sqsupseteq y$, or $y \sqsubseteq x$, if and only if $|x|_a \geq |y|_a$, for every $a \in V$. The union of two multisets x and y is denoted by $x \sqcup y$ and is defined to be the multiset with $|x \sqcup y|_a = |x|_a + |y|_a$, for every $a \in V$. For $m, n \in \mathbb{N}$, by $[m..n]$ we denote the set $\{x \in \mathbb{N} \mid m \leq x \leq n\}$.

An extended Lindenmayer system (an E0L system for short) is a construct $G = (V, T, P, w)$, where V is an alphabet, $T \subseteq V$ is the *terminal* alphabet, $w \in V^*$ is the *axiom*, and P is a finite set of *non-cooperative rules* over V of the form $a \to x$. In a derivation step, each symbol present in the current sentential form is rewritten using one rule arbitrarily chosen from P. The language generated by G, denoted by $L(G)$, consists of all the strings over T which can be generated in this way by starting from w. An E0L system with $T = V$ is called a 0L system. As a technical detail we have to mention that in the theory of Lindenmayer systems usually it is required that for every symbol a from V at least one rule $a \to w$ in P exists. If for every symbol a from V exactly one rule $a \to w$ in P exists, then this Lindenmayer system is called *deterministic*, and we use the notations DE0L and D0L systems. By $E0L$ and $0L$ ($DE0L$ and $D0L$) we denote the families of languages generated by (deterministic) E0L systems and 0L systems, respectively. It is known from [8] that $CF \subset E0L \subset CS$, with CF being the family of context-free languages and CS being the family of context-sensitive languages, and that CF and $0L$ are incomparable, with $\left\{a^{2^n} \mid n \geq 0\right\} \in D0L - CF$.

As the paper deals with P systems where we consider symbol objects, we will also consider E0L systems as devices that generate sets of (vectors of) non-negative integers; to this aim, given an E0L system G, we define the set of non-negative integers generated by G as the length set $N(G) = \{\, |x| \mid x \in L(G) \,\}$ as well as $Ps\,(G)$ to be the set of Parikh vectors corresponding to the strings in $L\,(G)$. In the same way, the length sets and the Parikh sets of the languages generated by context-free and context-sensitive grammars can be defined. The corresponding families of sets of (vectors of) non-negative integers then are denoted by NX and PsX, for $X \in \{E0L, 0L, DE0L, D0L, CF, CS\}$, respectively.

3 Tissue P Systems With Non-cooperative Rules

Now we formally introduce the notion of tissue P systems with non-cooperative rules by giving the following definition.

Definition 1. *An* extended tissue P system with non-cooperative rules *is a construct*

$$\Pi = (n, V, T, R, C_0, i_0)$$

where

1. *n is the number of* cells*;*
2. *V is a finite alphabet of symbols called* objects*;*
3. *$T \subseteq V$ is a finite alphabet of* terminal symbols *(*terminal objects*);*
4. *R is a finite set of multiset rewriting rules of the form*

$$(a, i) \to (b_1, h_1) \ldots (b_k, h_k)$$

for $i \in [1..k]$, $a \in V$ as well as $b_j \in V$ and $h_j \in [1..n]$, $j \in [1..k]$;

5. $C_0 = (w_1, \ldots, w_n)$, *where the* $w_i \in V^*$, $i \in [1..n]$, *are finite multisets of objects for each* $i \in [1..n]$,
6. i_0 *is the* output cell.

A rule $(a, i) \to (b_1, h_1) \ldots (b_k, h_k)$ in R_i indicates that a copy of the symbol a in cell i is erased and instead, for all $j \in [1..k]$, a copy of the symbol b_j is added in cell h_j.

In any configuration of the tissue P system, a copy of the symbol a in cell i is represented by (a, i), i.e., (a, i) is an element of $V \times [1..n]$.

Π is called *deterministic* if in every cell for every symbol from V exactly one rule exists.

From the initial configuration specified by $(w_1, ..., w_n)$, the system evolves by transitions getting from one configuration to the next one by applying a maximal set of rules in every cell, i.e., by working in the *maximally parallel derivation mode*. A *computation* is a sequence of transitions. In contrast to the common use of P systems to generate sets of multisets, as a result of the P system we take the contents of cell i_0, provided it only consists of terminal objects only, at each step of any computation, no matter whether this computation will ever stop or not, i.e., we do not take into account any halting condition. The set of all multisets generated in that way by Π is denoted by $L(\Pi)$. If we are only interested in the number of symbols instead of the Parikh vectors, the corresponding set of numbers generated by Π is denoted by $N(\Pi)$.

The family of sets of multisets generated by tissue P systems with non-cooperative rules with at most n cells in the maximally parallel derivation mode is denoted by $PsEtOP_n(noncoop, maxpar)$. Considering only the length sets instead of the Parikh vectors of the results obtained in the output cell during the computations of the tissue P systems, we obtain the family of sets of non-negative integers generated by tissue P systems with non-cooperative rules with at most n cells in the maximally parallel derivation mode, denoted by $NEtOP_n(noncoop, maxpar)$. The corresponding families generated by non-extended tissue P systems – where all symbols are terminal – are denoted by $XtOP_n(noncoop, maxpar)$, $X \in \{Ps, N\}$. For all families generated by (extended) tissue P systems as defined before, we add the symbol D in front of t if the underlying systems are deterministic. If the number of cells is allowed to be arbitrarily chosen, we replace n by $*$.

3.1 A well-known example

Consider the D0L system with the only rule $a \to aa$, i.e.,

$$G = (\{a\}, \{a\}, \{a \to aa\}, a).$$

As is well known, the language generated by G is $\{a^{2^n} \mid n \geq 0\}$ and therefore $N(G) = \{2^n \mid n \geq 0\}$.

The corresponding deterministic one-cell tissue P system is

$$\Pi = (\{a\}, \{a\}, \{(a, 1) \to (a, 1)\,(a, 1)\}, (a)).$$

Obviously, we get $L(\Pi) = Ps(L(G))$ and $N(G) = N(\Pi)$.

We should like to point out that in contrast to this tissue P system without imposing halting, there exists no tissue P system with only one symbol in one cell

$$\Pi = (\{a\}, \{a\}, R, (w))$$

that with imposing halting is able to generate $\{2^n \mid n \geq 0\}$, because such systems can generate only finite sets (singletons or the empty set):

- if $w = \lambda$, then $N(\Pi) = \{0\}$;
- if R is empty, then $N(\Pi) = \{|w|\}$;
- if $w \neq \lambda$ and R contains the rule $a \to \lambda$, then $N(\Pi) = \{0\}$, because no computation can stop as long as the contents of the cell is not empty;
- if $w \neq \lambda$ and R is not empty, but does not contain the rule $a \to \lambda$, then R must contain a rule of the form $a \to a^n$ for some $n \geq 1$, yet this means that there exists no halting computation, i.e., $N(\Pi)$ is empty.

4 The Computational Power of Tissue P Systems With Non-cooperative Rules

In this section we present some results concerning the generative power of (extended) tissue P systems with non-cooperative rules; as we shall show, there is a strong correspondence between these P systems with non-cooperative rules and E0L systems.

Theorem 1. *For all $n \geq 1$,*

$$\begin{aligned} PsE0L &= PsEtOP_n(noncoop, maxpar) \\ &= PsEtOP_*(noncoop, maxpar). \end{aligned}$$

Proof. We first show that

$$PsE0L \subseteq PsEtOP_1(noncoop, maxpar):$$

Let $G = (V, T, P, w)$ be an E0L system. Then we construct the corresponding extended one-cell tissue P system

$$\Pi = (1, V, T, R, (w), 1)$$

with

$$R = \{(a, 1) \to (b_1, 1) \dots (b_k, 1) \mid a \to b_1 \dots b_k \in P\}.$$

Due to the maximal parallel derivation mode applied in the extended tissue P system Π, the derivations in Π directly correspond to the derivations in G. Hence, $L(\Pi) = Ps(L(G))$.

As for all $n \geq 1$, by definition we have

$$PsEtOP_1 (noncoop, maxpar) \subseteq PsEtOP_n (noncoop, maxpar),$$

it only remains to show that

$$PsEtOP_* (noncoop, maxpar) \subseteq PsE0L :$$

Let

$$\Pi = (n, V, T, R, (w_1, \ldots, w_n), i_0)$$

be an extended tissue P system. Then we first construct the E0L system

$$G = (V \times [1..n], T_0, P, w)$$

with

$$w = \sqcup_{i=1}^{n} h_i (w_i)$$

($\sqcup$ represents the union of multisets) and

$$T_0 = h_{i_0} (T) \cup \cup_{j \in [1..n], j \neq i_0} h_j (V)$$

where the $h_i : V^* \to \{(a, i) \mid a \in V\}^*$ are morphisms with $h_i (a) = (a, i)$ for $a \in V$ and $i \in [1..n]$, as well as

$$P = R \cup P'$$

where P' contains the rule $(a, i) \to (a, i)$ for $a \in V$ and $i \in [1..n]$ if and only if R contains no rule for (a, i) (which guarantees that in P there exists at least one rule for every $b \in V \times [1..n]$).

We now take the projection $h : T_0^* \to T^*$ with $h ((a, i_0)) = a$ for all $a \in T$ and $h ((a, j)) = \lambda$ for all $a \in V$ and $j \in [1..n]$, $j \neq i_0$. Due to the direct correspondence of derivations in Π and G, respectively, we immediately obtain $Ps (h (L (G))) = L (\Pi)$.

As $E0L$ is closed under morphisms (e.g., see [8], vol. 1, p. 266f.) and therefore $L (\Pi) = Ps (L (G'))$ for some E0L system G', we finally obtain $L (\Pi) \in PsE0L$. □

As an immediate consequence of Theorem 1, we obtain the following results:

Corollary 1. *For all $n \geq 1$,*

$$\begin{aligned} NE0L &= NEtOP_n (noncoop, maxpar) \\ &= NEtOP_* (noncoop, maxpar). \end{aligned}$$

Proof. Given an E0L system G, we construct the corresponding extended tissue P system Π as above in Theorem 1; then we immediately infer $N (G) = N (\Pi)$. On the other hand, given an extended tissue P system Π, by the constructions elaborated in Theorem 1, we obtain

$$N (\Pi) = N (G') = \{|x| \mid x \in h (L (G))\}$$

and therefore $N (\Pi) \in NE0L$. □

Corollary 2. *For* $X \in \{Ps, N\}$, $X0L = XtOP_1 (noncoop, maxpar)$.

Proof. This result immediately follows from the constructions elaborated in Theorem 1 with the specific restriction that for proving the inclusion $PstOP_1 (noncoop, maxpar) \subseteq Ps0L$ we can directly work with the symbols of V from the given non-extended tissue P system Π for the 0L system G to be constructed (instead of the symbols from $V \times \{1\}$) and thus do not need the projection h to get the desired result $L(\Pi) = L(G) \in Ps0L$. Besides this important technical detail, the results of this corollary directly follow from Theorem 1 and Corollary 1, because any non-extended system corresponds to an extended system where all symbols are terminal. □

For tissue P systems with only one cell, the non-cooperative rules can also be interpreted as antiport rules in the following sense: an antiport rule of the form a/x in a single-cell tissue P system means that the symbol a goes out to the environment and from there (every symbol is assumed to be available in the environment in an unbounded number) the multiset x enters the single cell. The families of Parikh sets and length sets generated by (extended, non-extended) one-cell tissue P systems using antiport rules of this specific form working in the maximally parallel derivation mode are denoted by $XEtOP_1 (anti_{1,*}, maxpar)$ and $XtOP_1 (anti_{1,*}, maxpar)$ for $X \in \{Ps, N\}$, respectively. We then get the following corollary:

Corollary 3. *For* $X \in \{Ps, N\}$,

$$XEtOP_1 (anti_{1,*}, maxpar) = XE0L$$

and

$$XtOP_1 (anti_{1,*}, maxpar) = X0L.$$

Proof. The results immediately follow from the previous results and the fact that the application of an antiport rule $a/b_1 \dots b_k$ has exactly the same effect on the contents of the single cell as the non-cooperative evolution rule $(a, 1) \to (b_1, 1) \dots (b_k, 1)$. □

For one-cell tissue P systems, we obtain a characterization of the families generated by the deterministic variants of these systems by the families generated by the corresponding variants of Lindenmayer systems:

Corollary 4. *For* $X \in \{Ps, N\}$ *and* $Y \in \{noncoop, anti_{1,*}\}$,

$$XED0L = XtEDOP_1 (Y, maxpar)$$

and

$$XD0L = XtDOP_1 (Y, maxpar).$$

Proof. As already mentioned in the proof of Corollary 2, the results immediately follow from the constructions elaborated in Theorem 1 with the specific restriction that for proving the inclusion $PsEDtOP_1\,(noncoop, maxpar) \subseteq PsED0L$ we can directly work with the symbols of V from the given (extended) deterministic tissue P system Π for the ED0L system G to be constructed (instead of the symbols from $V \times \{1\}$) and thus do not need the projection h to get the desired result $L\,(\Pi) = L\,(G) \in PsED0L$. The remaining statements follow from these constructions in a similar way as the results stated in Corollaries 1, 2, and 3. □

The constructions described in the proofs of Corollary 2 and 4 cannot be extended to (non-extended, deterministic) tissue P systems with an arbitrary number of cells, because in that case again the application of a projection h would be needed.

5 Conclusions and Future Research

In this paper we have shown that the Parikh sets as well as the length sets generated by (extended) tissue P systems with non-cooperative rules (without halting) coincide with the Parikh sets as well as the length sets generated by (extended) Lindenmayer systems.

In the future, we may also consider other variants of extracting results from computations in (extended) tissue P systems with non-cooperative rules, for example, variants of halting computations or only infinite computations, as well as other derivation modes as the sequential or the minimally parallel derivation mode. For the extraction of results, instead of the intersection with a terminal alphabet we may also use other criteria like the occurrence/absence of a specific symbol.

As inspired by the ideas elaborated in [3], we may investigate in more detail the evolution/behaviour of deterministic tissue P systems with non-cooperative rules based on the mathematical theory of Lindenmayer systems: as there is a one-to-one correspondence between deterministic tissue P systems with non-cooperative rules in one cell and D0L systems, the well-known mathematical theory for D0L systems can directly be used to describe/ investigate the behaviour of the corresponding deterministic tissue P systems with non-cooperative rules.

Acknowledgements. The authors gratefully acknowledge the interesting discussions with Miguel Gutiérrez-Naranjo and Mario Pérez-Jiménez on the ideas presented in their paper [3].

References

1. J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory*. EATCS Monograph in Theoretical Computer Science, Springer, Berlin, 1989.

2. R. Freund, Gh. Păun, and M.J. Pérez-Jiménez, Tissue-like P systems with channel states. *Theoretical Computer Science* **330** (2005), 101–116.
3. M.A. Gutiérrez-Naranjo and M.J. Pérez-Jiménez, Efficient computation in real-valued P systems, see this volume.
4. C. Martín-Vide, Gh. Păun, J. Pazos, and A. Rodríguez-Patón, Tissue P Systems, *Theoretical Computer Science*, **296** (2003), 295–326.
5. Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences*, **61** (2000), 108–143.
6. Gh. Păun, *Membrane Computing. An Introduction.* Natural Computing Series, Springer, Berlin, 2002.
7. G. Rozenberg and A. Salomaa, *The Mathematical Theory of L Systems.* Academic Press, New York, 1980.
8. G. Rozenberg and A. Salomaa (Eds), *Handbook of Formal Languages.* 3 volumes, Springer, Berlin, 1997.
9. The P Systems Web Page: `http://ppage.psystems.eu`.

A P System Modeling an Ecosystem Related to the Bearded Vulture

Mónica Cardona[1], M. Angels Colomer[1], Mario J. Pérez-Jiménez[2], Delfí Sanuy[3], Antoni Margalida[4]

[1] Dpt. of Mathematics, University of Lleida
Av. Alcalde Rovira Roure, 191. 25198 Lleida, Spain
E-mail: {mcardona,colomer}@matematica.udl.es

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

[3] Dpt. of Animal Production, University of Lleida
Av. Alcalde Rovira Roure, 191. 25198 Lleida, Spain
E-mail: dsanuy@prodan.udl.cat

[4] Bearded Vulture Study & Proteccion Group
Adpo. 43 E-25520 El Pont de Suert, Lleida, Spain
E-mail: margalida@inf.entorno.es

Summary. The Bearded Vulture is one of the rarest raptors in Europe and it is an endangered species. In this paper, we present a model of an ecosystem related with the Bearded Vulture which is located in the Catalan Pyrenees, by using P systems. The population dynamics constituted by the Bearded Vulture (that feeds almost exclusively on bones) and other five subfamilies that provide the bones they feed on, is studied.

P systems provide a high level computational modeling framework which integrates the structural and dynamical aspects of ecosystems in a comprehensive and relevant way. P systems explicitly represent the discrete character of the components of an ecosystem by using rewriting rules on multisets of objects which represent individuals of the population and bones. The inherent stochasticity and uncertainty in ecosystems is captured by using probabilistic strategies.

In order to give an experimental validation of the P system designed, we have constructed a simulator that allows us to analyze the evolution of the ecosystem with different initial conditions.

1 Introduction

An ecosystem is a natural unit consisting of all plants, animals and microorganisms (biotic factors) in an area functioning together with all of the non-living physical (abiotic) factors of the environment.

Animal species are interconnected in a food chain in which some species depend on others [6], [16]. The variations of different biomass affect the composition of the population structures [15]. In mountain ecosystems where livestock activities are developed in a traditional way, the relationship among ungulates and their predators has been disrupted by the presence of domestic animals [5]. Animals located at the top of the ecological pyramid are influenced by their presence and number. The abandonment of corpses on mountains is a major source of food for necrophagous animals [8].

The study of population ecology and how the species interact with the environment is one of the aspects of the conservation biology with more and more interest for managers and conservationists [2]. A widespread tool used are the ecological models, based on mathematical representations of ecological processes [13]. Ecosystems are dynamic entities composed of the biological community and the abiotic environment. An ecosystem's abiotic and biotic composition and structure is determined by the state of a number of interrelated environmental factors. Changes in any of these factors, such as nutrient availability, temperature, light intensity, grazing intensity, and species population density, will result in dynamic changes in the nature of these systems.

Scientists have recognized that organisms can be organized according to several different functional levels. The functional level known as species refers to a group of organisms that are similar in morphology and physiology and which have the ability to interbreed. A population is formed by all the different organisms of a single species occupying a specific area on the Earth. A community is defined as all the populations of different species inhabiting a particular region of the Earth. The most complex functional level of organization is the ecosystem. An ecosystem consists of the community and its relationship with the abiotic factors of the environment.

Ecosystems are primarily governed by stochastic events, the reactions they provoke on non-living materials and the responses by organisms to the conditions surrounding them. Thus, an ecosystem results from the sum of myriad individual responses of organisms to stimuli from non-living and living elements from the environment. As the number of species in an ecosystem increases, the number of stimuli also does. Ever since life began, organisms have survived continuous change through natural selection of successful feeding, reproductive and dispersal behavior. Ecosystems can be of any size. An ecosystem can be as large as a desert or a lake or as small as a tree or a puddle. A terrarium can be described as an artificial ecosystem.

Ecosystem models, or ecological models, are mathematical representations of ecosystems. Given the complexity of the problems to be modeled, simplifications need to be made in order to achieve a good numerical approach. Ecosystem models are a development of theoretical ecology that describes the major dynamics of ecosystems. Ecosystem models aim not only to integrate the understanding of these systems but also to be able to predict their behavior (in general terms, or in response to particular changes). Due to the complexity of ecosystems in terms of

numbers of species/ecological interactions, ecosystem models tend to simplify the systems they are studying down to a limited number of pragmatic components. These may be particular species of interest, or may be broad functional types such as autotrophs, heterotrophs or saprotrophs. In biogeochemistry, ecosystem models usually include representations of non-living "resources" such as nutrients, which are consumed by living components of the model.

The process of simplification reduces an ecosystem to a small number of state variables. Depending upon the system under study, these may represent ecological components in terms of numbers of discrete individuals or quantify the component continuously as a measure of the total biomass of all organisms of that type, often using a common model currency. The components are then linked together by mathematical functions that describe the nature of the relationships between them. For instance, in models which include predator-prey relationships, the two components are usually linked by some function that relates total prey captured to the populations of both predators and prey. Deriving these relationships is often extremely difficult given habitat heterogeneity, the details of component behavioral ecology (including issues such as perception, foraging behavior), and the difficulties involved in unobtrusively studying these relationships under field conditions. Typically, relationships are derived statistically or heuristically. For example, some standard functional forms describing these relationships are linear, quadratic, hyperbolic or sigmoid functions.

Besides establishing the components to be modeled and the relationships between them, another important factor in ecosystem model structure is the representation of space used. Historically, models have often ignored the confusing issue of space by using zero-dimensional approaches such as ordinary differential equations. As computing power increases in, models which incorporate space are being increasingly used, for example, based on partial differential equations or cellular automata. The inclusion of the factor of space allows dynamics not present in non-spatial frameworks to be considered, and sheds light on processes that lead to the formation of patterns in ecological systems. One of the earliest and best-known ecological models is the predator-prey model described by Alfred J. Lotka (1925) and Vito Volterra (1926). This model is composed of a pair of ordinary differential equations where one represents a prey species and the other its predator.

Volterra originally devised the model to explain fluctuations in fish and shark populations observed in the Adriatic Sea following the First World War when fishing had been curtailed. However, the equations have subsequently been applied more generally. Although simple, they illustrate some of the salient features of ecological models: the considered biological populations grow, interact with other populations (either as predators, prey or competitors), and suffer mortality.

The objective of this paper is to design a model that studies the evolution of an ecosystem located in the Pyrenees, taking advantage of the capacity the P Systems to work in parallel. The ecosystem includes six species: the Bearded Vulture (Gypaetus barbatus) as scavenger (predator) species and the Pyrenean Chamois (Rupicapra pyrenaica), Red Deer (Cervus elaphus), Fallow Deer (Dama

dama), Roe Deer (Capreolus capreolus) and Sheep (Ovis capra) as carrion (prey) species.

The paper is structured as follows. In the next section, basic concepts of the ecosystem to be modeled are introduced. The most outstanding aspects of each species are detailed as well as interactions among them. In Section 3, a probabilistic P system is presented in order to describe the ecosystem. A simulator of that probabilistic P system is designed in Section 4, in order to study the dynamics of the ecosystem. Section 5 is devoted to the analysis of the results produced by the simulator. Finally, conclusions are presented in the last section.

2 Modeling the Ecosystem

The ecosystem to be modeled is located in the Catalan Pyrenees, in the Northeast of Spain. This area contains a total of 35 territories that constitutes the 34.3% of the population of the Bearded Vulture Spanish population in 2007 (n = 102). See Figure 1 [8].

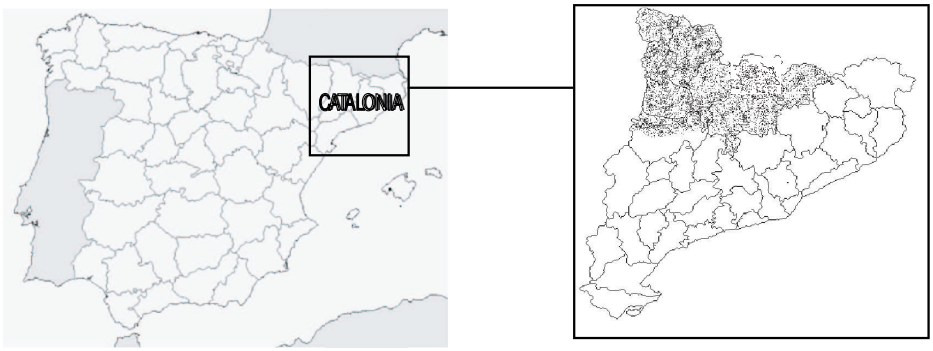


Fig. 1. Regional distribution of the Bearded Vulture in the Catalan Pyrenees (NE Spain).

The ecosystem is composed of six species: the Bearded Vulture (predator species) and the Pyrenean Chamois, Red Deer, Fallow Deer, Roe Deer, and Sheep (prey species). The last five prey species belong to the bovid family, they are herbivores and their bone remains form the basic source of nourishment for the Bearded Vulture in the Pyrenees.

The Bearded Vulture (Gypaetus barbatus) is a cliff-nesting and territorial large scavenger distributed in mountains ranges in Eurasia and Africa. It is one of the rarest raptors in Europe (150 breeding pairs in 2007), it inhabits areas of high altitude (1500-4000 m), though it can be seen in areas of lower altitude (500-800 m) in the winter when high mountains are covered with snow. The mean lifespan of wild Bearded Vultures is 21.4 years [4]. The mean age of first breeding is 8.1 years,

whereas the mean age of first successful breeding is 11.4 years [1]. Egg-laying take places during December-February and, after 52-54 days of incubation and around 120 days of chick-rearing, the chick abandons the nest between June-August [12]. The clutch size in this species is usually two eggs, but only one chick survives as a consequence of sibling aggression [11]. Bearded Vultures are fertile from the age of eight, when they become adult, and they cease to be fertile at the age of twenty. The female's annual fertility rate in Catalonia during the last five years is estimated around 38% [8], the female lays two eggs and incubates them for 55-57 days. However, as with most birds of prey, only one of the youngsters fledges.

The Bearded Vulture is the only vertebrate animal that feeds almost exclusively on bone remains. Its main food source is bone remains of dead small and medium-sized animals. It searches for food either alone or in pairs. In the Pyrenees the bone remains of Pyrenean Chamois, Red Deer, Fallow Deer, Roe Deer, and Sheep form 67% of the vulture's food resources of nourishment, and the other 33% includes the bone remains of small size mammals (e.g., dogs, cats), badgers, least weasel, large mammals (cows, horses), medium size mammals (e.g., wild boars) and birds [8]. A pair of Bearded Vultures needs an average 341 Kg of bones per year [10], [9].

During the dispersal period (from fledgling until the bird become territorial at 5-6 years), the non-adult Bearded Vultures birds cover large distances surveying different areas. For example, the averaged surface covered by four young monitored after fledging was 4932 km^2 (range 950-10294 km^2, [14]). They may return to breed in the area where they were born but it is not frequent. The choice of the area where they settle down definitively is based on different parameters such as the availability of food and the vulture population density. Breeding mature birds are territorial and the approximate home ranges obtained for eight pairs studied varied between 250 km^2 and 650 km^2. They almost never leave their territory to settle in neighboring mountainous areas even if these are close by, and this fact makes it more difficult to estimate the growth of its population. Studies by Margalida estimate the average annual growth in the population of Bearded Vultures in the Pyrenees to be 4-53% of the existing population, and that the average floating population is 20 principally remains in feeding stations situated in the central Pyrenees (Aragon) birds.

The natural behavior of the five bovid species is similar as they are all herbivores and they all reach the size of the adult animal when they are one year old. In general, they arrive at the sexual maturity within two years from birth. Chamois and the Red Deer have a longer life expectancy than Fallow Deer and Roe Deer. The natural mortality rates are similar in all five species, in the first year of life it is calculated to be 50% and 6% during the remaining years. In spite of the great degree of similarity among these five species, there are differences among them, some are of natural origin and other are induced by human action. It is essential to bear them in mind in order to define a P system that can reliably simulate the ecosystem.

Red Deer are very much appreciated by the hunters, not for their meat but as trophies and so only males are hunted, and this causes the natural evolution of the population to be modified. The hunter only takes the head as a trophy leaving the animal's body on the field, and so the carcass is eaten by other species and the bones may then be eaten by the Bearded Vulture.

Fallow Deer and Roe Deer live in areas that are difficult to reach and for this reason the Bearded Vulture cannot take advantage of the bones of all of the dead animals. Studies estimate that the Bearded Vulture only uses 20% of the bones available on the field.

As sheep are domestic animals, humans exert a high level of control over the sheep populations. The size and growth of the sheep population is limited by the owners of the flocks. The natural average life expectancy of sheep is longer that their actual life expectancy in the field because when its fertility rate decreases at the age of eight, they are taken out of the habitat. Most of the lambs are sold to market and so they are taken out of the habitat too in the first year of life. Only 20% of the lambs, mostly females, are left on the field and they are used to replace sheep that have died naturally and the old ones that have been removed from the flock.

As the Bearded Vulture is an endangered species, there are many projects that study its behavior and the way it is affected by its environment. Thanks to these studies there is available a large amount of information which is required to define the P System and to validate the results obtained.

In this study, the feeding of the Bearded Vulture is dependent on the evolution of the P System. However, the P System does not consider that the availability of food limits the feeding of the herbivores, and so the growth of the vegetation is not modeled.

Taking all of this background information into consideration, the following data was required for each species:

- $k_{i,1}$: age at which adult size is reached. This is the age at which the animal eats like the adult does, and at which if the animal dies, the amount of biomass it leaves is similar to the ole 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 starts to be fertile.
- $k_{i,3}$: age at which it stops being fertile.
- $k_{i,4}$: average life expectancy.
- $k_{i,5}$: fertile ratio (number of descendants by fertile female).
- $k_{i,6}$: population growth.
- $k_{i,7}$: mortality ratio in first years ($age < k_{i,1}$) in which biomass in the form of bones is not left on the field.
- $k_{i,8}$: mortality ratio in first years ($age < k_{i,1}$) in which biomass in the form of bones is left on the field.
- $k_{i,9}$: mortality ratio in adult animals ($age \geq k_{i,11}$) in which biomass in the form of bones is not left on the field.

- $k_{i,10}$: mortality ratio in adults animals ($age \geq k_{i,1}$) in which biomass in the form of bones is left on the field.
- $k_{i,11}$ is equal to 1 if the animal dies at the age of $k_{i,4}$ leaving biomass, and it is equal to 0 if the animal dies at the age of $k_{i,4}$ without leaving bones.
- $k_{i,12}$: amount of bones from young animals ($age < k_{i,1}$).
- $k_{i,13}$: amount of bones from adult animals ($age \geq k_{i,1}$).
- $k_{i,14}$: percentage of females in the population.
- $k_{i,15}$: type of food.
- $k_{i,16}$: amount of food necessary per year and animal (1 unit is equal 0.5 kg of bones).

When an animal dies, the weight of bones which it leaves is around 20% of its total weight. Table 1 shows the average weight of each animal as well as the weight of bones they leave. In the case of Fallow Deer and Roe Deer, the value of the weight of bones must then be multiplied by 0,2 (20%) which is the proportion of bones that are accessible for the Bearded Vulture.

Table 1. Bones

Species	Weigh Male	Weigh Female	Percentage Female	Average weigh	Biomass: bons adult	Biomass: bons young	Kg accessible by B.Vulture (adult/young)
B.Vulture	5	6.5	60	2	-	-	-
Chamois	28	32	50	30	6	3	6/3
Red Deer Female	-	75	-	75	15	7.5	15/7.5
Red Deer Male	120	-	-	120	24	12	24/12
Fallow Deer	63	42	80	46	9	4.5	2/1
Roe Deer	27	23	66	24	5	2.5	1/0.5
Sheep	42	35	97	35.2	7	3.5	7/3.5

33% of the Bearded Vulture's nutrition is formed by bone remains of other species that belong to the ecosystem but which are not studied in this model, and so it was necessary for the P system to include an annual contribution from the ecosystem. This contribution was calculated to be 12500 kg bones by considering the size of the Bearded Vulture population and its weight. By subtracting the 3500 kg of food in form of bones that is consumed by the floating population of Bearded Vultures, the annual external contribution is calculated to be 9000 kg of bones. There are seven feeding stations in Catalonia which provide around 10500 kg of bone remains annually. These artificial feeding sites have not been considered in the study and most of the floating birds feed at these sites.

The required information about each species is shown in Table 2.

Table 2. Constants

Specie	i	$k_{i,1}$	$k_{i,2}$	$k_{i,3}$	$k_{i,4}$	$k_{i,5}$	$k_{i,6}$	$k_{i,7}$	$k_{i,8}$	$k_{i,9}$	$k_{i,10}$	$k_{i,11}$	$k_{i,12}$	$k_{i,13}$	$k_{i,14}$	$k_{i,15}$	$k_{i,16}$
B.Vulture	1	1	8	20	21	0	3	6	0	7	0	0	0	0	50	bones	341
Chamois	2	1	2	18	18	75	0	0	55	0	6	1	6	12	55	grass	0
Red Deer Female	3	1	2	13	17	50	0	0	50	0	6	1	15	30	100	grass	0
Red Deer Male	4	1	2	18	20	0	0	0	50	0	36	1	24	48	0	grass	0
Fallow Deer	5	1	2	11	12	50	0	0	50	0	7	1	2	4	75	grass	0
Roe Deer	6	1	2	9	10	90	0	0	55	0	5	1	1	2	67	grass	0
Sheep	7	1	2	7	7	75	0	59	15	0	4	0	7	14	96	grass	0

3 A P System Based Model of the Ecosystem

In this section we present a model of the ecosystem described in Section 2 by means of probabilistic P systems. We will study the behavior of this ecosystem in diverse initial conditions.

First, we define the P systems based framework (probabilistic P systems), where additional features such as two electrical charges which describe specific properties in a better way, are used.

Definition 1. *A probabilistic P system of degree n is a tuple*

$$\Pi = (\Gamma, \mu, \mathcal{M}_0, \ldots, \mathcal{M}_{n-1}, R)$$

where:

- Γ *is the alphabet (finite and nonempty) of objects (the working alphabet).*
- μ *is a membrane structure, consisting of n membranes, labeled* $0, 1, \ldots, n-1$. *The skin membrane is labeled by 0. We also associate electrical charges with membranes from the set* $\{0, +\}$, *neutral and positive.*
- $\mathcal{M}_0, \ldots, \mathcal{M}_{n-1}$ *are strings over* Γ, *describing the multisets of objects initially placed in the n regions of* μ.
- R *is a finite set of evolution rules. An evolution rule is of the form* $r : u \xrightarrow{k} v$, *where* u, v *are a multiset over* Γ *and* $k \in [0, 1]$ *is a real number between 0 and 1 associated with the rule.*

We assume that a global clock exists, marking the time for the whole system (for all compartments of the system); that is, all membranes and the application of all rules are synchronized.

The n-tuple of multisets of objects present at any moment in the n *regions* of the system constitutes the *configuration* of the system at that moment. The tuple $(\mathcal{M}_0, \ldots, \mathcal{M}_{n-1})$ is the initial configuration of the system.

We can pass from one configuration to another one by using the rules from R 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 and all occurrences of the left–hand side of the rules are consumed, as usual.

3.1 The model

Our model consists in the following probabilistic P system of degree 2 with two electrical charges:

$$\Pi = (\Gamma, \mu, \mathcal{M}_0, \mathcal{M}_1, R)$$

where:

- 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 state); it also contains the auxiliary symbols B and C.

$$\Gamma = \{X_{ij},\ Y_{ij},\ Z_{ij} :\ 1 \leq i \leq 7,\ 0 \leq j \leq k_{i,5}\} \cup \{\ B,\ C\}$$

- In the membrane structure we represent two regions, the skin (where animals reproduce) and an inner membrane (where animals feed and die): $\mu = [\ [\ ]_1\ ]_0$ (neutral polarization will be omitted)
- In $\mathcal{M}_0$ and $\mathcal{M}_1$ we specify the initial number of objects present in each regions (encoding the initial population and the initial food).
  - $\mathcal{M}_0 = \{X_{ij}^{q_{ij}} : 1 \leq i \leq 7,\ 0 \leq j \leq k_{i,}\}$, where the multiplicity q_{ij} indicates the number of animals, of species i whose age is j that are initially present in the ecosystem.
  - $\mathcal{M}_1 = \{C\ B^{18000}\}$, where the object B represent 0.5 kg of bones, and 9000 kg is the external contribution of bones to the P system corresponding to the 33% of feeding that come from animals do not modeled in the P system.
- The set R of evolution rules consists of:
  - Reproduction-rules

    Adult males:

    $$r_0 \equiv [X_{ij} \xrightarrow{1-k_{i,14}} Y_{ij}]_0, 1 \leq i \leq 7, 0 \leq j \leq k_{i,4}$$

    Adult females that reproduce:

    $$r_1 \equiv [X_{ij} \xrightarrow{k_{i,5}k_{i,14}} Y_{ij}Y_{i0}]_0,\ 1 \leq i \leq 7,\ k_{i,2} \leq j < k_{i,3}$$

    Adult females that do not reproduce:

    $$r_2 \equiv [X_{ij} \xrightarrow{(1-k_{i,5})k_{i,14}} Y_{ij}]_0,\ 1 \leq i \leq 7,\ k_{i,2} \leq j < k_{i,3}$$

    Young animals that do not reproduce:

    $$r_3 \equiv [X_{ij} \rightarrow Y_{ij}]_0,\ 1 \leq i \leq 7,\ k_{i,3} \leq j < k_{i,2}$$

  - Young animals mortality rules:

    Those which survive:

$$r_4 \equiv Y_{ij}[\,]_1 \xrightarrow{1-k_{i,7}-k_{i,8}} [Z_{ij}]_1 :\ 1 \le i \le 7,\ 0 \le j < k_{i,1}$$

Those which die and leaving bones:

$$r_5 \equiv Y_{ij}[\,]_1 \xrightarrow{k_{i,8}} [B^{k_{i,12}}]_1 :\ 1 \le i \le 7,\ 0 \le j < k_{i,1}$$

Those which die and do not leave bones:

$$r_6 \equiv Y_{ij}[\,]_1 \xrightarrow{k_{i,7}} [\,]_1 :\ 1 \le i \le 7,\ 0 \le j < k_{i,1}$$

– Adult animals mortality rules:

Those which survive:

$$r_7 \equiv Y_{ij}[\,]_1 \xrightarrow{1-k_{i,9}-k_{i,10}} [Z_{ij}]_1 :\ 1 \le i \le 7,\ k_{i,1} \le j < k_{i,4}$$

Those which die leaving bones:

$$r_8 \equiv Y_{ij}[\,]_1 \xrightarrow{k_{i,10}} [B^{k_{i,13}}]_1 :\ 1 \le i \le 7,\ k_{i,1} \le j < k_{i,4}$$

Those which die and do not leave bones:

$$r_9 \equiv Y_{ij}[\,]_1 \xrightarrow{k_{i,9}} [\,]_1 :\ 1 \le i \le 7,\ k_{i,1} \le j < k_{1,4}$$

Animals that die at an average life expectancy:

$$r_{10} \equiv Y_{ij}[\,]_1 \to [B^{k_{i,13} \cdot k_{i,11}}]_1 :\ 1 \le i \le 7,\ j = k_{i,4}$$

– Feeding rules:

$$r_{11} \equiv [Z_{ij} B^{k_{i,16}}]_1 \to X_{ij+1}[\,]_1^+ :\ 1 \le i \le 7,\ 0 \le j \le k_{i,4}$$

– Rules of mortality due to lack of food, and the elimination from the system of bones that are not eaten by the Bearded Vulture:

Elimination of remaining bones:

$$r_{12} \equiv [B]_1^+ \to [\,]_1$$

External contribution that represent the bones:

$$r_{13} \equiv [C]_1^+ \to [CB^{18000}]_1$$

Adult animals that die because they have not enough food:

$$r_{14} \equiv [Z_{ij}]_1^+ \to [B^{k_{i,13} \cdot k_{i,11}}]_1 :\ 1 \le i \le 7,\ k_{i,1} \le j \le k_{i,4}$$

Young animals that die because they have not enough food:

$$r_{15} \equiv [Z_{ij}]_1^+ \to [B^{k_{i,12} \cdot k_{i,11}}]_1 :\ 1 \le i \le 7,\ j < k_{i,1}$$

Figure 2 gives a schematic view of how the P system works.

4 A Simulator

In order to study the dynamics of the species that belong to the ecosystem, we have designed a simulator written in $C++$ language. This program runs on a PC.

In the simulation, the objects that encode the species and the age are represented by two vectors which are related through the number assigned to each animal of the ecosystem. The objects of the P system evolve in a random way; this

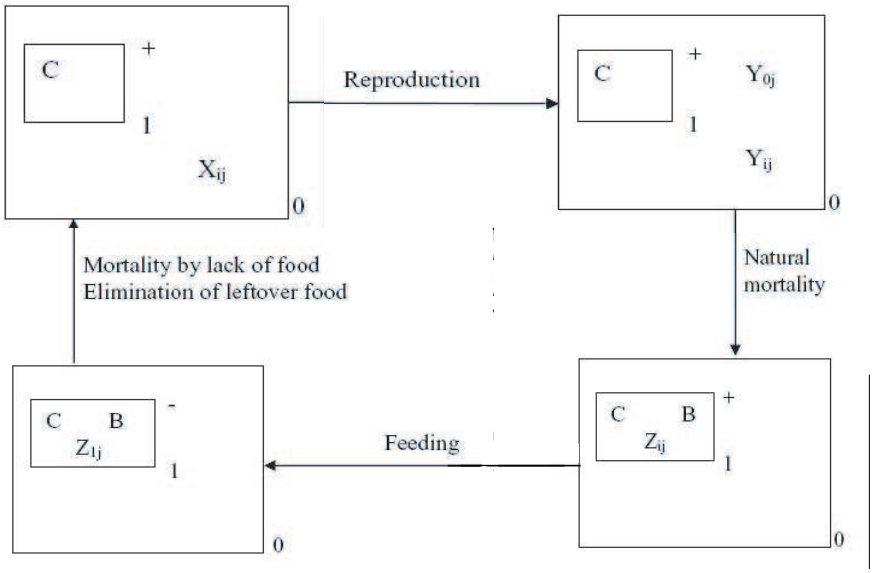


Fig. 2. Schema

stochasticity is implemented by generating random numbers between 1 and 100, according to an uniform distribution. One of the generated numbers is assigned to each animal. Then, the animal evolves according to the assigned number and the constant probability. For example, when the probability of surviving is 70%, the animal will die if the assigned number is greater than 70.

The input of the program consists of the parameters of each species that are considered in the P system and the number of animals of each species and age that are present at time zero. The output is the number of animals and age of each species that are present every year after completing the following processes: reproduction, mortality and feeding.

In nature an ecosystem is governed by nondeterminism, and this implies a complex mathematical model. Nevertheless all the processes that are carried out have an important degree of randomness. This randomness can be predicted and can be quantified at every moment and situation of the ecosystem.

The program has been structured in four modules:

- *Reproduction.* The inputs are the age at which each species begins to be fertile, the age at which it stops being fertile, the fertility rate, and the proportion of females of the species. This module also requires the total number of existing animals and the distribution of these animals in terms of species and ages. The output of this module is the number and age of animals of each species.

- *Mortality.* The inputs are the mortality rate based on the age, the average life expectancy of each species, and finally the weight of bones left by the dead animal which is dependent on its age. Once again this module also requires the total number of animals and their distribution in terms of species and ages. As with the other modules, the output of this module is the number and age of animals of each species when the process is completed. Another output of this module is the amount of food that is generated in terms of the weight of bones produced that provide the Bearded Vulture's basic source of nourishment.
- *Feeding.* The inputs are the amount of food available in the ecosystem and the annual amount of food that is necessary for the animal to survive in suitable conditions, in other words, conditions under which the animals are not debilitated and do not suffer the consequent effects on their capabilities. As was seen in the previous modules, inputs are generated by the P system itself as it quantifies objects representing the number of animals of each existing species and age. Once again, the output of this module is the number and age of animals of each species.
- *Elimination* of unused leftover food and the animal mortality from insufficient feeding. The input of this module is part of output of the feeding module. The aim of this process is to eliminate the number of animals that were not able to find the necessary amount of food for their survival, and also to consider the amount of leftover food that is degraded with time and that therefore stops having a role in the model. The animals that die due to a lack of food are transformed into bones that can then be eaten by the Bearded Vulture. The output of this module is an amount of food in form of bones that is available to the Bearded Vulture.

The unit of reference used in this study is the year, that is, the food consumed throughout an annual period is given at one single point in time, and with one application of each rule. The mortality of animals in an ecosystem is also a process that is carried out in a continuous way, throughout the year. However, reproduction is an activity that takes place at a specific time of the year, and moreover, it takes place at the same time for all of the species considered in this study. It will be necessary to verify if the one year unit time which was chosen is correct or whether a shorter inferior unit of time should be used in the P system. It was also necessary to check the robustness of the proposed model and to do this, it was ran a second time with a modified order of application of the four processes modules. Given independence of the four modules that form the P system, it would be a simple exercise to run probability experiments with each module.

5 Results and Discussions

In order to validate the P system, it is necessary to check its robustness. Considering the year as unit time it was necessary to discretize feeding and mortality variables.

According to the design of the P system, reproduction rules have a higher priority than mortality ones. Then, the robustness of the model regarding the change of that priority is analyzed. For that reason, two variants of the simulator have been studied changing the order of the corresponding modules (in the P system we change variable X by variable Y in the initial multiset $\mathcal{M}_0$).

The data used to verify the robustness of the P system correspond to the present situation (2008) in the Catalan Pyrenees of the ecosystem formed by the Bearded Vulture, Chamois, Red Deer, Fallow Deer, Roe Deer and Sheep (see Table 3).

Table 3. Number of animals, at the moment, in the Pyrenean Catalan

Species	Number
Bearded Vulture	74
Chamois	12000
Red deer female	4400
Red deer male	1100
Fallow deer	900
Roe deer	10000
Sheep	200000

As shown in Table 3, the data of the current number of animals in the Catalan Pyrenees does not indicate the ages of animals. An age distribution has been generated considering the different constants that affect the animals throughout their life. These constants are fertility rate, mortality rate and percentage of females in the population. The age distribution obtained is shown in Table 4.

In both cases, the simulator was ran 10 times until it covered a period of 20 years.

In these figures, solid lines and dashed lines represent the population dynamics when the simulator modules are applied following the *orders* reproduction–mortality–feeding and mortality–feeding–reproduction, respectively. Taking into account that the P system behavior is similar in both cases, it can be deduced that our model is robust with regard to the properties considered.

The very important factor of population density was not considered in the model of the ecosystem. This is the reason why the population of some of the species grew in an exponential way reaching values which cannot be obtained in the ecosystem. It is well-known, for example, that when a population of Red Deer reaches a level of 15000 animals, a regulation process starts that implies a drastic decrease of the population down to 1000 individuals.

6 Conclusions

A probabilistic P System modeling an ecosystem related with the Bearded Vulture which is located in the Catalan Pyrinees, has been presented.

Table 4. Number of animals for age

Age	B. Vulture	Chamois	Red deer female	Red deer male	Fallow deer	Roe deer	Sheep
1	0	988	978	254	125	1210	31246
2	0	987	780	192	110	1207	30310
3	0	890	625	154	103	1207	29400
4	0	889	500	124	95	1207	28519
5	0	889	400	99	89	1085	27663
6	0	795	240	60	83	1083	26834
7	0	792	195	48	77	1083	26028
8	6	690	155	38	71	959	0
9	6	689	123	30	52	959	0
10	6	592	97	24	50	0	0
11	6	592	78	20	45	0	0
12	5	592	62	16	0	0	0
13	5	497	50	12	0	0	0
14	5	497	40	10	0	0	0
15	5	496	32	8	0	0	0
16	5	395	25	6	0	0	0
17	5	394	20	5	0	0	0
18	5	336	0	0	0	0	0
19	5	0	0	0	0	0	0
20	5	0	0	0	0	0	0
21	5	0	0	0	0	0	0

By using this P system has been possible to study the dynamics of the ecosystem modifying the framework in order to calculate how the ecosystem would evolve if different biological factors were modified either by nature or through human intervention.

A simulator of the P system has been designed and the robustness of the model with respect the order of application of the different kinds of rules, has been shown.

The P system does not consider levels of population density and, as a consequence, an exponential growth of populations of species was obtained. In a future work this factor and other parameters (e.g., the amount of food of the herbivores species, the climatic changes in the ecosystem, etc.) should be considered.

Acknowledgement

The third author acknowledges the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, co–financed by FEDER funds, and of the Project of Excellence TIC 581 of the Junta de Andalucia.

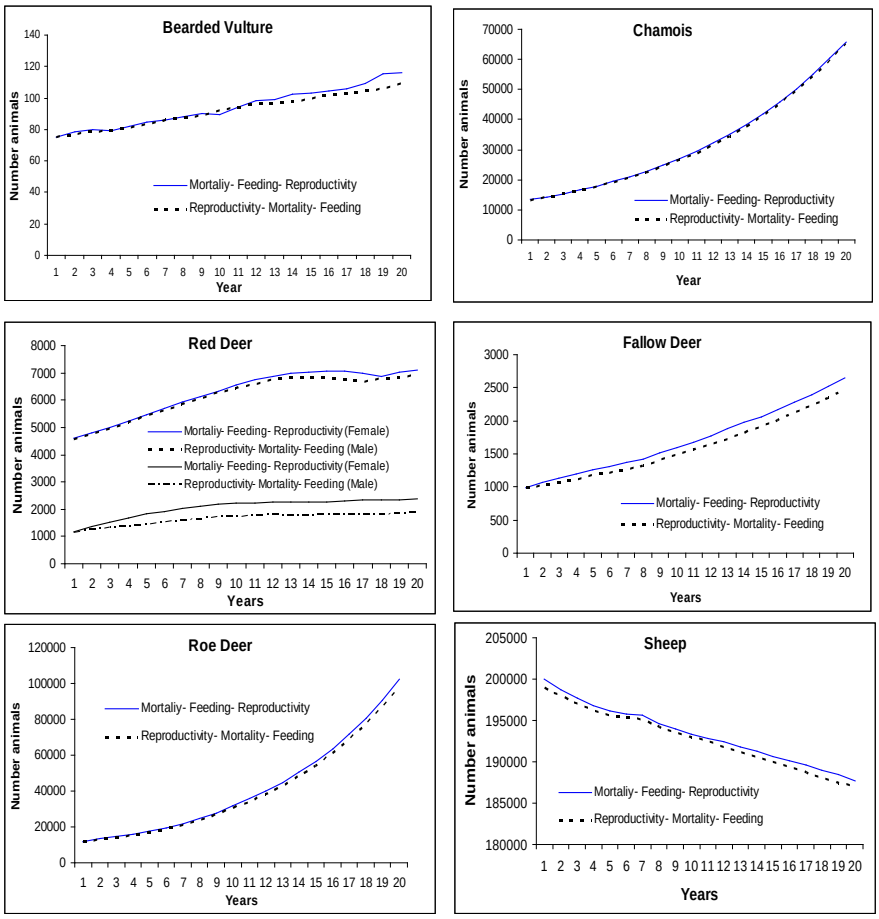


Fig. 3. Robustness of the ecosystem

References

1. R.J. Antor, A. Margalida, H. Frey, R. Heredia, L. Lorente, J.A. Sesé. Age of first breeding in wild and captive populations of Bearded Vultures (Gypaetus barbatus). *Acta Ornithologica*, **42** (2007), 114–118.
2. M. Begon, J.L. Harper, C.R. Townsend. *Ecology: Individuals, Populations and Communities*. Blackwell Scientific Publications Inc., Oxford, UK, 1988.
3. D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri. Modelling metapopulations with stochastic membrane systems. *BioSystems*, **91**, 3 (2007), 499–514.
4. C.J. Brown. Population dynamics of the bearded vulture Gypaetus barbatus in southern Africa. *African Journal of Ecology*, **35** (1997), 53–63.

5. C. Chocarro, R. Fanlo, F. Fillat, P. Marín. Historical evolution of natural resource use in the central Pyrenees of Spain. *Mountain Research And Development* , **10** (1990), 257–265.
6. P.H. Harvey, A. Purvis. Understanding the ecological and evolutionary reasons for life history variation: mammals as a case study. In McGlade J., ed. *Advanced ecological theory: principles and applications*. Blackwell Science Publications, Oxford, 1999, pp. 232–247.
7. J. Del Hoyo, A. Elliott, J. Sargatal. *Handbook of the Birds of theWorld, vol. 2.*. Lynx Edicions, Barcelona, 1994.
8. A. Margalida, D. García, A. Cortés-Avizanda. Factors influencing the breeding density of Bearded Vultures, Egyptian Vultures and Eurasian Griffon Vultures in Catalonia (NE Spain): management implications. *Animal Biodiversity and Conservation*, **30**, 2 (2007), 189–200.
9. A. Margalida, S. Mañosa, J. Bertran, D. García. Biases in studying the diet of the Bearded Vulture. *The Journal of Wildlife Management*, **71**, 5 (2006),1621–1625.
10. A. Margalida, J. Bertran, J. Boudet. Assessing the diet of nestling Bearded Vultures: a comparison between direct observation methods, *Journal of Field Ornithology*, **76**, 1 (2005), 40–45.
11. A. Margalida, J. Bertran, J. Boudet, R. Heredia. Hatching asynchrony, sibling aggression and cannibalism in the Bearded Vulture (Gypaetus barbatus). *Ibis*, **146**, (2004) 386–393.
12. A. Margalida, D. García, J. Bertran, R. Heredia. Breeding biology and success of the Bearded Vulture Gypaetus barbatus in the eastern Pyrenees. *Ibis*, **145**, (2003) 244–252.
13. H. McCallum. *Population Parameters: Estimation for Ecological Models*. Blackwell Science Publications, Oxford, 2000.
14. C. Sunyer. El periodo de emancipación en el Quebrantahuesos: consideraciones sobre su conservación. In R. Heredia, B. Heredia, eds. *El Quebrantahuesos (Gypaetus barbatus) en los Pirineos* Colección Técnica. Madrid, ICONA, 1991, pp. 47–65.
15. D. Tilman. *Resource Competition and Community Structure*. Princeton University Press, Princeton, New Jersey, 1982.
16. A. Watson. *Animal Populations in Relation to Their Food Resources*. Blackwell Scientific Publications, Oxford, 1970.

On Modeling Signal Transduction Networks

Alberto Castellini, Giuditta Franco

Verona University, Computer Science Dept.
Strada Le Grazie 15, 37134 Verona, Italy
E-mails: {alberto.castellini, giuditta.franco}@univr.it

Summary. Signal transduction networks are very complex processes employed by the living cell to suitably react to environmental stimuli. Qualitative and quantitative computational models play an increasingly important role in the representation of these networks and in the search of new insights about these phenomena. In this work we analyze some graph-based models used to discover qualitative properties of such networks. In turn, we show that MP systems can naturally extend these graph-based models by adding some qualitative elements. The case study of integrins activation during the lymphocyte recruitment, a crucial phenomenon in inflammatory processes, is described, and a first MP graph for this network is designed. Finally, we discuss some open problems related to the qualitative modeling of signaling networks.

1 Introduction

Biological signal transduction is a series of processes employed by the living cell to convert signals coming from the external environment [17, 1]. Signal conversions usually involve sequences of chemical reactions among proteins which generate complex networks. By these mechanisms the cell can suitably react in order to accomplish its biological functions.

The discovery of hidden interaction mechanisms supports the development of new medical treatments and drugs for specific diseases, thus, in the last decades many efforts have been addressed to "decipher" the interactions among the actors of signaling networks. Despite of the great medical and pharmaceutical interest for these networks, a lot of them are still completely or partially unknown, because of their huge size and high complexity. So far, the best results have been achieved at a qualitative level, by the representation and the analysis of protein interactions [18]. Such results have been mainly reached by means of suitable data structures (mainly graphs) that support the topological analysis of biological networks, while an open challenge concerns the modeling of dynamics related to the protein activation. This could provide significant improvements for the prediction of signaling network behaviors.

Both *qualitative* and *quantitative* modeling of metabolic and signal transduction networks have drawn large advantages from the use of computational models, which disclose new features of these systems by processing a great deal of data [33]. The most used mathematical models for the dynamical analysis of biological networks are the traditional systems of ordinary differential equations (ODE). They represent a biological system by a set of mathematical equations, where every equation rules the temporal evolution of a substance. Unfortunately, ODE models have some drawbacks, such as the hard definition of equations from observed phenomena, since a deep knowledge about microscopic molecular kinetics is required [20]. Thus, several discrete and bio-inspired models have been proposed, in order to symbolically describe cellular processes.

P systems, or *membrane systems*, were introduced in [28] as a new computational model which takes its inspiration from the structure and functioning of the living cell. This model is rooted in the context of formal language theory and it is based on *multisets* and *membranes* rewriting for which many computational universality results have been achieved [29]. This mathematical framework has been often employed to model biological processes underlying metabolism [2, 5, 27], pathologies [14, 13] and signaling [12, 26, 32].

Metabolic P systems, shortly MP systems, were recently introduced in [24] and developed, along with different lines, see for example [6, 21, 22, 23]. The aim of this non-conventional mathematical framework is properly the modeling of metabolic dynamics. In fact the development of this model has been based on the *mass partition principle*, which defines the transformation rate of object populations, according to a suitable generalization of chemical laws [20].

Some equivalence results have been proved in [11] and in [9] between MP systems and, respectively, autonomous ODE and Hybrid Functional Petri nets. The dynamics of several biological processes has been effectively modeled by means of MP systems [3], among them: the Belousov-Zhabotinsky reaction (in the Brusselator formulation) [5, 6], the Lotka-Volterra dynamics [5, 24], the SIR (Susceptible-Infected-Recovered) epidemic [5, 3], the leukocyte selective recruitment in the immune response [5], the Protein Kinase C activation [6], the circadian rhythms [3], the mitotic cycles in early amphibian embryos [23, 12], a Pseudomonas quorum sensing model [7] and the *lac* operon gene regulatory mechanism in glycolytic pathway [9].

In this work we investigate the possibility to employ MP systems for modeling signal transduction networks. We point out that some chemical laws which regulate metabolic processes cannot be applied to signaling processes, and several measurement problems can arise, due to current experimental limitations. With respect to models of metabolism, the focus has to be moved from substance transformations (complex creation and disintegration) to protein activation (e.g., phosphorylation and dephosphorylation).

In section 2 it is presented a qualitative model currently used to represent protein interactions [18], and some motivations and problems related to quantitative models are discussed. In Section 3 it is proposed the case study of a signaling

network involved in the lymphocyte recruitment, while in the last section some approaches are addressed for modeling the above network by MP systems.

2 From Qualitative to Quantitative Modeling

Several biological systems can be represented, at a *qualitative* level, by a network of elements connected by functional interactions. Some examples are the nervous system (physically connected neurons), the immune system (where cells and molecules are connected with different kinds of interaction) and also signaling systems (physically connected intracellular molecules). These networks can be symbolically represented by *graphs*, mathematical structures used to model pairwise relations (*edges*) between objects (*nodes*) from a certain collection. In signal transduction networks, nodes act as molecules, typically proteins, and arcs represent the capability of a molecule to activate or deactivate another molecule (Figure 1).

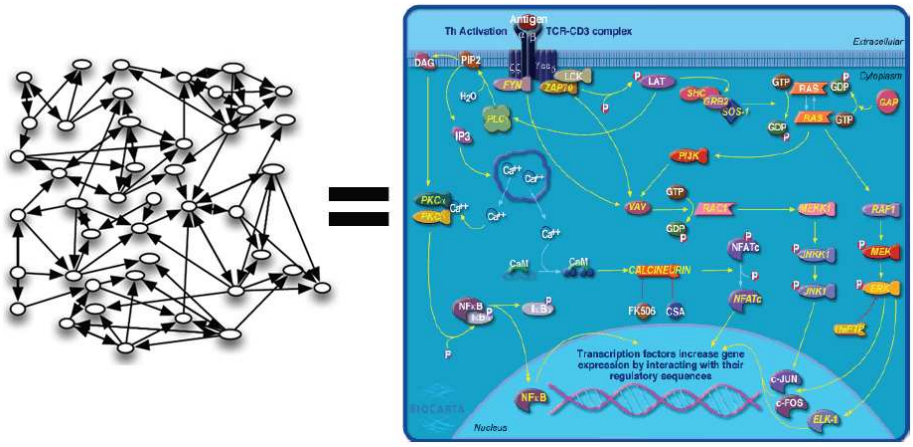


Fig. 1. A qualitative representation of a signal transduction network by a directed graph. Nodes correspond to molecules and arcs represent activation or deactivation [18].

Graph theory investigates structural properties of graphs, and these properties can assume a biological meaning if the graph represents a specific biological network. Topological analyses based on graph theory often address interesting questions about single elements of the network, clusters of elements, or the entire network. There exist specific computational tools [31] to visualize networks and analyze topological properties, such as the node degree (the number of edges connected to a node), the average degree of the network, the shortest path between two nodes, the average shortest path of the network, the network diameter,

clustering coefficients, the presence of network motifs (e.g., loops) or network statistical properties [18]. In a signal transduction network, the knowledge of such properties most of the times allows the identification of substances which regulate the process.

The qualitative approach described above is valid for a statical analysis of networks. While structural properties can be found, the temporal evolution of a network instead cannot be "simulated" to forecast dynamical behaviors (e.g., oscillations) occurring under different conditions. *Quantitative* models aim at extending qualitative models to achieve this target. They rely on new experimental methods which support the measurement of real systems (e.g., activation levels or concentration values), absolutely necessary to validate models. Namely, in the last decade, the wide improvements of high-throughput data acquisition techniques in molecular biology have made possible to screen and to analyze the expression of entire genomes, as well as to assess large numbers of proteins and to characterize in detail the metabolic state of a cell population, although several problems must be still overcome. On the other hand, new mathematical and computational techniques have been conceived to infer coherent theories and models from huge amounts of experimental data [33].

Systems of Ordinary Differential Equations (ODE) have been largely used for the quantitative modeling of signal transduction networks, but lately some network-oriented and bio-inspired models are overcoming several drawbacks of traditional ODE models. They allow a new insight about biological processes, which cannot be obtained using the "glasses" of classical mathematics [2].

In the next section we focus on a cellular process of great immunological interest, with the goal to design a model for this case study. It is a partially unknown network obtained by long and complex laboratory experiments and proposed very recently in [8].

3 A Case Study

Inflammatory processes in living organisms activate a tissue-specific recruitment of leukocytes, that relies on the complex functional interplay between some surface molecules of leukocytes circulating in the blood and the endothelial cells covering the blood vessel. Leukocyte recruitment into tissues requires extravasation from blood, by a process of transendothelial migration. Three major steps have been identified in the process of leukocyte extravasation: tethering-rolling of free-flowing white bloods cells, leukocyte activation and their arrest by the adherence to endothelial cells. After the arrest *diapedesis* happens, namely leukocytes pass from blood to the tissue beyond endothelial cells [30].

The recruitment process takes place when molecules called *chemokines* are produced by the epithelium and by bacteria that have activated the inflammation. Chemokines bind with *receptors* located on the leukocyte membrane, then activating an internal signaling network. The main output of this network is the

activation of *integrins*, different receptors that interact with endothelial *counter-receptors* slowing down the leukocyte speed, until its arrest. If we call A the initial state, with leukocytes quickly circulating into the blood, B the state of rolling, C the state of activation, and D the state of adhesion, three main phases can be recognized: $A \rightarrow B$ ruled by receptor-receptor interactions, $B \rightarrow C$ ruled by chemokine-receptor interactions, and $C \rightarrow D$ ruled by integrin-receptor interactions. This process was modeled by P systems in [14], where the concept of receptor was integrated with objects transformed and moved through membranes by rewriting systems having rules with priority.

Very recent works have discovered a minimal signaling module activated by chemokines and controlling the integrin activation during the whole process of recruitment of *lymphocytes* B and T, two specific types of leukocyte [8]. Figure 2 shows the entire module (in the center) surrounded by the lymphocyte membrane in which receptors *CXCR4* and integrins *LFA-1* are placed. Elliptical nodes represent types of molecules, *PA* and *PIP2* are second messengers, continuous arrows indicate experimentally demonstrated direct activations (with physical interactions and complex formations), dashed arrows indicate indirect effects, flat line endings indicate inhibitions, empty circle endings indicate second messenger production, the arch ending indicates a direct binding, that is without any enzymatic activation.

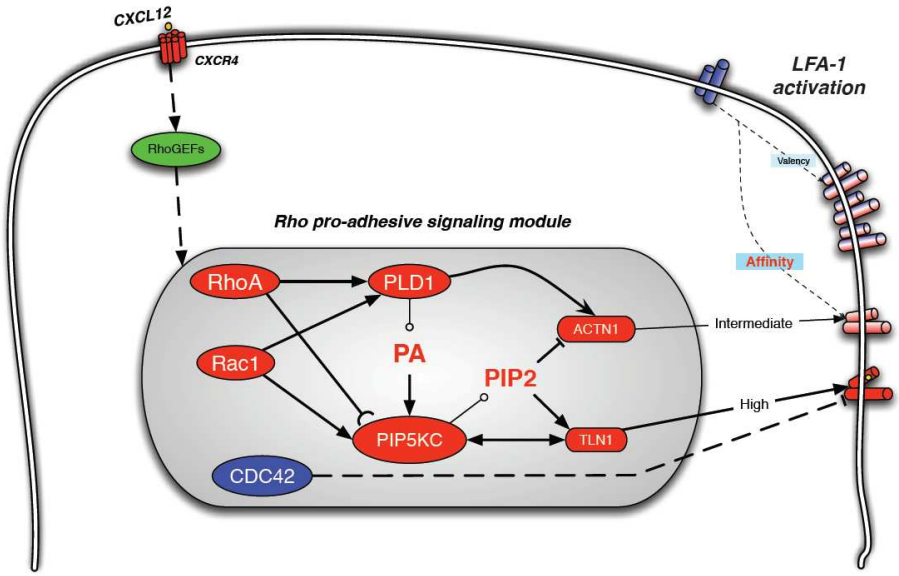


Fig. 2. Model of the Rho-signaling module, activated by chemokines and controlling conformer-specific *LFA-1* affinity triggering during the lymphocyte homing [8].

The *Rho pro-adhesive signaling module* proposed in [8] and reported in Figure 2 takes its input from the receptor *CXCR4*, which is activated by chemokines *CXCL12* and indirectly activates *RhoGEFs* molecules . In turn, RhoGEFs activate the three small GTPase proteins *RhoA*, *Rac1* and *CDC42* inside of the functional module. RhoA and Rac1 activate PLD1, thus leading to phosphatidic acid (*PA*) accumulation. *PIP5KC* bind to RhoA and it may be activated by Rac1 and PA. At the same time, PLD1 may interact with alpha-actinin1 (*ACTN1*), facilitating the interaction between *ACTN1* and the integrins *LFA-1* leading their transition to intermediate affinity state. LFA-1 transitions among low, intermediate and high level are depicted from the top to the bottom (they are all integrins) on the right side of Figure 2. Simultaneously, activated PIP5KC triggers the local accumulation of phosphatidylinositol 1-4-5 phosphate (*PIP2*), which has a central role in the transition of LFA-1 from intermediate to high affinity state, and for the leukocyte firm arrest. The increase of the PIP2 concentration, in fact, may inhibit ACTN1, facilitating its detachment from LFA-1, and may activate Talin1 (*TLN1*), driving the final transition to the high affinity state. We also notice that PIP5KC may activate directly TLN1, and this may promote direct transition of LFA-1 to the high affinity state. In this complex context, *CDC42* plays a "negative" regulatory role by preventing LFA-1 activation.

The signaling network reported in Figure 2 is a qualitative representation of the integrin LFA-1 activation during the lymphocyte homing. An interesting open problem from the biomedical viewpoint is the discovery of qualitative parameters which rule the dynamics of the network over time, such as activation speeds or delays. The development of a qualitative model would allow notable predictions of system behaviors in presence of normal or abnormal (e.g., pathological and pharmacological) conditions. However, qualitative models could imply quite a few experimental problems, because only some quantities are measurable over time, while most of them cannot be measured by current technologies. Our final goal is the discovery of regulative mechanisms of the Rho pro-adesive signaling module, namely starting from the curve of LFA-1 affinity in lymphocytes stimulated by chemokines, showed in Figure 3.

4 Signaling Networks Modeled by MP Systems

As we hinted in the previous sections, the main difficulties of signaling modeling, compared to metabolic modeling, are (i) the lack of stoichiometric coefficients that rule the ratios of chemical interactions, (ii) the consequent non-applicability of some chemical laws, such as the mass conservation law, the Avogadro principle and the Dalton principle, that constrain metabolic models, (iii) the lack of data acquisition techniques to measure activation and concentration levels over time. These difficulties could make inappropriate several methods yet employed for modeling metabolic pathways. New meaningful models must be identified to predict signaling network behaviors, hopefully despite the current lack of information about the underlying phenomena.

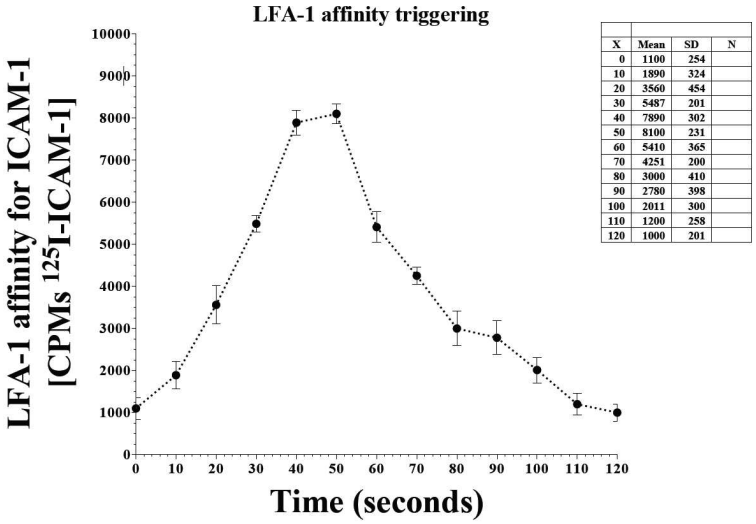


Fig. 3. LFA-1 affinity of a cluster of lymphocytes stimulated by chemokines.

In the following we propose a representation of the network of Figure 2 by means of MP systems, since it is a network-oriented quantitative model which has several features in common with the graph-based qualitative models described in Section 2. *Petri nets* have been also employed to "graphically" model signaling networks from a qualitative point of view [25], in fact their equivalence with MP systems has been proved in [9].

MP systems are deterministic P systems developed to model the dynamics of biological phenomena related to metabolism. They naturally extend graph-based models because they consist of (i) a set of **substances** X, each one associated to (concentration or activation) quantity when observed, (ii)a set of **reactions** R, that move substances, (iii) a set of **parameters** V (such as pressure, temperature,...) each equipped with an evolution function, and (iv)a set Φ of **flux regulation maps**, which state the amount of substances consumed/produced by every reaction in any system transition. The dynamics δ of this model is represented by the evolution of substances and parameters in every temporal interval τ starting from the initial state σ_0.

A graphical representation of MP systems by means of *MP graphs* has been introduced in [23]. MP graphs depict biochemical reactions as bipartite graphs with two levels, in which the first level describes the *stoichiometry* of reactions, while the second level expresses the *regulation* which tunes the flux of every reaction (i.e. the quantity of chemicals transformed at each step) depending on the state of the system (see for example Figure 4).

Recent works aim at deducing MP models from suitable macroscopic observations of given metabolic behaviors along a certain number of steps [21, 19]. In order to assist biologists in the simulation of MP systems we implemented a Java

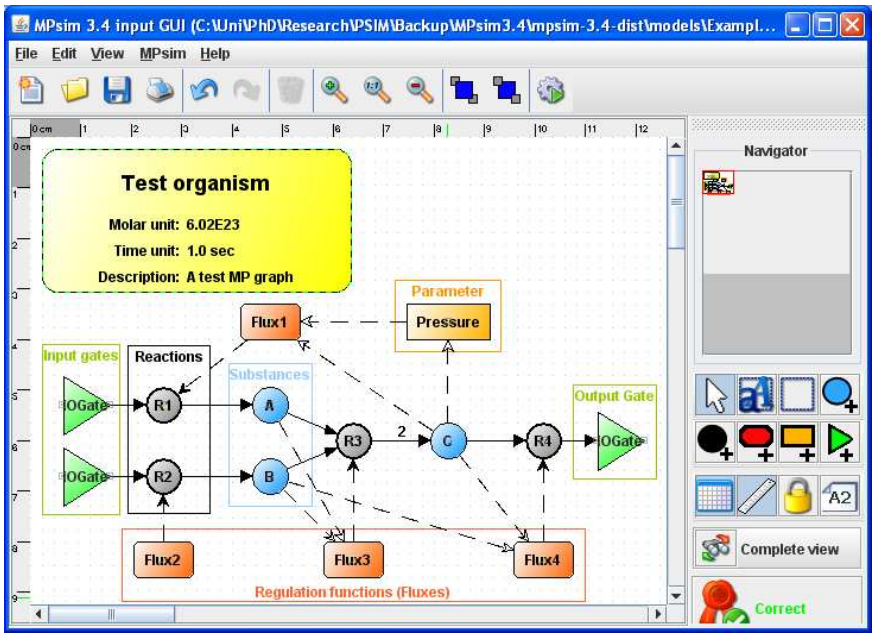


Fig. 4. An example of MP graph visualized by the MPsim 3 graphical user interface. Frame labels point out MP system elements in the MP graph representation.

tool called *MPsim* [4]. The current release of the software employs a friendly user interface to define MP models by means of MP graphs [23], and it produces the plotting of curves given by the system dynamics. The last developments of the software involve a new plugin framework for solving parameter estimation, analysis, visualization and importation tasks in order to increase the system capabilities.

Figure 5 shows an MP graph representation of the signal transduction network depicted in Figure 2. For every chemical involved in the network we set a *substance node* (ellipses) and a linked *reaction node* (circles), which update the substance quantity at each step. We model the quantities associated to PA and PIP2 as concentrations, and the quantities associated to the other substances as activation levels. The *activation level* of a protein expresses its capability to activate other chemicals in the signaling cascade, therefore it is a very important feature for qualitative modeling. Instead, second messengers such as PA and PIP2 tune their interaction with other chemicals depending on their concentration.

Every reaction node is linked to a *flux node* (squares) which computes the quantity added or removed by the reaction at each step, depending on the system state. Dashed arcs from a substance node x to a flux node φ_y (which regulates the substance y) indicates that x is a "regulator" of y. For instance, the activation arc from *RhoA* to *PLD1* in Figure 2 is mapped to the "regulation" arc from the substance node *RhoA* to the flux node *F6*, because the updating of *PLD1*

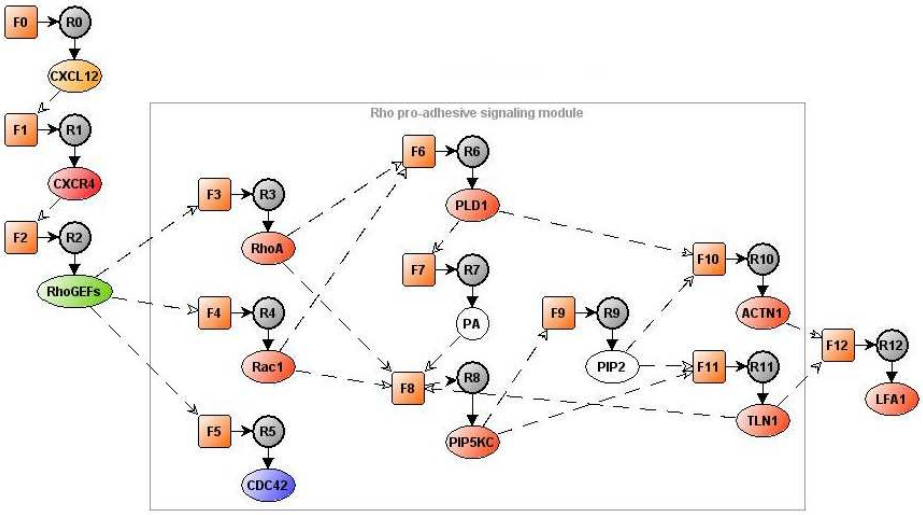


Fig. 5. An MP graph representation of Rho-signaling module activated by chemokines and controlling conformer-specific *LFA-1* affinity.

activation level over time depends on the *RhoA* activation level. All the arcs of Figure 2 have been mapped to MP graph dashed arcs in this way.

The mapping of a qualitative model to an MP graph seems quite a simple task though a crucial problem keeps open: the definition of significant activation speeds and delays which suitably fit the observed dynamics of the system (Figure 3). These parameters affect the regulation functions Φ of the MP system and they would allow good predictions of the system behaviors.

As a future research, we intend to attack this problem by bio-inspired methodologies, like neural networks, and evolutionary techniques, like genetic programming, already proposed by John R. Koza in [16] for the automatic synthesis of metabolic pathways and genetic networks. The latter computational technique could suggest solutions even for the topology validation of qualitative models. Indeed, activation arcs of qualitative models (as that in Figure 2) are drawn when an experiment proves the interaction between two molecules. However, the topology we have of a network is possibly not complete, because some interactions have not been discovered yet. In these cases quantitative computational models, which are based on wide data observations, can point out interaction lacks and suggest new experiments to discover different networks.

References

1. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular biology of the cell.* Garland Science, New York, 4th edition, 2002.

2. F. Bernardini and V. Manca. Dynamical aspects of P systems. *BioSystems*, 70:85–93, 2003.
3. L. Bianco. *Membrane models of biological systems.* PhD thesis, University of Verona, 2007.
4. L. Bianco and A. Castellini. Psim: a computational platform for Metabolic P systems. In G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, and A. Salomaa, editors, *8th International Workshop on Membrane Computing, WMC 2007, Thessaloniki, Greece, June 25-28, 2007, Revised, Selected and Invited Papers*, volume 4860 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2007.
5. L. Bianco, F. Fontana, G. Franco, and V. Manca. P systems for biological dynamics. In Ciobanu et al. [10], pages 81–126.
6. L. Bianco, F. Fontana, and V. Manca. P systems with reaction maps. *International Journal of Foundations of Computer Science*, 17(1):27–48, 2006.
7. L. Bianco, D. Pescini, P. Siepmann, N. Krasnogor, F. J. Romero-Campero, and M. Gheorghe. Towards a P systems pseudomonas quorum sensing model. In Hoogeboom et al. [15], pages 197–214.
8. M. Bolomini-Vittori, A. Montresor, C. Giagulli, D. Staunton, B. Rossi, M. Martinello, G. Constantin, and C. Laudanna. Conformer-specific LFA-1 activation and turnover regulation by chemokine-triggered rho signaling module in human lymphocytes. *Nature Immunology*, 2008. Submitted.
9. A. Castellini, G. Franco, and V. Manca. Toward a representation of Hybrid Functional Petri Nets by MP systems. In Y. Suzuki, G. Păun, A. Adamatzky, M. Hagiyra, and H. Umeo, editors, *Natural Computing, 2nd International Workshop on Natural Computing, IWNC 2007, Nagoya University, Japan, December 10-12, 2007*, pages 1–12, 2007.
10. G. Ciobanu, G. Păun, and M. J. Pérez-Jiménez, editors. *Applications of Membrane Computing.* Natural Computing Series. Springer, 2006.
11. F. Fontana and V. Manca. Discrete solutions to differential equations by metabolic P systems. *Theoretical Computer Science*, 372(2-3):165–182, 2007.
12. G. Franco, P. H. Guzzi, V. Manca, and T. Mazza. Mitotic oscillators as MP graphs. In Hoogeboom et al. [15], pages 382–394.
13. G. Franco, N. Jonoska, B. Osborn, and A. Plaas. Knee joint injury and repair modeled by membrane systems. *Biosystems*, 91(3), March.
14. G. Franco and V. Manca. A membrane system for the leukocyte selective recruitment. In *4th International Workshop on Membrane Computing, WMC 2003, Tarragona, Spain, July 17-22, 2003, Revised Papers*, 2004.
15. H. J. Hoogeboom, G. Păun, G. Rozenberg, and A. Salomaa, editors. *Membrane Computing, 7th International Workshop on Membrane Computing, WMC 2006, Leiden, Netherlands, July 17-21, 2006, Revised, Selected, and Invited Papers*, volume 4361 of *Lecture Notes in Computer Science.* Springer, 2006.
16. J.R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV : Routine Human-Competitive Machine Intelligence (Genetic Programming).* Springer, 2003.
17. G. Krauss. *Biochemistry of signal transduction and regulation.* Wiley-VCH, 2nd english edition, 2001.
18. C. Laudanna. Introduzione all'analisi topologica delle reti biologiche. Url: http://www.cbmc.it.
19. V. Manca. Log-gain principles for Metabolic P systems. In *G. Rozenberg Festschrift.* To appear.

20. V. Manca. The Metabolic Algorithm: Principles and applications. *Theoretical Computer Science*. To appear.
21. V. Manca. Discrete simulations of biochemical dynamics. In M. H. Garzon and H. Yan, editors, *13th International Meeting on DNA Computing, DNA13, Memphis, TN, USA, June 4-8, 2007, Revised, Selected Papers*, volume 4848 of *Lecture Notes in Computer Science*, pages 231–235. Springer, 2007.
22. V. Manca. Metabolic P systems for biochemical dynamics. *Progress in Natural Sciences,* Invited Paper, 17(4):384–391, 2007.
23. V. Manca and L. Bianco. Biological networks in metabolic P systems. *BioSystems*, 91(3):489–498, March 2008.
24. V. Manca, L. Bianco, and F. Fontana. Evolutions and oscillations of P systems: Applications to biochemical phenomena. In G. Mauri, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *5th International Workshop, WMC 2004, Milan, Italy, June 14-16, 2004, Revised Selected and Invited Papers*, volume 3365 of *Lecture Notes in Computer Science*, pages 63–84. Springer, 2005.
25. H. Matsuno, Y. Tanaka, H. Aoshima, A. Doi, M. Matsui, and S. Miyano. Biopathways representation and simulation on Hybrid Functional Petri Nets. *In Silico Biology*, 3(3):389–404, 2003.
26. A. Paun, M. J. Pérez-Jiménez, and F. J. Romero-Campero. Modeling signal transduction using P systems. In Hoogeboom et al. [15], pages 100–122.
27. M. J. Pérez-Jiménez and F. J. Romero-Campero. P systems: a new computational modelling tool for systems biology. *Transactions on Computational Systems Biology VI, Lecture Notes in Bioinformatics, 4220*, pages 176–197, 2006.
28. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
29. G. Păun. *Membrane Computing. An Introduction.* Springer, Berlin, Germany, 2002.
30. L. A. Segel and I. R. Cohen (eds). *Design principles for the Immune System and Other Distributed Autonomous Systems.* Oxford University Press, 2001.
31. The Cytoscape Web Site. Url: http://www.cytoscape.org/.
32. Y. Suzuki and H. Tanaka. Modeling p53 signaling pathways by using multiset processing. In Ciobanu et al. [10], pages 203–214.
33. E. Voit, A. R. Neves, and H. Santos. The intricate side of systems biology. *PNAS*, 103(25):9452–9457, June 20 2006.

Sorting Omega Networks Simulated with P Systems: Optimal Data Layouts

Rodica Ceterchi[1], Mario J. Pérez-Jiménez[2], Alexandru Ioan Tomescu[1]

[1] Faculty of Mathematics and Computer Science, University of Bucharest
Academiei 14, RO-010014, Bucharest, Romania
[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mails: rceterchi@gmail.com, marper@us.es, alexandru.tomescu@gmail.com

Summary. The paper introduces some sorting networks and their simulation with P systems, in which each processor/membrane can hold more than one piece of data, and perform operations on them internally. Several data layouts are discussed in this context, and an optimal one is proposed, together with its implementation as a P system with dynamic communication graphs.

1 Introduction

Paper [9] proposed two models to sort a sequence of N numbers, based on the bitonic sorting network. The first one consisted of N membranes, each storing two numbers; one number was an element of the sequence, and the other one was an auxiliary register used to route values. A number x was codified as the number of appearances of a symbol a in each membrane. Moreover, the membranes were disposed on a 2D-mesh, where only communication between neighbor membranes on the mesh are permitted. This model, using a variant of P Systems, called P systems with dynamic communication graphs, (see [8]), follows closely the implementation of the bitonic sort on the 2D-mesh.

The second model consisted of only one membrane, where all the N numbers were encoded as occurrences of N different symbols. Restrictions on communication were no longer imposed, as if the underlying communication graph were the complete graph.

In this paper we introduce a model in between the two. First of all, observe that the first model has the advantage of a codifying alphabet of fixed size, while the second has the advantage of a small communication overhead. The model we put forth in this paper captures these two benefits. Each membrane holds a fixed number of values, and each of the membranes can communicate with any other.

Additionally, in order to minimize the communication between membranes, we use a periodic remap of values to membranes, according to the steps of the omega network.

The problem of mapping values to processors has been previously addressed in the context of parallel sorting algorithms. The bitonic sorting network, which can sort N keys in time $O(\log^2 N)$, is probably one of the most well-known parallel sorting algorithms. However, modern architectures differ greatly from the theoretical models under which such good results were obtained. As coarse-grained processors can store internally more than one value, the following problem arises: how to map N keys to P processors ($N > P$), such that inter-processor communication is minimized. In the bitonic sorting algorithm, and for $N \geq P^2$, the solution given in [13, 14] consisted in alternating a blocked layout with a cyclic layout, performing thus the minimal number of remaps. This paper gives an optimal mapping strategy for the bitonic sort for any $N > P$, and then applies this result to P Systems.

The paper is organized as follows. Section 2 presents preliminaries on bitonic sorting networks and defines omega networks. Section 3 approaches the problem of mapping N keys among P processors, each processor manipulating $n = N/P$ keys, such that overall communication is minimized. Optimal data layouts for the omega network are proposed along the lines of [20], and some essential results are proved about them. Section 4 discusses about internal processing in one processor, and how we model it in our implementation with P systems. Section 5 introduces the P system which simulates the omega network with optimal data layouts, and the algorithms which generate the sequence of dynamic communication graphs of this model. Complexity issues are addressed at the end of Sections 3 and 5.

2 Preliminaries on Bitonic Sorting Networks and Omega Networks

A bitonic sequence is a concatenation of two monotonic sequences, one ascending, and the other one descending, or a sequence such that a cyclic shift of its elements would put them in such a form.

The key components of a bitonic network are the bitonic splitters and the bitonic mergers. The splitter of size N takes as input a bitonic sequence of length N and partitions it in two bitonic sequences of equal length, such that all the elements in the first sequence are smaller than (or greater than) all the elements in the second sequence. A bitonic merger of size N consists of a splitter of size N and of two mergers of size $N/2$, of opposite direction. It accepts as input a bitonic sequence and sorts it in ascending or descending order (direction).

As any sequence of two numbers is bitonic, the sorting network uses bitonic mergers of increasing size and alternating direction to construct bitonic sequences of increasing length. The last such merger, of size N, renders the whole sequence of N numbers sorted.

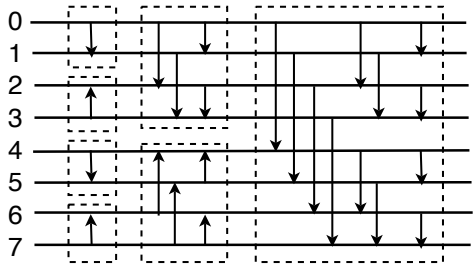
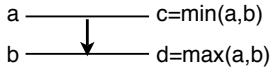
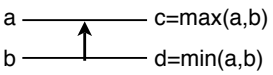


Fig. 1. A bitonic sorting network of size $N = 8$. The network can be partitioned in three stages, each containing bitonic mergers of size 2, 4, and 8, respectively.



(a) Increasing comparator



(b) Decreasing comparator

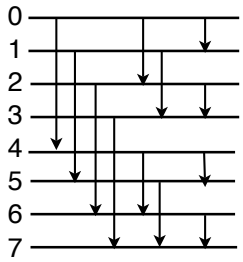
Fig. 2. Network devices

Following [15] it is customary to represent a network as an ordered set of N *lines* (wires) connected by a set of compare-exchange devices (*comparators*, for brevity). A comparator has two input terminals, a and b, and produces two output terminals c and d. If the comparator is increasing, Fig. 2(a), then $c = \min(a, b)$ and $d = \max(a, b)$, while if the comparator is decreasing, Fig. 2(b), $c = \max(a, b)$ and $d = \min(a, b)$. A bitonic sorting network for $N = 8$ is represented in Fig. 1.

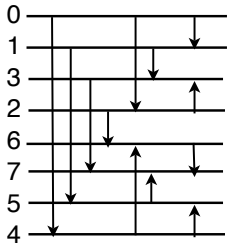
We introduce some more notations regarding the serial and parallel connections of networks T_1 and T_2, of size N. Their serial connection, T_1T_2, is a network in which the i-th output terminal of T_1 is connected to the i-th input terminal of T_2. The parallel connection, $T_1 \circ T_2$, is the union of T_1 and T_2, with terminal i of T_1 becoming terminal i of $T_1 \circ T_2$, and terminal i of T_2 becoming terminal $i + N$ of $T_1 \circ T_2$ $(i = 0, \ldots, N - 1)$.

Definition 1 (Omega network, Fig. 3(d)). *Let D_k, $k \geq 1$ be a one-step network of $N = 2^k$ lines with a device between the pair of lines $(i, i + N/2)$, for $i = 0 \ldots N/2 - 1$. Then the omega network OM_k is recursively defined as $OM_k = D_k(OM_{k-1} \circ OM_{k-1})$.*

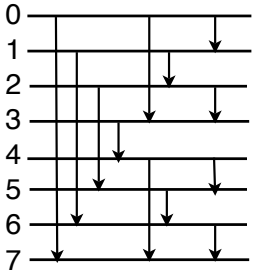
In [6] the striking similarity between the bitonic merger (Fig. 3(a), 3(b)) and the balanced merger (Fig. 3(c)) is investigated. Although prior research [11] showed that there is no permutation of lines to transform the bitonic merger into a balanced merger, a framework is developed under which it is shown that the two



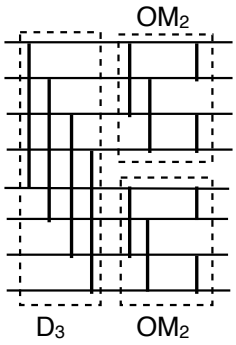
(a) The bitonic merger - classical representation



(b) The bitonic merger - after a permutation of lines



(c) The balanced merger



(d) The omega network OM_3

Fig. 3. The bitonic merger, the balanced merger, and the omega network of size 8

mergers are isomorphic graphs, also isomorphic to the graph of the omega network (Fig. 3(d)).

As a serial connection of $\log N$ identical networks in the class of omega networks forms a sorting network [6], in what follows we will concentrate mainly on the omega network.

3 How They Communicate

A sorting network is a fine-grained theoretical model, containing exactly one input key on each wire. Additionally, comparators require communication between wires,

which can sometimes be more time consuming than the comparison operation itself [1, 3, 10, 16]. When redesigning parallel sorting algorithms for coarse-grained PRAM, one has to pay particular attention to both communication and computation.

Given N keys and P processors ($N > P$), we have to map $n = N/P$ keys to each processor, such that overall communication is minimized. Ionescu and Schauser [13, 14] investigated this problem for the bitonic sorting algorithm. As initially suggested in [10], they proposed a smart periodical switch between a blocked layout and a cyclic layout. They observed that in each stage of the sorting algorithm, the last $\log n$ steps can be performed locally under a blocked layout, while under the cyclic layout the first $\log n$ steps are local. A necessary condition for the two layouts to span enough depth to cover an entire stage of the network is $N \geq P^2$. In addition, the two layouts are particular to the sorting network being implemented. We shall see, for example, that the balanced merger [11, 12], which, as the bitonic merger, belongs to the class of omega networks, also admits data layouts optimizing overall communication.

An approach from the opposite side was put forth by Lee and Batcher [17]. They used a parity strategy for a shared-memory model with $N = 2P$ to store even-parity keys in local memory, while only odd-parity keys were recirculated. This decreased by a factor of 2 the number of shared memory references.

The main contribution of this paper is a general scheme to map N values to P processors, for any $N > P$ and for any sorting network with the topology of the omega network. Our idea captures the essence from the alternating smart layout of [14], and makes it generally applicable, even when $N < P^2$. The number of data layouts is no longer two, but it depends on the granularity of the processors.

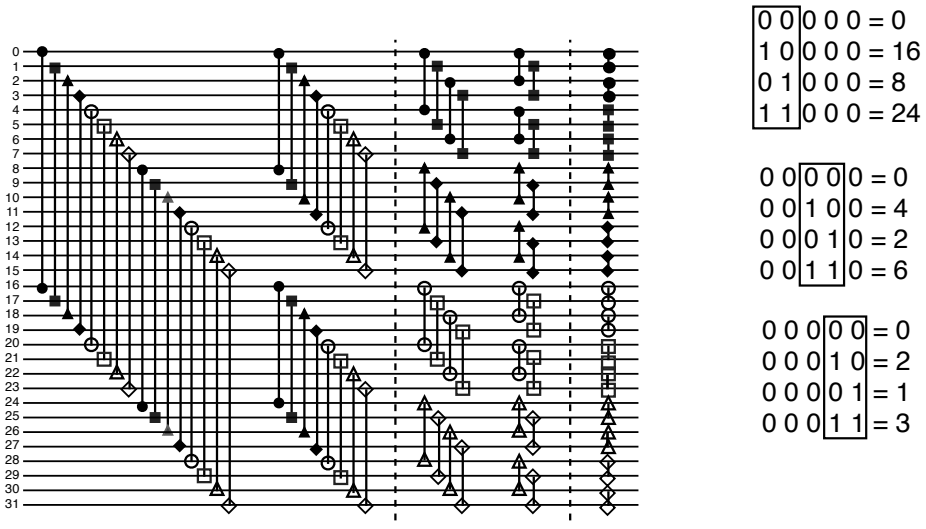
3.1 Optimal Data Layouts for the Omega Network

In the following, without explicitly mentioning it, we assume we have to sort $N = 2^k$ keys using P processors, $N > P$, each processor holding $n = N/P$ keys. Any number $i \in \{0, \ldots, 2^k - 1\}$ has a bit representation $i = a_1 a_2 \cdots a_k$, a_1 being the most significant bit, and a_k the least significant one. To simplify notation, we say that a sequence of bits $a_j \cdots a_i$, where $i, j \in \{1, \ldots, k\}$ and $j > i$, stands for the void sequence. The number of parallel steps of OM_k is k, and step t of the omega network OM_k contains devices linking lines whose bit representations differ of bit t, with $1 \leq t \leq k$. For any $t \in \{1, \ldots, k\}$, consider the function $bc_t : \{0, 1, \ldots, 2^k - 1\} \longrightarrow \{0, 1, \ldots, 2^k - 1\}$, the bit complement of the t-th bit, defined by $bc_t(a_1 a_2 \cdots a_t \cdots a_k) = a_1 a_2 \cdots \bar{a}_t \cdots a_k$. The function bc_t is injective and idempotent.

First, we give a formal definition of a data layout.

Definition 2 (Data layout). *A data layout of N values to P processors is a function $\mathcal{D} : \{0, \ldots, N-1\} \to \{0, \ldots, P-1\}$.*

We introduce the following data layouts, as suggested in [10, 14].



(a) An omega network on size 32. Lines marked with same shape are assigned to the same processor in one data layout.

(b) Keys mapped to processor 0 in each of the three data layouts

Fig. 4. Three data layouts for the omega network OM_5.

Definition 3 (Blocked layout). *A blocked layout for mapping N keys on P processors is a function $\mathcal{D}^b : \{0, \ldots, N-1\} \to \{0, \ldots, P-1\}$, such that $\mathcal{D}^b(i) = \lfloor i/n \rfloor$, where $n = N/P$.*

Definition 4 (Cyclic layout). *A cyclic layout for mapping N keys on P processors is a function $\mathcal{D}^c : \{0, \ldots, N-1\} \to \{0, \ldots, P-1\}$, such that $\mathcal{D}^c(i) = i \bmod P$.*

We note that Definition 5 in [13], and the definition for the cyclic layout indicated in Section 2.1 of [14] are incorrect, since if we map the i-th key to the i mod n processor, where $n = N/P$, we have that $n \leq P$, which implies $N \leq P^2$, which clearly is not the case considered.

In a blocked layout, the first $\log N - \log n$ steps require remote communication, while the last $\log n$ steps are local. In a cyclic layout, the situation is reversed: the first $\log N - \log n$ steps are local, while the last $\log n$ steps are remote. The idea

proposed in [14] when mapping $N \geq P^2$ values in the bitonic sort is to periodically switch between the two layouts, such that all steps are local. Moreover, as the stages in a bitonic sort have increasing size, the author proposes an improved "smart" remap such that a layout spans through multiple stages of the algorithm, achieving a total of $\log P + 1$ remaps.

Our paper better highlights the reasoning behind these remaps, in the case of the bitonic sort. Consider the omega network OM_k, and consider we choose to map key 0 to processor 0. If each processor can hold 2^m values, which other keys are mapped to processor 0? As we can see, at step 1 we have a device linking line 0 with line $0 + 2^{k-1}$. At step 2 we have a device linking line 0 with line 2^{k-2}, and a device linking line 2^{k-1} and line $2^{k-1} + 2^{k-2}$. We also note that in step 1 lines 2^{k-2} and $2^{k-1} + 2^{k-2}$ were also linked with a device. We continue until step m, where we identify 2^m lines linked by 2^{m-1} devices. It would be natural to map these lines to processor 0, as all comparisons at step m are local. However, one more problem remains: all comparisons at stages 0 through $m-1$ are also local? As we shall see, the answer is yes.

The following lemma is straightforward from the definition of OM_k.

Lemma 1. *At each step $1 \leq t \leq k$ of OM_k, and for any $0 \leq i < 2^k$, line i is linked by a device only with line $bc_t(i)$.*

Lemma 2. *In OM_k, for any $0 \leq i < 2^{k-m}$, $1 \leq m \leq k$ and $0 \leq t \leq k-m$, in steps $t+1, \ldots, t+m$ there is no device linking lines in the set $P_i^{t,m} = \{a_1 a_2 \cdots a_k \mid a_1 \cdots a_t a_{t+m+1} \cdots a_k = i$, where $a_1 \cdots a_k$ is a bit representation$\}$ with lines from $\{0, \ldots, 2^k - 1\} \setminus P_i^{t,m}$.*

Proof. Suppose there are $1 \leq r \leq m$, $l \in P_i^{t,m}$ and $l' \notin P_i^{t,m}$ such that at step $t + r$ there is a device linking l and l'. From Lemma 1 we have that $l' = bc_{t+r}(l)$, which implies $l' \in P_i^{t,m}$, a contradiction.

We can therefore derive the data layouts for the omega network. Suppose we have $N = 2^k$, $n = 2^m$, and $P = 2^{k-m}$. We first assign to each processor P_i all values in the set $P_i^{0,m}$, for $0 \leq i \leq P - 1$. By Lemma 2 we have that the first $\log n = m$ steps are entirely local. After m steps, we remap to each processor P_i all the values in the set $P_i^{m,m}$, and perform the next m stages locally, and so on. We can now give the definition of our proposed data layout.

Definition 5. *Given $N = 2^k$ keys and $P = 2^{k-m}$ processors, which can store $n = 2^m$ values, $m \geq 1$, the sequence of optimal data layouts consists of $\lceil \log N / \log n \rceil = \lceil k/m \rceil$ data layouts. In each data layout $\mathcal{D}_s$, $0 \leq s \leq \lceil k/m \rceil - 1$, values in the set $P_i^{sm,m}$ are mapped to processor P_i, for all $0 \leq i \leq 2^{k-m}$. More formally, for any $0 \leq u < 2^k$ such that $u \in P_i^{sm,m}$, we have $\mathcal{D}_s(u) = i$.*

The following is a consequence of Lemma 1 of [14].

Lemma 3. *The maximum number of successive steps of the omega network that can be executed locally, under any data layout is $\log n$, where $n = N/P$.*

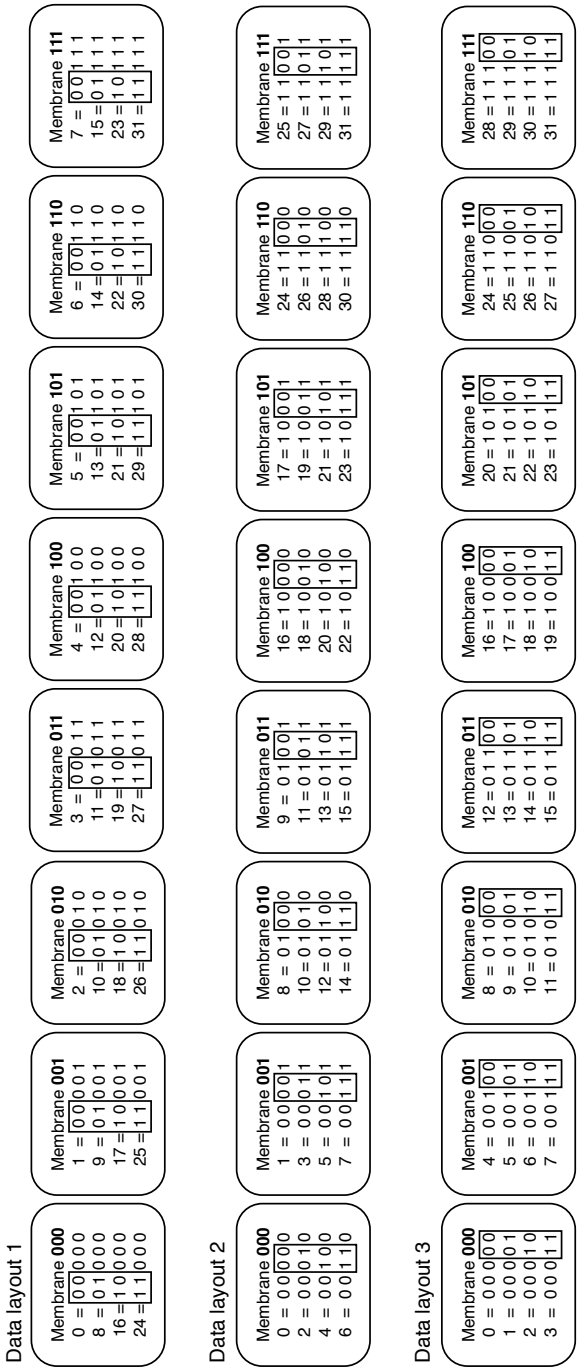


Fig. 5. The three data layouts for the omega network in Figure 4(a).

In each data layout $\mathcal{D}_s$, $0 \leq s \leq \lceil k/m \rceil - 2$, $\log n = m$ steps are local. For $s = \lceil k/m \rceil - 1$, the last $k \bmod m$ steps of the network are local. From Lemma 3 we have that the proposed data layouts for the omega network are optimal.

In the case $N \geq P^2$, we notice that $2m > k$, hence two data layouts are enough to cover the whole omega network. However, they do not coincide with $\mathcal{D}^b$ or $\mathcal{D}^c$, as in the blocked layout, the last m stages are local, while in the cyclic layout, the first $k - m$ stages are local.

3.2 Computation Complexity

In each data layout, a processor holds n values and performs $\log n$ steps locally, taking time $O(n \log n)$. As we have $\lceil \log N / \log n \rceil$ data layouts, we get an overall time complexity of the omega network of $O(n \log N)$. From [6] we have that a serial connection of $\log N$ omega networks of size N is enough to sort a sequence of N numbers. Hence, the complexity to sort N numbers using P processors, each holding $n = N/P$ values, using our proposed data layouts, is $O(n \log^2 N)$.

This remark has a quite profound significance. In the fine-grained theoretical model we have $n = 1$, and its complexity is $O(\log^2 N)$. The complexity of the network using a more coarse-grained model depends linearly on the degree of parallelism of the model. At the opposite end, when $n = N$ and the entire sorting network is simulated locally, we have a complexity of $O(N \log^2 N)$, which is worse than $O(N \log N)$, the complexity of most sequential sorting algorithms. It would be desirable to choose n such that this bound is not surpassed in the parallel model. We impose $n \log^2 N \leq N \log N$, which implies $n \leq N / \log N$.

An algorithm to find the minimum of a bitonic sequence of size n in time $O(\log n)$, was introduced in [14]. This gives a time complexity of each data layout of $O(n)$. In the case of a network obtained from a serial connection of bitonic mergers, this observation gives an overall time complexity of $O(\frac{n}{\log n} \log^2 N)$.

4 What Happens Inside One Processor/Membrane

One processor (and the membrane which simulates it) will be capable of holding $n = N/P = 2^m$, pieces of data. We label the data with indices in the set $\{0, 1, \cdots, n-1\}$. For any such index we consider its writing as a binary string of length m, for instance $i = x_1 x_2 \cdots x_t \cdots x_m$.

Inside one processor, several comparisons are performed, in parallel, between the n pieces of data, in the following manner: for every bit t, (starting with 1, the most significant bit, and ending with m) we compare and exchange if necessary (to obtain an increasing order) all pairs of values codified with a_i and $a_{bc_t(i)}$. More precisely, we have the following algorithm to be performed inside each processor/membrane:

for $t \leftarrow 1$ **to** m **do**
  forall $i < bc_t(i)$ **in parallel do**
    compare$(a_i, a_{bc_t(i)})$;

Algorithm 1: A parallel algorithm for the bitonic merger

where by **compare**(a_i, a_j) we denote sorting in an ascendant manner the values codified by a_i and a_j, i.e. we end by having the minimum of the two values codified by a_i and the maximum by a_j.

The procedure **compare**(a_i, a_j) works in a membrane in the following manner: let s_i, s_j and t_i, t_j be four auxiliary symbols, for the sources and the targets of a comparator. The set of rules

$$\{a_k \to s_k \mid k = i, j\} \cup \{s_i s_j \to t_i t_j, s_i \to t_j s_j \to t_j\} \cup \{t_k \to a_k \mid k = i, j\}$$

implement an increasing comparator between values codified by a_i and a_j. We first rewrite the as to ss, next we have the comparator which writes the minimum to t_i and the maximum to t_j, and then we rewrite these back to a_i and a_j respectively.

For all the comparisons which are to be done in parallel, take auxiliary alphabets $S = \{s_0, \cdots, s_{n-1}\}$ and $T = \{t_0, \cdots, t_{n-1}\}$. We rewrite all initial symbols to symbols in S:

$$\{a_i \to s_i \mid i = 0, 1, \cdots, n-1\}.$$

Next we put the comparators between appropriate pairs:

$$\{s_i s_j \to t_i t_j, s_i \to t_j, s_j \to t_j \mid i = 0, 1, \cdots, n-1, i < j = bc_t(i)\}.$$

Then we rewrite back to the original alphabet:

$$\{t_i \to a_i \mid i = 0, 1, \cdots, n-1\}.$$

The parallel comparisons at each step t

forall $i < bc_t(i)$ **in parallel do**
  compare$(a_i, a_{bc_t(i)})$;

will thus be simulated in a membrane P by the rules

$$\{a_i \to s_i \mid i = 0, 1, \cdots, n-1\} \cup$$
$$\cup \{s_i s_j \to t_i t_j, s_i \to t_j, s_j \to t_j \mid i = 0, 1, \cdots, n-1, i < j = bc_t(i)\} \cup$$
$$\cup \{t_i \to a_i \mid i = 0, 1, \cdots, n-1\}.$$

5 A P System which Simulates the Omega Network

In this section we introduce a P system with dynamic communication [7], along the same general lines as the model proposed in [8, 9]. For each of the processors

$\mathcal{P}_i$, $i \in \{0, 1, \ldots, P-1\}$ we have an associated membrane, which we label i. The graphs we consider are sub-graphs of the complete graph, K_P, or of the identity graph.

Note that at a certain step of the sorting algorithm not all edges are involved in communication. Therefore we call *active sub-graphs* of K_P those graphs containing only such edges. We also introduce the *identity* graph, with

$$V(Id) = \{0, 1, \ldots, P-1\},$$

$$E(Id) = \{(i, i) \mid 0 \leq i \leq P-1\}$$

for modeling internal processing steps.

In order to describe the evolution of such a P system, we use pairs of the type $[graph, rules]$. We have $graph$ a sub-graph of K_P or Id and $rules$ a mapping from the set of all edges of $graph$, $E(graph)$, to the set of all symbol/object rewriting rules for routing or comparison operations.

The formal definition of the P system is

$$\begin{aligned} \Pi = (V = \{a_0, \ldots, a_{n-1}\} \cup \mathcal{A}, \langle [a_0^{x_0^0}, a_1^{x_1^0}, \ldots, a_{n-1}^{x_{n-1}^0}]_0, \ldots, \\ [a_0^{x_0^{P-1}}, a_1^{x_1^{P-1}}, \ldots, a_{n-1}^{x_{n-1}^{P-1}}]_{P-1} \rangle, R_\mu), \end{aligned}$$

where the membrane indices are $\{0, 1, \ldots, P-1\}$. The alphabet $\{a_0, \ldots, a_{n-1}\}$ is of fixed size, and the set $\mathcal{A}$ contains the auxiliary symbols necessary to simulate the omega network, as indicated in Section 4. Numbers x_i^j with $0 \leq i \leq n-1$ are the values stored on the wires mapped to processor j, $0 \leq j \leq P-1$ in the first data layout. Each of them is codified as the number of occurrences of a symbol a_i inside membrane j. Finally, R_μ is the finite sequence of pairs $[graph, rules]$ which guides the computation.

We will see in the sequel that R_μ is generated algorithmically, by concatenating sequences of pairs $[graph, rules]$[3].

Lemma 4. *Given $N = 2^k$ keys and $P = 2^{k-m}$ membranes, which can store $n = 2^m$ values, $m \geq 1$, after the computation for the data layout $\mathcal{D}_s$ is finished, symbol a_i of membrane j codifies the value corresponding to wire $u \in \{0, \ldots, N-1\}$, where the bit representation of u is $u = j_1 \ldots j_{sm} i_1 \ldots i_m j_{sm+1} \ldots j_{k-m}$. By $j_1 \ldots j_{k-m}$ and by $i_1 \ldots i_m$ we denoted the bit representations of j, and i, respectively.*

Proof. The proof is immediate by Definitions 1, 5 and Lemma 2.

We observe that the remap of values from a data layout to the other can be done in $P+1$ steps. When passing from data layout $\mathcal{D}_{s-1}$ to $\mathcal{D}_s$, with $0 < s \leq \lceil k/m \rceil - 1$, in each step j, $0 \leq j \leq P-1$, membrane j sends its contents along the edges of the communication graph C_s^j. To avoid collisions in the destination membranes,

[3] We denote the empty sequence by λ, and the concatenation of two sequences by ".".

it also performs a rewriting of symbols from a_t to a'_t, for all $t \in \{0, \ldots, n-1\}$. In the last step $P+1$, all auxiliary symbols a'_t will be rewritten back to a_t in all membranes, and the local computation can begin in each membrane.

We give below two algorithms generating the communication graphs C_s^j, and the rules associated to each edge.

$E(C_s^j) \leftarrow \emptyset$;
for $j \leftarrow 0$ **to** $P-1$ **do**
  for $i \leftarrow 0$ **to** $n-1$ **do**
    let j have bit representation $j_1 \cdots j_{sm} j_{sm+1} \cdots j_{k-m}$;
    let i have bit representation $i_1 \cdots i_m$;
    `// the destination membrane of value encoded by` a_i `in`
    `membrane` $j\ z \leftarrow j_1 \cdots j_{sm} i_1 \cdots i_m j_{(s+1)m+1} \cdots j_{k-m}$;
    `// the destination symbol of value encoded by` a_i `in`
    `membrane` $j\ t \leftarrow j_{sm+1} \cdots j_{sm+m}$;
    $E(C_s^j) := E(C_s^j) \cup \{j, z\}$;
    $\text{rules}_{C_s^j}((j,z)) := a_i \rightarrow a'_t$;

Algorithm 2: Generation of the sequence of P communication graphs when passing from data layout $\mathcal{D}_{s-1}$ to $\mathcal{D}_s$, with $0 < s \leq \lceil k/m \rceil - 1$.

for $j \leftarrow 0$ **to** $P-1$ **do**
  $\text{rules-endcomm}((j,j)) := \{a'_i \rightarrow a_i \mid 0 \leq i \leq n-1\}$;

Algorithm 3: Generation of the rules associated to the identity graph which rewrite back the auxiliary symbols a'_t when passing from any data layout $\mathcal{D}_{s-1}$ to $\mathcal{D}_s$, with $0 < s \leq \lceil k/m \rceil - 1$.

We assume that the sequence denoted by $SimOM$ is the sequence of pairs $[graph, rules]$ which simulates the omega network of size n, OM_m ($n = 2^m$). Its construction was indicated in Section 4 and is expressed algorithmically below.

$SimOM \leftarrow \lambda$;
for $t \leftarrow 1$ **to** $m = \log n$ **do**
  forall $p \leftarrow 0$ **to** $P-1$ **in parallel do**
    $\text{rules}_{t,1}((p,p)) \leftarrow \{a_i \rightarrow s_i \mid i = 0, 1, \ldots n-1\}$;
    $\text{rules}_{t,2}((p,p)) \leftarrow \{s_i s_j \rightarrow t_i t_j, s_i \rightarrow t_j, s_j \rightarrow t_j \mid i = 0, 1, \ldots, n-1, i < j = bc_t(i)\}$;
    $\text{rules}_{t,3}((p,p)) \leftarrow \{t_i \rightarrow a_i \mid i = 0, 1, \cdots n-1\}$;
  $SimOM \leftarrow SimOM \cdot [Id, \text{rules}_{t,1}] \cdot [Id, \text{rules}_{t,2}] \cdot [Id, \text{rules}_{t,3}]$;

Algorithm 4: Generation of the sequence $SimOM$ which simulates the omega network of size n.

We can now give the algorithm which generates the whole sequence R_μ guiding the computation.

$R_\mu \leftarrow \lambda$;
for $s \leftarrow 1$ **to** $\lceil k/m \rceil - 1$ **do**
  $R_\mu \leftarrow R_\mu \cdot SimOM$;
  for $j \leftarrow 0$ **to** $P - 1$ **do**
    $R_\mu \leftarrow R_\mu \cdot [C_s^j, \text{rules}_{C_s^j}]$;
  $R_\mu \leftarrow R_\mu \cdot [Id, \text{rules-endcomm}]$;
$R_\mu \leftarrow R_\mu \cdot SimOM$;

Algorithm 5: Generation of the sequence R_μ which guides the computation.

5.1 Computation complexity

Observe that the length of the sequence $SimOM$ is $3 \log n$. As we have $\frac{\log N}{\log n}$ data layouts, and that in each data layout $3 \log n$ steps are needed for $SimOM$ and another $P + 1$ steps are needed for communication, the length of R_μ is $3 \log N + \frac{N \log N}{n \log n}$. A sorting network can be obtained by a serial connection of $\log N$ omega networks, hence our model can sort in time $O(\log^2 N + \frac{N \log^2 N}{n \log n})$. Note that when $n = N$ all computation is local, and the complexity is the best possible, $O(\log^2 N)$. When $n = 2$ the complexity increases to $O(N \log^2 N)$.

References

1. A. Aggarwal, A.K. Chandra, M. Snir, "Communication Complexity of PRAMs", *Theoretical Computer Science,* vol. 71, no.1, pp. 3 - 28, Mar. 1990.
2. M. Ajtai, J. Komlos, and E. Szemeredi, "An $O(N \log N)$ Sorting Network", *Proc. 15th Ann. ACM Symp. Theory of Computing,* pp. 1-9, May 1983.
3. A. Alexandrov, M. Ionescu, K.E. Schauser, C. Scheiman, "LogGP: Incorporating Long Messages into the LogP model", *Journal of parallel and distributed computing,* vol. 44, no. 1, pp. 71-79, 1997.
4. A. Alhazov, D. Sburlan, "Static Sorting P Systems", Chapter 8 in *Applications of Membrane Computing*, (G. Ciobanu, Gh. Păun, M.J. Pérez Jiménez Eds.), Springer, 2005.
5. K.E. Batcher, "Sorting networks and their applications", *Proc. AFIPS Spring Joint Comput. Conf.,* vol. 32, pp. 307-314, Apr. 1968.
6. G. Bilardi, "Merging and Sorting Networks with the Topology of the Omega Network", *IEEE Transactions on Computers,* vol. 38, no. 10, pp. 1396-1403, Oct. 1989.
7. R. Ceterchi, C. Martín-Vide, "Dynamic P Systems", *LNCS*, vol. 2597, pp. 146 - 186, 2003.
8. R. Ceterchi, M.J. Pérez Jiménez, "On two-dimensional mesh networks and their simulation with P systems", *LNCS*, vol. 3365, pp. 259-277, 2005.
9. R. Ceterchi, M.J. Pérez Jiménez, A.I. Tomescu, "Simulating the Bitonic Sort Using P Systems", *G. Eleftherakis et al. (Eds.): WMC8 2007, LNCS,* vol. 4860, pp. 172-192, 2007.

10. D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation", *Proc. Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming,* pp.1-12, May 1993.
11. M. Dowd, Y. Perl, M. Saks, L. Rudolph, "The balanced sorting network", *Proc. Second annual ACM symp. on Principles of distributed computing,* pp. 161-172, 1983.
12. M. Dowd, Y. Perl, M. Saks, L. Rudolph, "The periodic balanced sorting network", *JACM,* vol. 36. no. 4, pp. 738-757, 1989.
13. M.F. Ionescu, "Optimizing Parallel Bitonic Sort", *Tech. Report TRCS96-14,* Dept. of Comp. Sci., Univ. of California, Santa Barbara, July 1996.
14. M.F. Ionescu, K.E. Schauser, "Optimizing parallel bitonic sort", *Proc. 11th Int'l Parallel Processing Symp.,* pp. 303-309, 1997.
15. D.E. Knuth, *The art of computer programming*, volume 3: sorting and searching, second ed. Redwood City, CA: Addison Wesley Longman, 1998.
16. C. Kruskal, L. Rudolph, M. Snir. "A complexity theory of efficient parallel algorithms", *Theoretical Computer Science,* vol.71, no.1, pp. 95 - 132, Mar. 1990.
17. J.D. Lee, K.E. Batcher, "Minimizing Communication in the Bitonic Sort", *IEEE Trans. on Parallel and Distributed Systems,* vol. 11, no. 5, pp. 459-474, May 2000.
18. F. Leighton, "Tight Bounds on the Complexity of Parallel Sorting," *IEEE Trans. Computers,* vol. 34, no. 4, pp. 344-354, Apr. 1985.
19. M.S. Paterson, "Improved Sorting Networks with $O(\log N)$ Depth," *Algorithmica,* vol. 5, pp. 75-92, 1990.
20. A.I. Tomescu, "Optimal Data Layouts for Omega Networks", *manuscript.*

Spiking Neural P Systems – A Natural Model for Sorting Networks

Rodica Ceterchi, Alexandru Ioan Tomescu

Faculty of Mathematics and Computer Science, University of Bucharest
Academiei 14, RO-010014, Bucharest, Romania
E-mails: rceterchi@gmail.com, alexandru.tomescu@gmail.com

Summary. This paper proposes two simulations of sorting networks with spiking neural P systems. A comparison between different models is also made.

1 Introduction

Sorting is one of the most studied problem in Computer Science, as it has a wide range of applications, including sequential and parallel algorithms. Over the last decades, it has been investigated under parallel architectures, as utilizing many functional units to sort concurrently can improve performance. Batcher introduced the bitonic sorting network and the odd-even sorting network in [5], which can sort N keys in $O(\log^2 N)$ time, and with $O(N \log^2 N)$ comparators. Various improvements over these networks have been proposed in [2, 17, 18], which provide better bounds for depth or number of comparators.

Spiking Neural (SN) P systems were introduced in [10]. They simulate the behavior of neurons sending signals through axons, consisting of membranes which contain a number of occurrences of only one symbol, and release them through connections towards other membranes.

In the paper [11] an application of SN P systems for sorting N numbers has been proposed. We introduce in this paper a different approach, by first constructing SN P systems which act as comparators, and next by assembling these building blocks according to the topology of a sorting network.

Sorting has been modeled or simulated with a variety of P systems. In this paper we introduce first a model which uses a SN P system comparator (of two values), and next another model based on an n-comparator. Section 2 presents preliminaries on sorting networks. Section 3 introduces the SN P systems used as ascending/descending comparators, and shows how to connect them by classical sorting networks in order to obtain sorting SN P systems. Section 4 presents yet another model, an n-comparator generalization. This question is related to optimal data layouts for networks of processors capable of holding more than one piece of

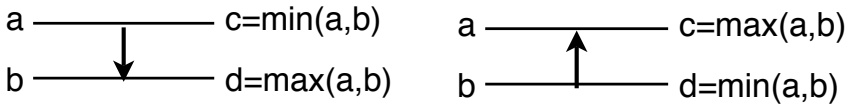
data. Finally, in Section 5 a comparison is made between the three models, the one introduced in [11], and the two other ones presented in this paper.

2 Preliminaries on Sorting Networks

A bitonic sequence is a concatenation of two monotnic sequences, one ascending, and the other one descending, or a sequence such that a cyclic shift of its elements would put them in such a form.

The key components of a bitonic network are the bitonic splitters and the bitonic mergers. The splitter of size N takes as input a bitonic sequence of length N and partitions it in two bitonic sequences of equal length, such that all the elements in the first sequence are smaller than (or greater than) all the elements in the second sequence. A bitonic merger of size N consists of a splitter of size N and of two mergers of size $N/2$, of opposite direction. It accepts as input a bitonic sequence and sorts it in ascending or descending order (direction).

As any sequence of two numbers is bitonic, the sorting network uses bitonic mergers of increasing size and alternating direction to construct bitonic sequences of increasing length. The last such merger, of size N, renders the whole sequence of N numbers sorted.



(a) Increasing comparator (b) Decreasing comparator

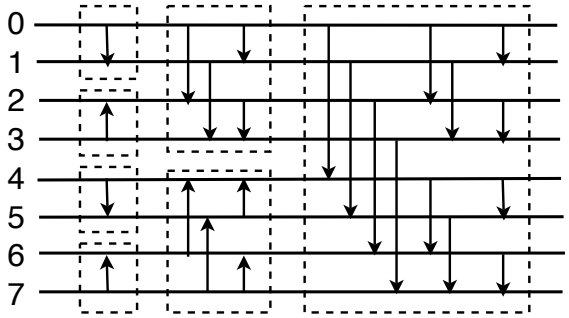
Fig. 1. Network devices

Following [14] it is customary to represent a network as an ordered set of N *lines* (wires) connected by a set of compare-exchange devices (*comparators*, for brevity). A comparator has two input terminals, a and b, and produces two output terminals c and d. If the comparator is increasing, Fig. 1(a), then $c = \min(a, b)$ and $d = \max(a, b)$, while if the comparator is decreasing, Fig. 1(b), $c = \max(a, b)$ and $d = \min(a, b)$. A bitonic sorting network for $N = 8$ is represented in Fig. 2(a).

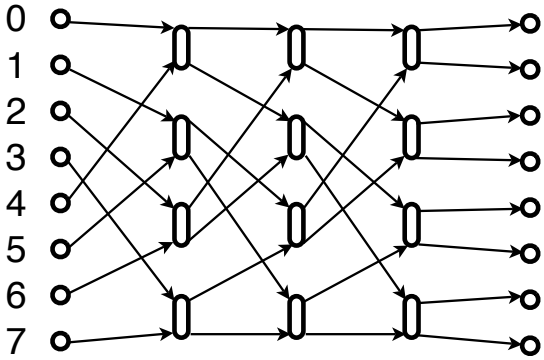
A network can also be represented as a directed acyclic graph [8].

Definition 1 (Network). *A network T of size N is a directed acyclic graph such that:*

1. *there are N nodes, called input terminals, with in-degree 0 and out-degree 1, labeled from 0 to $N - 1$;*



(a) A bitonic sorting network of size $N = 8$. The network can be partitioned in three stages, each containing bitonic mergers of size 2, 4, and 8, respectively.



(b) The bitonic merger for $N = 8$ represented as a graph.

Fig. 2. The bitonic sorting network and the bitonic merger of size 8.

2. *there are N nodes, called output terminals, with in-degree 1 and out-degree 0, labeled from 0 to $N - 1$;*
3. *all the remaining nodes u, representing comparators, have in-degree and out-degree 2.*

In Fig. 2(b) is represented the bitonic merger under the above formalism.

We define the depth of a node u of network T, $d(u)$, as the length of the longest path in T from an input node to u. The depth of network T, $d(T)$, is the maximum

depth of a node of in-degree and out-degree 2 in T. Any network can be viewed as a series of steps, each containing at most $N/2$ parallel devices. Each step t contains the nodes of T at depth t.

The arcs of a network can be partitioned in N arc-disjoint paths, each joining an input node to an output node. Such a partition yields a *line-representation* of T, as in [14].

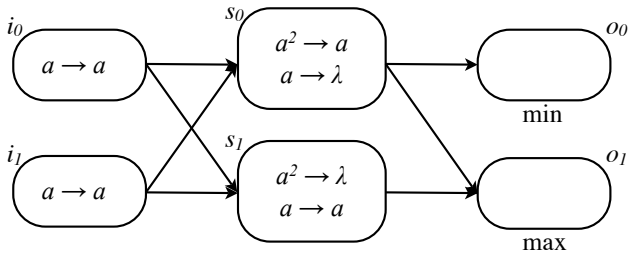
3 Spiking Neural P Systems for Sorting Networks

We note that the above representation is a theoretical model which indicates the comparisons between input values. However, in the context of SN P systems, this model has a straightforward implementation. Each wire is now represented by a synapse between two neurons, and each value x travels between two neurons as x spikes, one spike per time unit. Comparators are implemented by a set of neurons which send the minimum and the maximum (as number of spikes) through designated synapses. Once these two ingredients are at hand, we proceed to construct a SN P system in the same way the original sorting network was constructed.

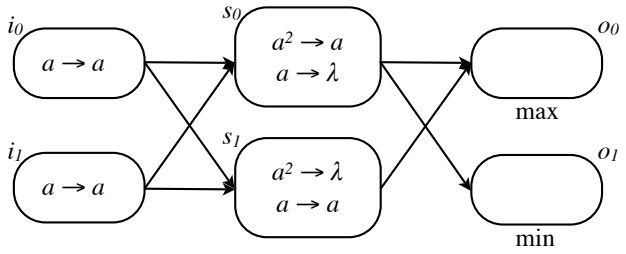
Ionescu and Sburlan [11] introduced a SN P system which sorts N numbers, and consisted of 3 layers of N neurons each. The first layer was made up of input neurons which in the initial configuration contained the input values codified as numbers of spikes. At each time unit these neurons sent one spike each to the second layer. This layer decanted the spikes to the third layer, where the output neurons were located. After a number of steps equal to the maximum value of the N numbers, the ith output neuron received the ith smallest value, codified as number of spikes, sorting thus in ascending order. In a way, the idea of the algorithm is the same as that of bead sort [4].

In this section we are concerned only with comparators of two elements, hence with SN P systems which sort two numbers (called for brevity SN P comparators). In Fig. 3(a) we give an ascending comparator, and in Fig. 3(b) we give a descending comparator. The SN P ascending comparator functions in the following way: the first layer of neurons (labelled with i) initially contains the values to be compared, codified as number of spikes. At each step they instantaneously send one spike to both s_0 and s_1. As long as both s_0 and s_1 receive spikes, only s_0 sends one spike to o_0 and o_1. After one input neuron has consumed its spikes, the minimum is obtained in o_0. There will be only one input neuron to send spikes to s_0 and s_1. In this case, s_0 forgets its spikes, and s_1 sends them to o_1, where the maximum is obtained.

Consider the SN P system modeling an ascending comparator and the numbers x and y to be sorted. In order to be able to use these SN P systems as building blocks of a bitonic sorting network, we assume that instead of loading the numbers x and y as spikes in i_0 and i_1 in the initial configuration, they are fed one by one to these input neurons by another neuron.



(a) Increasing comparator



(b) Decreasing comparator

Fig. 3. SN P systems modeling comparators

Lemma 1 (Composition lemma). *Suppose that in each time unit from t_0 until $t_0+(x-1)$ neuron i_0 receives one spike and that in rest it does not receive any spike. Analogously, suppose that in each time unit from t_0 to $t_0+(y-1)$ neuron i_1 receives one spike, and that in rest it does not receive any spike. Then neuron o_0 does not receive any spike, except for time moments from t_0+2 until $t_0+2+(\min(x,y)-1)$, when it receives one spike at each moment. Analogously, neuron o_1 does not receive any spike, except for time moments from t_0+2 until $t_0+2+(\max(x,y)-1)$, when it receives one spike at each moment.*

Proof. Consider the time moments t, with $t_0 \leq t \leq t_0 + (\min(x,y) - 1)$. Both neurons i_0 and i_1 receive spikes and in turn send them through the synapses (by the rule $a \to a$). s_0 and s_1 receive two spikes each, neuron s_0 sends one spike to o_0 and o_1 (by the rule $a^2 \to a$), while neuron s_1 forgets them (by the rule $a^2 \to \lambda$). Therefore at time moment $t + 2$ neurons o_0 and o_1 receive one spike each. From time moment $t_0 + \min(x,y)$ onward, only one neuron of i_0 and i_1 sends spikes, hence the configuration of the synapses of s_0 and s_1 prevent o_0 from receiving other spikes. The first part of the claim is proved.

At each time moment t, with $t_0 + \min(x, y) \leq t \leq t_0 + (\max(x, y) - 1)$, neuron o_1 receives one spike at moment $t + 2$. After time moment $t_0 + \max(x, y)$ there are no other spikes entering in system, hence from time moment $t_0 + 2 + \max(x, y)$ onward there will be no other spikes entering neuron o_1.

A similar lemma is valid in the case of a SN P decreasing comparator.

Assume that we are given a network T as a graph, and that we have a line-representation of it (i.e. a set of N arc-disjoint path linking input terminals with output terminals). Hence, we extend Definition 1, by labeling edges, apart from input and output terminals. For every path that begins with input terminal labeled i, we label all its edges with i. More formally, we have the following definition.

Definition 2 (Edge labeling). *Given a graph T as in Definition 1 representing a sorting network, and a line-representation of T, we attach to each edge $e \in E(T)$ that belongs to a path in the line representation of T beginning with i, label $l(e) = l(i)$ (supposing that i is labeled with $l(i)$).*

For example, in Figure 5 we have a labeled bitonic merger.

A SN P system modeling a sorting network given as a graph is obtained in the following way. For each input terminal node i we have a corresponding input neuron i_i. For each comparator (ascending / descending) we have the s- and o-neurons of a SN P comparator (ascending / descending). For each edge of the graph between two comparators we have synapses between corresponding SN P comparators. The output terminal nodes are the o-neurons of the last SN P comparators. Additionally, we add to all o-neurons, except the output ones, the rule $a \to a$.

More formally, we construct and label the SN P system in the following recursive way.

i) for each input terminal node i we have a corresponding input neuron $i_i = i_{i,1}$, $0 \leq i \leq n - 1$;
ii) for each comparator at depth $1 \leq k \leq d(T)$ with incident edges labeled with i and j, $i < j$, we add the s- and o-neurons of a SN P comparator, connected in the previously specified way. With the notations in Figure 3, let s_0 and s_1, and o_0 and o_1 be the s-, and o-neurons, respectively, just added. We add synapses between the following pairs of neurons: $(i_{i,k}, s_0)$, $(i_{i,k}, s_1)$, $(i_{j,k}, s_0)$, $(i_{j,k}, s_1)$. Additionally, if $k < d(T)$, we label o_0 with $o_{i,k} = i_{i,k+1}$, and o_1 with $o_{j,k} = i_{j,k+1}$; else we label o_0 with $o_{i,k} = o_i$, and o_1 with $o_{j,k} = o_j$.

As an example, Figure 4 depicts a SN P system which models the bitonic merger of size $N = 8$.

Theorem 1. *For any SN P comparator at depth k corresponding to a comparator with incident edges $i < j$ which carry values x and y, respectively, we have that*

1a) in each time moment from $2(k-1)$ until $2(k-1) + x$ neuron $i_{i,k}$ receives one spike;

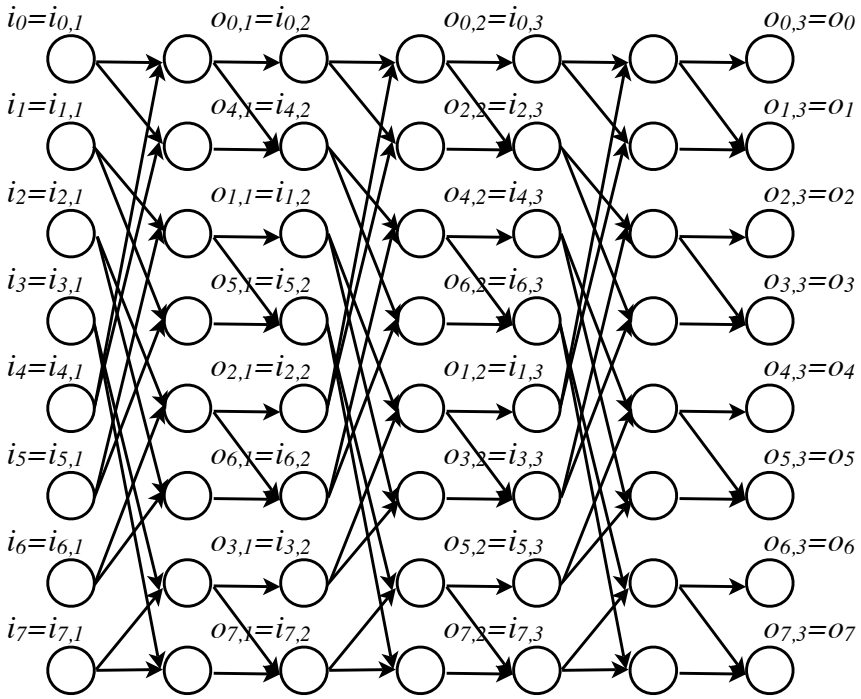


Fig. 4. A SN P system modeling the bitonic merger of size $N = 8$.

1b) in each time moment from $2(k-1)$ *until* $2(k-1)+y$ *neuron* $i_{j,k}$ *receives one spike.*

In case of an ascending comparator,

2a) in each time moment from $2k$ *until* $2k + \min(x, y)$ *neuron* $o_{i,k}$ *receives one spike;*

2b) in each time moment from $2k$ *until* $2k + \max(x, y)$ *neuron* $o_{j,k}$ *receives one spike.*

In case of a descending comparator,

3a) in each time moment from $2k$ *until* $2k + \max(x, y)$ *neuron* $o_{i,k}$ *receives one spike;*

3b) in each time moment from $2k$ *until* $2k + \min(x, y)$ *neuron* $o_{j,k}$ *receives one spike.*

Proof. We prove the claim by induction on k. When $k = 1$ we are at time moment $t = 0$. We have explained previously that the behaviour of the system when the spikes are loaded initially in the input neurons is identical to when they are fed one by one to these neurons. Claims 2 and 3 are true from Lemma 1 and $t_0 = 0$.

We now suppose that the claim is true for k, with $1 \leq k < \log N$, and prove it for $k+1$. From claims $2b$ and $3b$ of the induction hypothesis, we know that $o_{i,k} = i_{i,k+1}$ receives one spike from $2k$ until $2k+u$, where u is the value carried by wire i before the comparator at depth $k+1$. Analogously, $o_{j,k} = i_{j,k+1}$ receives one spike from $2k$ until $2k+v$, where v is the value carried by wire t before the comparator at depth $k+1$. This proves claims $1a$ and $1b$. If we take $t_0 = 2k$, $x = u$, and $y = v$ in Lemma 1, we have that claims 2 and 3 are true.

Corolary 1 *Given a network T of size N, if we replace the comparator nodes by the appropriate SN P systems sub-networks, the result is still a sorting network.*

The sorting network obtained with SN P systems works differently than the initial one. At time moment $2d(T) + \min\{x_0, \ldots, x_{N-1}\}$ all output neurons contain the value $\min\{x_0, \ldots, x_{N-1}\}$ as number of spikes a. Let us denote with $min_1 = \min(\{x_0, \ldots, x_{N-1}\} \setminus \{\min(x_0, \ldots, x_{N-1})\})$. Then at time moment $2d(T) + min_1$ all output neurons $o_1, \ldots, o_{N-1}$ contain min_1 spikes and o_0 remains with $\min\{x_0, \ldots, x_{N-1}\}$. Finally, at $2d(T) + \max\{x_0, \ldots, x_{N-1}\}$ we have in o_{N-1} the value $\max\{x_0, \ldots, x_{N-1}\}$, and all other output neurons contain the initial set in ascending order.

4 An n-Comparator Improvement

In the previous section we were concerned with constructing a SN P system which implemented a given sorting network, each comparator having a corresponding SN P systems. We now address the problem of comparators of more than two values, and show how we can transform a network given as in Definition 1 into a generalized one, with n-comparators which can sort n values. The only restriction we make is that the network has comparators on only one direction (ascending or descending). This is not a limiting assumption in our treatment of sorting with spiking neural P systems, as, for example, a bitonic sorting network is a serial and parallel connection of bitonic mergers, which have comparators of the same direction. From [11] we have at hand a SN P system which can sort n values, therefore we show how to assemble these building blocks to get a sorting SN P system which can sort N values.

The idea of the algorithm we propose stems from the following observation. Suppose we have N input terminals, and we can use comparators of at most n values. We try to design the first step of the generalized network, and have that the depth of the furthest comparator of the initial network that can be simulated by one n-comparator is $\log_2 n$. Let this device be u and let i and j be its incident edges. This implies that all prior devices involving lines i, j and all other lines that they were connected with, have to be implemented by the same n-comparator as the one which act on i and j. We call these lines *predecessors* of depth $\log n$ of u and we say that they are *mapped* to the same comparator as i and j. In addition, observe that as the n-comparator sorts the whole sequence of predecessors, then it also implemented correctly the standard comparators.

Definition 3 (Predecessors of a node). *We denote by predecessors of depth m of node u the set $P_m(u) = \{l(xy) \mid xy \in E(T)$ and there exists a path from y to u of length $m-1\}$*

Apart from u, at depth $d(u)$ reside other devices between lines mapped to the same comparator as i and j by the above procedure. These devices have to be simulated by the same comparator holding i and j. We call these devices *neighbours* of depth $\log n$ of u, and give the following definition.

Definition 4 (Neighbours of a device). *We denote by neighbours of depth m of node u of T having $d(u) \geq 2$, the set $N_m(u) = \{v \in V(T) \mid d(v) \geq 2$ and $P_m(u) \cap P_m(v) \neq \emptyset\}$*

In order to simulate $\log n$ steps locally in one n-comparator, any n-comparator should accommodate all the lines which compare values in these $\log n$ steps. This imposes a limit on the number of neighbours of depth $\log n$ of a device u. More specifically, we have the following property:

Property 1. We say that a network T of size N admits a generalized network with comparators of n values if $|N_{\log n}(u)| \leq n/2$, for any $1 \leq s \leq d(T)/\log n$, and any node $u \in V(T)$ with $d(u) = s \log n$.

Bearing all this in mind, we give an algorithm to construct $\lceil d(T)/\log n \rceil$ mapping functions $\mathcal{D}_s$ which for any wire $i \in \{0, \ldots, N-1\}$ indicates the comparator to which is mapped at step s of the generalized network T'.

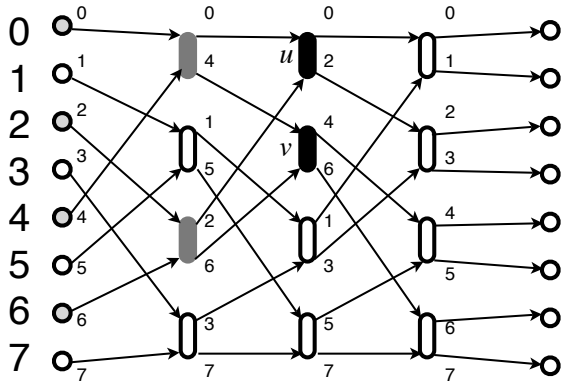


Fig. 5. The bitonic merger of size 8, given as a graph. Edges are labeled as in Definition 2, according to the classical line-representation of the bitonic merger. Neighbouring nodes u and v at depth 2 are shown in black. They have predecessors $\{0, 2, 4, 6\}$. The rest of devices linking these lines are shown in gray.

Input: A network T of size N and a line-representation of it, n the maximum capacity of one comparator. Network T has all comparators of the same direction and satisfies Property 1.

Output: A sequence of functions D_s, $0 \leq s < \lceil d(T)/\log n \rceil$, with domain $\{0, \ldots, N-1\}$ representing a mapping of wires to comparators at stage s of the generalized network T'.

label edges of T as in Definition 2;
forall $0 \leq s < \lfloor d(T)/\log n \rfloor$ **do**
    set counter $p = 0$;
    reset previous markings;
    forall *nodes* $u \in V(T)$ *not marked, at depth* $d(u) = \log n + s \log n$ **do**
        forall $v \in P_{\log n}(u)$ **do**
            $\mathcal{D}_s(v) = p$;
        forall $v \in N_{\log n}(u)$ **do**
            mark node v;
        $p = p + 1$;
`// treatment of the special case when` $d(T)$ `is not divisible by` $\log n$
remaining-depth $\leftarrow d(T)$ *mod* $\log n$;
if *remaining-depth* > 0 **then**
    $s \leftarrow \lfloor d(T)/\log n \rfloor$;
    set counter $p = 0$;
    reset previous markings;
    forall *nodes* $u \in V(T)$ *not marked, at depth* $d(T)$ **do**
        forall $v \in P_{remaining-depth}(u)$ **do**
            $\mathcal{D}_s(v) = p$;
        forall $v \in N_{remaining-depth}(u)$ **do**
            mark node v;
        $p = p + 1$;

Algorithm 1: Deriving the mapping of wires to comparators in the generalized network.

5 Conclusions and Open Problems

This paper has proposed two models (which we call Model 2 and Model 3) of simulating a sorting network with SN P systems and has proved the correctness of the construction. These systems do not have a simple form as the one in [11] (Model 1), so their usefulness remains to be investigated. We consider here a number of measures of the models: number of neurons, number of synapses, total number of rules in all neurons, maximal length of rules, and time complexity.

Model 1 has three layers, with N neurons each. On the other hand, Model 2 has $1+2+\ldots+\log N$ steps, each being implemented by $2N$ neurons. If we also add

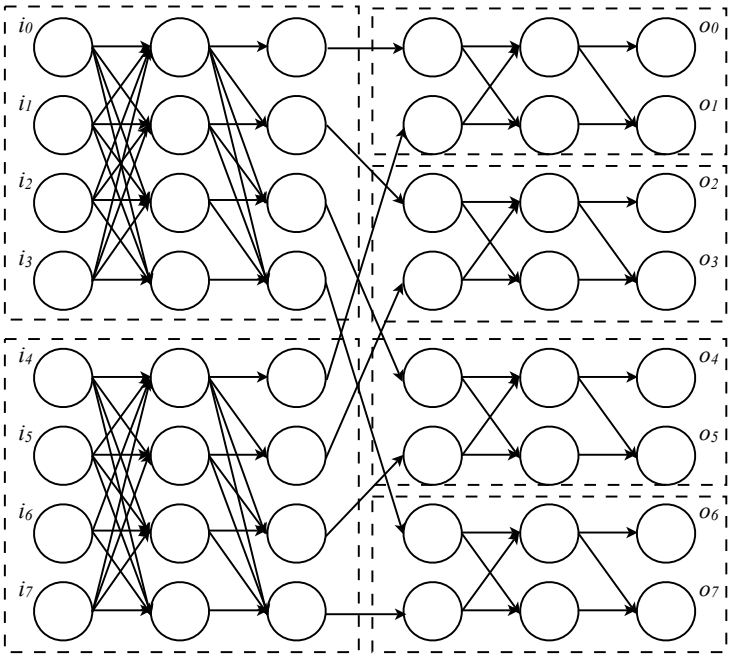


Fig. 6. A SN P system constructed from the generalized network of the bitonic merger of size 8.

the N input neurons, we get a total number of neurons of $N + N \log N(\log N + 1)$. In Model 3 the situation is similar, except that now we have $\frac{\log N(\log N+1)}{2 \log n}$ steps, which give a total number of $N + N\frac{\log N(\log N+1)}{\log n}$ neurons. Even if in the two proposed models this measure has increased by a factor of $\log^2 N$, we will see that concerning other measures, we get a benefit of at least $\frac{\log^2 N}{N}$.

The number of synapses of Model 1 is quadratic in N, as we have synapses between any pair of neurons in the first two layers. The total number of synapses is $\frac{3N^2+N}{2}$. In Model 2, for each step of the bitonic sorting network, we have $2N$ synapses between i-neurons and s-neurons, and $2\frac{N}{2} + \frac{N}{2}$ between s-neurons and o-neurons. In Model 3, at each step of the generalized network we have $\frac{3n^2+n}{2}\frac{N}{n}$ synapses. This gives a total number of synapses in Model 2 of $\frac{7}{2}N\frac{\log N(\log N+1)}{2}$, and in Model 3 of $\frac{3n+1}{2}N\frac{\log N(\log N+1)}{2 \log n}$.

Concerning the total number of rules, in Model 1 we have again a quadratic dependence $N^2 + N$. In Model 2 we have $3N$ rules in each step of the network, hence $3N\frac{\log N(\log N+1)}{2}$ rules in all. The number of rules per step in Model 3 is $\frac{N}{n}(n + n^2)$, which gives a total of $N(n + 1)\frac{\log N(\log N+1)}{2 \log n}$.

As in each time unit only one spike is discharged from the input neurons, then the complexity of the algorithm of [11] is $O(M)$, where M is the maximum of the N numbers. As, in general, we have to sort N distinct numbers, then the maximum of them is N, hence the complexity of the algorithm is $\Omega(N)$. The time complexity of the two proposed models is $O(M + d(T))$, where $d(T)$ is the depth of the network being simulated (i.e. $\log^2 N$, and $\frac{\log^2 N}{\log n}$, respectively). Usually, the maximum number M does not depend on N, so we have $O(M) = O(M + d(T))$.

We also note that now the length of the rules is constant. An overview of these measures are presented in Table 1.

An open problem that remains to be investigated is how to further reduce the number of neurons of a sorting SN P system. We propose for scrutiny the class of periodic sorting networks, which are composed of a sequence of identical blocks. Since only one block needs to have a SN P system implementation, then a periodic sorting network can be realized by recirculating the output of a block back as its input. This results in savings in neurons and synapses. Consider for example the odd-even sorting network of Batcher [5] which is composed of N identical applications of a period of depth 2. This can provide a linear number of neurons in N, with the same time complexity $O(M)$.

However, the main difficulty behind such an approach is the ability to tell when the numbers are sorted. We note that the output neuron holding the minimum has to stop recirculating spikes before the output neuron holding the maximum. The idea of a global clock holding a number of spikes proportional to the number of times the identical blocks have to be applied is not enough.

Table 1. Comparison between the model proposed in [11] (Model 1), the direct simulation of the bitonic sorting network with a SN P system (Model 2), and the simulation of a generalized bitonic sorting network with n-comparators (Model 3). The models sort N numbers, M being the maximum.

Measure	Model 1 [11]	Model 2	Model 3
Number of neurons	$3N$	$N + N \log N(\log N + 1)$	$N + N\frac{\log N(\log N+1)}{\log n}$
Number of synapses	$\frac{3N^2+N}{2}$	$\frac{7}{2}N\frac{\log N(\log N+1)}{2}$	$\frac{3n+1}{2}N\frac{\log N(\log N+1)}{2\log n}$
Number of rules	$N^2 + N$	$3N\frac{\log N(\log N+1)}{2}$	$N(n+1)\frac{\log N(\log N+1)}{2\log n}$
Maximal length of rules	$N + 1$	3	$n + 1$
Time complexity	$O(M)$	$O(M + \log^2 N) = O(M)$	$O(M + \frac{\log^2 N}{\log n}) = O(M)$

References

1. A. Aggarwal, A.K. Chandra, M. Snir, "Communication Complexity of PRAMs", *Theoretical Computer Science,* vol. 71, no.1, pp. 3 - 28, Mar. 1990.

2. M. Ajtai, J. Komlos, and E. Szemeredi, "An $O(N \log N)$ Sorting Network", *Proc. 15th Ann. ACM Symp. Theory of Computing,* pp. 1-9, May 1983.
3. A. Alexandrov, M. Ionescu, K.E. Schauser, C. Scheiman, "LogGP: Incorporating Long Messages into the LogP model", *Journal of parallel and distributed computing,* vol. 44, no. 1, pp. 71-79, 1997.
4. J.J. Arulanandham, "Implementing Bead – Sort with P Systems", *Unconventional Models of Computation 2002 (C.S. Calude, M.J. Dinneen, F. Peper Eds.)*, LNCS vol. 2509, pp. 115-125, 2002.
5. K.E. Batcher, "Sorting networks and their applications", *Proc. AFIPS Spring Joint Comput. Conf.,* vol. 32, pp. 307-314, Apr. 1968.
6. G. Bilardi, "Merging and Sorting Networks with the Topology of the Omega Network", *IEEE Transactions on Computers,* vol. 38, no. 10, pp. 1396-1403, Oct. 1989.
7. D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation", *Proc. Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming,* pp.1-12, May 1993.
8. M. Dowd, Y. Perl, M. Saks, L. Rudolph, "The balanced sorting network", *Proc. Second annual ACM symp. on Principles of distributed computing,* pp. 161-172, 1983.
9. M. Dowd, Y. Perl, M. Saks, L. Rudolph, "The periodic balanced sorting network", *JACM,* vol. 36. no. 4, pp. 738-757, 1989.
10. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
11. M. Ionescu, D. Sburlan, "Some Applications of spiking neural P systems", *Preproceedings of Membrane Computing, International Workshop - WMC8*, Thessaloniki, Greece, pp. 383-394, 2007.
12. M.F. Ionescu, "Optimizing Parallel Bitonic Sort", *Tech. Report TRCS96-14,* Dept. of Comp. Sci., Univ. of California, Santa Barbara, July 1996.
13. M.F. Ionescu, K.E. Schauser, "Optimizing parallel bitonic sort", *Proc. 11th Int'l Parallel Processing Symp.,* pp. 303-309, 1997.
14. D.E. Knuth, *The art of computer programming*, volume 3: sorting and searching, second ed. Redwood City, CA: Addison Wesley Longman, 1998.
15. C. Kruskal, L. Rudolph, M. Snir. "A complexity theory of efcient parallel algorithms", *Theoretical Computer Science,* vol.71, no.1, pp. 95 - 132, Mar. 1990.
16. J.D. Lee, K.E. Batcher, "Minimizing Communication in the Bitonic Sort", *IEEE Trans. on Parallel and Distributed Systems,* vol. 11, no. 5, pp. 459-474, May 2000.
17. F. Leighton, "Tight Bounds on the Complexity of Parallel Sorting," *IEEE Trans. Computers,* vol. 34, no. 4, pp. 344-354, Apr. 1985.
18. M.S. Paterson, "Improved Sorting Networks with $O(\log N)$ Depth," *Algorithmica,* vol. 5, pp. 75-92, 1990.
19. L. Rudolph, "A robust sorting network", *IEEE Trans. Comput.* C-32,4, pp. 326-335, 1985.

Computational Complexity of Simple P Systems

Gabriel Ciobanu, Andreas Resios

"A.I.Cuza" University of Iaşi
Romania
gabriel@info.uaic.ro

Summary. We introduce a new class of membrane systems called simple P systems, and study its computational complexity using the classical theory. We start by presenting the knapsack problem and analyzing its space and time complexities. Then we study the computational complexity of simple P systems by considering the static allocation of resources enabling the parallel application of the rules. We show that the problem of allocating resources for simple P systems is **NP**-complete by reducing it to the knapsack problem. Thus we express the computational complexity of this class of P systems in terms of classical complexity theory.

1 Introduction

We describe the computational complexity of the simple P systems in classical complexity theory, extending the short note presented initially in [5]. Membrane computing is a rather young field of *natural computing* which has been developing very fast in the last decade. It combines the power of distributed parallel rewriting systems with the power and context evolution to achieve computational universality. We use a subclass of transition P systems, simple P systems, where the left side of the rules can contain only a single object with different multiplicity. We use a classical combinatorial **NP**-complete problem, the *knapsack* problem, to show that the static allocation of rules for this class is **NP**-complete.

The structure of this paper is as follows. In Section 2 we present a pseudo-polynomial algorithm for the knapsack problem and study its complexity. Then we give a short presentation of transition P systems, and introduce the simple P systems in Section 3. Then we show that static allocation of the available resources to rules is **NP**-complete. In Section 5 we present a new approach to study the complexity of simple P systems. Conclusion and references end the paper.

2 Knapsack Problem

The knapsack problem, a classical combinatorial optimization problem, refers to finding a maximum total value given a set of objects values and weights, and a weight limit. The discrete version refers to the fact that we can only include an object as a whole, not just a part of it. Mathematically, the discrete knapsack problem can be formulated as follows: given a bag of capacity c and n objects, labelled from $1, \ldots, n$, each having value p_i and weight w_i, maximize $\sum_{i=1}^{n} p_i x_i,\ x_i \in \{0, 1\}$, subject to $\sum_{1}^{n} w_i x_i \leq c$, where $x_i = 1$ means that we take object i. This problem is known to be an **NP**-complete problem. However, there exists a *pseudo-polynomial* time algorithm using dynamic programming with running time of $O(n \cdot c)$. An algorithm is said to run in pseudo-polynomial time if its running time is polynomial in the numeric value of the input (however this can be exponential with respect to the length). Formally, we say that a function f is pseudo-polynomial if $f(n)$ is no greater than a polynomial function of the problem size n and an additional property of the input $k(n)$. Note that pseudo-polynomial time becomes polynomial time if the values are encoded in unary base.

The knapsack problem can be expressed as a optimization problem: as follows:

- Objective function
$$\max \sum_{i=1}^{n} p_i x_i$$
- Restrictions
  - $x_i \in \{0, 1\}$;
  - $\sum_{1}^{n} w_i x_i \leq c$.

To solve this problem using dynamic programming we need to define the notion of a state and the transition between two states. For this problem a state is defined by the number of objects we take into consideration. Thus we start with an initial state where we do not have any object to choose from, and we make transitions to the next state until we reach a final state. A transition is represented by the choice between inserting the object in the bag or not. We denote by $f_i(X)$ the function which answers the question "What is the optimal value obtained by using only i objects and X weight?". Thus a state i is defined by the function f_i. The function f_i can be computed as follows:

$$f_i(X) = \begin{cases} -\infty & , X < 0 \\ 0 & , i = 0 \wedge X \geq 0 \\ \max\{f_{i-1}(X), f_{i-1}(X - w_i) + p_i\} & , \text{otherwise} \end{cases} \tag{1}$$

The answer to the knapsack problem is given by the value of $f_n(c)$. The functions f_i can be stored as a table, and can be computed starting from the initial state to the last state. Note that the current state depends only on the previous state, thus we can store only the last two lines in the table. We use the example given in Table 1 where we consider $c = 10$.

i	1	2	3
w_i	3	5	6
p_i	10	30	20

Table 1. Knapsack instance

X	0	1	2	3	4	5	6	7	8	9	10
f_0	0	0	0	0	0	0	0	0	0	0	0
f_1	0	0	0	10	10	10	10	10	10	10	10
f_2	0	0	0	10	10	30	30	30	40	40	40
f_3	0	0	0	10	10	30	30	30	40	40	40

Table 2. Values for f corresponding to instance defined in Table 1

We obtain Table 2 corresponding to the recurrence defined by f Note that function f_i has many repeating values. To solve the problem of space, we need to store only the different values for f_i. A possible solution is to associate with each different value of f_i a 3-uple $(k, W_{i,k}, T_{i,k})$ with the following meaning: k represents the profit, $W_{i,k}$ represents the sum of the objects weight which we can use to achieve profit k, and $T_{i,k}$ represents the objects which are used to achieve that profit.

By using this approach, we need to keep a list of 3-uples instead of the full table. We now show an example of how we use this list to solve the problem. We start with the list containing only $\{(0, 0, \emptyset)\}$. At each iteration we construct a new list A_i that contains the 3-uples with the valid profits that we could obtain by using the current object. The new list is obtain by merging the previous list with the new constructed list: $L_{i+1} = \mu(L_i, A_i)$, where $\mu(A, B)$ is the merging of two list of 3-uples. For the instance previously presented we have:

$$
\begin{aligned}
L_0 &= \{(0, 0, \emptyset)\} \\
A_0 &= \{(10, 3, \{1\})\} \\
L_1 &= \{(0, 0, \emptyset), (10, 3, \{1\})\} \\
A_1 &= \{(30, 5, \{2\}), (40, 8, \{1, 2\})\} \\
L_2 &= \{(0, 0, \emptyset), (10, 3, \{1\}), (30, 5, \{2\}), (40, 8, \{1, 2\})\} \\
A_2 &= \{(20, 6, \{3\}), (30, 9, \{1, 3\})\}) \\
L_3 &= \{(0, 0, \emptyset), (10, 3, \{1\}), (20, 6, \{3\}), (30, 5, \{2\}), (40, 8, \{1, 2\})\}
\end{aligned}
$$

The solution to the problem is given by the last element of L_3. In our example it is given by $(40, 8, \{1, 2\})$, which means we can obtain a profit of 40 by taking items 1 and 3 that weight 8.

The Algorithm 1 solves the knapsack problem using this approach. We now express the time and space complexity of Algorithm 1. We know that $|L_{i+1}| \leq 2 \cdot |L_i|$ because $|A_i| \leq |L_i|$. The computation of L_{i+1} from L_i and A_i is done in $O(|L_i| + |A_i|) = O(|L_i|)$ time. We know that the 3-uples from L_i satisfy the relation:

Algorithm 1 Knapsack(n,w,p,M)

```
1: L_0 ← {(0, 0, ∅)}
2: for i ← 1 to n do
3:    A_{i−1} ← ∅
4:    for all (k, W_{a,k}, T_{a,k}) in L_{i−1} do
5:       if W_{a,k} + w_i ≤ M then
6:          A_{i−1} ← A_{i−1} ∪ {(k + p_i, W_{a,k} + w_i, T_{a,k} ∪ {i})}
7:       end if
8:    end for
9:    L_i ← Merge(L_{i−1}, A_{i−1})
10: end for
11: return last(L_n)
```

$$0 \leq |L_i| \leq k \leq \sum_{j=1}^{i} p_j \leq n \cdot \max\{p_1, \ldots, p_n\}$$

It follows that $|L_i| \leq n \cdot \max\{p_1, \ldots, p_n\}$. In conclusion Algorithm 1 has the time complexity:

$$O(\sum_{i=1}^{n} |L_i|) = O(n^2 \cdot \max\{p_1, \ldots, p_n\})$$

Note that this complexity is pseudo-polynomial, and if $max\{p_1, \ldots, p_n\} > 2^n$ then this algorithm runs in exponential time w.r.t the size of its input.
The space complexity is:

$$O(\sum_{i=1}^{n} |L_i|) = O(\sum_{i=1}^{n} 2^i) = O(2^n)$$

In the *Merge* procedure presented as Algorithm 2, when we have two items with the same profit we choose the one with the lowest weight. This assures that for each possible profit we have the minimum weight.

3 Simple P Systems

Membrane computing represents an unconventional paradigm of computing which combines the power of distributed parallel rewriting systems with the power and context evolution. Local rules and the evolution contexts are biological metaphors: the rules are developmental rules in cells, and contexts denote division mechanisms of cells and active/mobile membranes. There are several ingredients in membrane systems which are meaningful from the point of view of biological media. The basic model and many variants are described in [7], and some applications are presented in [4]. Membrane computing is used both to model cells or biological systems and to study the computability and complexity of a new and unconventional computing device.

Algorithm 2 Merge(A,B)

```
1: S ← ∅, i ← 1, j ← 1
2: n ← min(|A|, |B|)
3: while i ≤ n && j ≤ n do
4:    (k1, W_{a,k1}, T_{a,k1}) ← A[i], (k2, W_{b,k2}, T_{b,k2}) ← B[j]
5:    if k1 < k2 then
6:       S ← S ∪ {(k1, W_{a,k1}, T_{a,k1})}, i ← i + 1
7:    else if k1 > k2 then
8:       S ← S ∪ {(k2, W_{b,k2}, T_{b,k2})}, j ← j + 1
9:    else if k1 = k2 then
10:      if W_{a,k1} < W_{b,k2} then // chose the one with minimum weight
11:         S ← S ∪ {(k1, W_{a,k1}, T_{a,k1})}
12:      else
13:         S ← S ∪ {(k2, W_{b,k2}, T_{b,k2})}
14:      end if
15:      i ← i + 1, j ← j + 1
16:   end if
17: end while
18: if i ≤ n then
19:   for j ← i to |A| do
20:      S ← S ∪ A[j]
21:   end for
22: else
23:   for i ← j to |B| do
24:      S ← S ∪ B[i]
25:   end for
26: end if
27: return S
```

The transition P system is represented by *regions* delimited by a *membrane structure* that contains *multisets* of objects that evolve according to associated rules. A computation consists of a number of transition between system configurations and the result is represented either by the objects present in the final configuration in a specific membrane or by the objects which leave the outermost membrane of the system (the skin membrane) during the computation.

Definition 1. *A* transition P system *of degree* $n, n \geq 1$*, is a construct*

$$\Pi = (O, T, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)\,,$$

where

1. O *is an alphabet of* objects*;*
2. $T \subseteq O$ *(the output alphabet);*
3. μ *is a membrane structure of degree* n*;*
4. $w_i, 1 \leq i \leq n$*, strings that represent multisets over* V *associated with the regions of* μ*;*

5. *R_i represents the rules from region μ_i of μ; an* evolution rule *is a pair (u, v) written as $u \to v$, where u is a string over O and $v = v'$ or $v = v'\delta$, where v' is a string over $\{a_{here}, a_{out}, a_{in_j} | a \in O, 1 \leq j \leq n\}$, and δ is a special symbol not in O; the length of u is called the* radius *of rule $u \to v$. $R_i, 1 \leq i \leq n$, are finite sets of evolution rules over O.*

Definition 2. *A* simple P system *is a transition P system where the left side of a rule can contain a single object with an arbitrary multiplicity. Formally, a rule $u \to v \in R_i$ has $u = k \cdot a$, where $k \in \mathbb{N}$ and $a \in O$.*

The *membrane structure* is a tree structure, where each node represents a membrane. The relation between a child node and a parent node symbolizes that the parent membrane contains the child membrane. To define the *membrane structure* we consider the language MS over the alphabet $\{[,]\}$ recursively defined as follows:

1. $[\,] \in MS$;
2. if $\mu_1, \ldots, \mu_n \in MS$, $n \geq 1$ then $[\mu_1 \ldots \mu_n] \in MS$;
3. nothing else is in MS.

We define a binary relation $\sim$ over the elements of MS: $x \sim y$ if and only if we can write $x = \mu_1\mu_2\mu_3\mu_4$, $y = \mu_1\mu_3\mu_2\mu_4$, for $\mu_1\mu_4 \in MS$ and $\mu_2, \mu_3 \in MS$. We denote by $\sim$ the reflexive and transitive closure of $\sim$ (note that $\sim$ is an equivalence relation). We denote by $\overline{MS}$ the set of equivalence classes of MS with respect to $\sim$. A *membrane structure* is an element of $\overline{MS}$, where each pair of matching parentheses $[,]$ is a membrane. The degree of a membrane structure is defined as the number of membranes it contains. A natural way to represent membrane structure is by a Venn diagram because this emphasizes the topological structure between computing compartments.

A configuration of the system is given by the membrane structure and the contents of each region. The initial configuration is a $(n+1)$-tuple $(\mu, w_1, \ldots, w_n)$. Having the possibility to dissolve a membrane, we can obtain a configuration which has only some of the initial membranes. Thus we define a configuration of Π as any sequence $(\mu', w'_{i_1}, \ldots, w'_{i_k})$, with μ' a membrane structure obtained by dissolving from μ all membranes different from $i_1, \ldots, i_k$, with w_{i_j} strings over O, $1 \leq j \leq k$, and $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$.

For two configurations $C_1 = (\mu', w'_{i_1}, \ldots, w'_{i_k})$, $C_2 = (\mu'', w''_{j_1}, \ldots, w''_{j_k})$ of Π we write $C_1 \Rightarrow C_2$, and we say that we have a *transition* from C_1 to C_2 if we can pass from C_1 to C_2 by using the evolution rules from $R_{i_1}, \ldots, R_{i_k}$. A sequence of transitions between configurations of a given P system, Π is called a *computation* with respect to Π. A computation *halts* if there is no rule applicable to the objects from the last configuration. The result of a computation is $\Psi_T(w)$, where w describes the multiset of output objects sent out by the system during the computation. $\Psi_T(w)$ is the Parikh mapping associated with T; it is defined by $\Psi_T(w) = (|w|_{a_1}, \ldots, |w|_{a_n})$, where $T = \{a_1, \ldots, a_n\}$, $w \in T^*$, and $|w|_{a_i}$ denotes the number of occurrences of a_i in w. The set of such vectors $\Psi_T(w)$ is denoted by $Ps(\Pi)$, and we say it is generated by Π.

4 Complexity of the Parallel Application of Rules by Static Allocation of Resources

We now study the computational power of simple P systems coming from the maximal parallel and nondeterministic application of the rules. To express this in terms of computational complexity, we envision a device capable of solving this problem and we express its complexity. We call this device a *resource allocator*, and abbreviate it by RA. Two operational semantics of membrane systems were defined in [1] and [2]. They differ only in the way the maximal parallel application of rules is described, and reflect the fact that resource allocation to rules can be done either statically or dynamically. A dynamic resource allocation is based on applying rules one by one in a nondeterministic manner until there is no applicable rule left [1].

An alternative is given by static allocation [2], where the existing resources are distributed in a nondeterministic and maximal way to the rules which then are applied in parallel. The equivalence between static and dynamic allocation semantics is proved in [2].

Therefore the purpose of the resource allocator is to allocate multisets of objects to rules such that the evolution is then done in a maximal parallel and nondeterministic way. Given this setup, the maximal parallel application of rules depends on RA being able to solve an instance of the discrete knapsack problem. The nondeterminism comes from the fact that we can choose different multisets of rules that correspond to the solution of the knapsack problem given the contents of RA and we choose one in a nondeterministic way. We can associate a resource allocator with every membrane of a simple P system. Given the parallel evolution of membranes, we note that every resource allocator resolves a particular instance associated with its membrane, and each one operates independent of the others. We represent multisets as a string over their support alphabet. A resource allocator can be formally defined as a mapping $RA : O^* \rightarrow (O^*)^{|R|+1}$, where O is the alphabet of objects, $w \in O^*$ and R is the set of rules associated with a membrane:

$$RA(w) = \{(w_1, w_2, \ldots, w_{|R|}, w') \mid w_i \not\subseteq w', i = \overline{1, |R|} \wedge \sum_{i=1}^{|R|} w_i + w' = w\} \quad (2)$$

The resource mapping problem can be formulated as follows: given a resource allocator RA of a simple P system, decide which of the rules are applied such that the system evolves in a maximal parallel and nondeterministic way. Formally, given (μ, w, R, RA) where

- μ is a membrane structure of a P system,
- w is the multiset of objects associated with μ,
- R is the set of rules associated with μ,
- RA is the resource allocator associated with μ,

maximize $|\sum_{i=1}^{|R|} w_i|$, where $(w_1, w_2, \ldots, w_{|R|}, w') \in RA(w)$ and $\exists\ u_1, \ldots, u_{|R|}$ such that $u_i \to v_i \in R\ \wedge u_i \neq u_j, \forall i \neq j\ \wedge\ \exists\ k_i \in \mathbb{N}\ w_i = k_i \cdot u_i, \forall i = \overline{1, |R|}$ and $\forall u \to v \in R$ we have that $u \nsubseteq w'$.

Note that the maximal parallel rewriting comes from the fact that the RA chooses nondeterministically which multiset of rules to apply (by assigning resources $w_1, w_2, \ldots, w_{|R|}$ to each rule), and the set is maximal because we cannot apply another rule using the remaining resources w'. When we have multiple solutions to the problem, we chose one nondeterministically.

To clarify the definition we give the following example: suppose that the resource allocator RA has to distribute the multiset $10a$ to the rules $4a \to b$, $3a \to c$ and $2a \to d$. We can now distribute the resources according to our definition in multiple ways. We show only a few:

$$10a \Rightarrow 2 \cdot 4a + 0 \cdot 3a + 1 \cdot 2a \text{ (no remaining a)}$$
$$10a \Rightarrow 0 \cdot 4a + 3 \cdot 3a + 0 \cdot 2a \text{ (remaining 1 a)}$$
$$10a \Rightarrow 1 \cdot 4a + 1 \cdot 3a + 1 \cdot 2a \text{ (remaining 1 a)}$$

Note that it is not possible to use the remaining resources to apply another rule, i.e. in our example we cannot have more than one remaining a because that means that we can apply another rule.

We show that the problem of resource mapping in simple P systems (shortly **RMP**) can be reduced to the *discrete knapsack problem* (shortly **KNAP**). In order to prove this, we first make a Karp reduction from **KNAP** to **RMP**.

Definition 3. *Let A,B be two decision problems over the alphabet Σ. We say that A can be Karp (or polynomial) reduced to B, and write $A \leq_m B$ if $\exists f : \Sigma^* \to \Sigma^*$, where f can be computed in deterministic polynomial time such that $x \in A \Leftrightarrow f(x) \in B, \forall x \in \Sigma^*$.*

Lemma 1. ***KNAP*** $\leq_m$ ***RMP****.*

Proof. We consider only the decision version of **RMP** and **KNAP**, because every optimization problem can be reduced to a decision one. We consider that the value of each object is equal with its weight, thus the knapsack problem becomes the *subset sum* problem (*subset sum* is also a **NP**-complete problem). We use the name **KNAP** because we are mainly interested in an implementation of the resource allocator. We transform in polynomial time an instance of **KNAP** into an instance of **RMP**. We denote the transformation by $f(c, n, W, P)$, and we show that $\mathbf{KNAP}(c, n, W, P) =$ yes implies $\mathbf{RMP}(f(c, n, W, P)) =$ yes, and $\mathbf{KNAP}\ (c, n, W, P) =$ no implies $\mathbf{RMP}(f(c, n, W, P)) =$ no. The transformation f has to create an instance of the **RMP** problem. We need to define a membrane structure μ, the set R of rules, a multiset w of objects in μ, and the resource allocator RA.
We use the same notations as above:

$$\begin{aligned}
&\mu \text{ is an arbitrary membrane structure,}\\
&w = a^c,\ a \in O,\\
&R = \{a^{w_i} \to b \mid b \in O,\ w_i \in W,\ \forall i = \overline{1,n}\},\\
&RA \text{ is a resource allocator defined as in equation (2).}
\end{aligned} \tag{3}$$

The transformation f is defined by the equations presented in (3). The membrane structure of μ can be chosen arbitrary because it is not involved in the distribution of object to rules. To express the capacity c of the knapsack, we define the contents of the membrane structure μ as a multiset composed of a single object a with multiplicity c. For every object we define a rule, such that if the object is used in the knapsack problem, then it will be used by RA only once. The transformation can be done in polynomial time with respect to the number of objects.

For the first part of the implication we start from $\mathbf{KNAP}(c, n, W, P) =$ yes, and we need to show that $\mathbf{RMP}(f(c, n, W, P)) =$ yes. Let us assume that $\mathbf{RMP}(f(c, n, W, P)) =$ no. This implies that there exists a better solution to the instance $f(c, n, W, P)$. Let the solution be $|\sum_{i=1}^{|R|} \bar{w}_i| \leq |w|$, where $(\bar{w}_1, \bar{w}_2, \ldots, \bar{w}_{|R|}, w') \in RA(w)$. We know that $|\sum_{i=1}^{|R|} \bar{w}_i| > |\sum_{i=1}^{|R|} w_i|$ because we assumed we have a better solution. Using this solution we construct a solution to $\mathbf{KNAP}(c, n, W, P)$ as follows:

$$x'_i = \begin{cases} 1 & , \quad \text{if } 0 < |\bar{w}_i| \\ 0 & , \quad \text{otherwise} \end{cases} \tag{4}$$

Note that if we have $|\bar{w}_i|$, then rule r_i has been used. We then have: $\sum_{i=1}^{n} p_i x'_i = \sum_{i=1}^{n} |\bar{w}_i| x'_i = |\sum_{i=1}^{n} \bar{w}_i| > |\sum_{i=1}^{n} w_i| = \sum_{i=1}^{n} p_i x_i$, because we assumed that we have a better solution. Thus we have $\mathbf{KNAP}\ (c, n, W, P) =$ no, which is a contradiction.

Now we prove that

$$\mathbf{KNAP}(c, n, W, P) = \text{no} \quad \text{implies} \quad \mathbf{RMP}(f(c, n, W, P)) = \text{no}$$

We consider that the decision problem was for the optimal value of T. Let us assume that we have $\mathbf{RMP}\ (f(c, n, W, P)) =$ yes, and the solution of this instance is formed by the following allocation: $(\bar{w}_1, \bar{w}_2, \ldots, \bar{w}_{|R|}, w') \in RA(w)$ and $|\sum_{i=1}^{|R|} \bar{w}_i| = T$. Like in the first implication, we construct an instance of **KNAP** such that $\mathbf{KNAP}(c, n, W, P) =$ yes. We construct this instance of **KNAP** as follows:

$$x'_i = \begin{cases} 1 & , \quad \text{if } 0 < |\bar{w}_i| \\ 0 & , \quad \text{otherwise} \end{cases} \tag{5}$$

We have $\sum_{i=1}^{n} p_i x'_i = T$. This is a contradiction, because we have $\mathbf{KNAP}$ $(c, n, W, P) =$ no.

Theorem 1. ***RMP*** *is* ***NP****-complete.*

Proof. We have

- **KNAP** $\leq_m$ **RMP**, by Lemma 1,
- **KNAP** is **NP**-complete,
- **NP** is closed under $\leq_m$.

Therefore **RMP** is **NP**-complete.

5 Complexity of Simple P Systems

We now present a way to use classic complexity theory classes to study the complexity of simple P systems. To use such an approach we need to take into account the distribution of objects to rules, rather than the number of steps performed in a computation because parallel evolution can consume more resources than sequential evolution w.r.t the number of steps.

We now extend the approach from [5], where the authors show that a P system can evolve using an **NP** oracle that solves the resource allocation problem for each of the membranes from the system. We avoid the oracle by using the resource allocator described in Section 4 which ensures the maximal parallelism and a nondeterministic evolution. Thus the parallel evolution of simple P systems can be viewed as a sequence of independent steps. Maximal parallel evolution means that we cannot apply another rule with the contents left in the membrane after the application of the selected rules. The maximal parallel application of rules depends on *RA* being able to solve an instance of the discrete knapsack problem and retrieve the solutions within a profit range that corresponds to this kind of evolution.

Each step is composed of three stages: the first consists of the assignment of objects to rules according to the resource allocator, the second represents the distribution of the results obtained from applying the selected rules, and finally the dissolution of certain membranes.

The first stage ensures a maximal parallel application of rules and consists of the creation of an instance of the discrete knapsack problem based on the multiset from the membrane, followed by solving the instance and obtaining the results. The second stage moves the objects obtained from the previous stage according to their tags. In the third stage we dissolve all membranes that contain the special symbol δ by transferring their resources to their parents (as an exception we do not transfer the δ symbols).

For the first stage of the process we present a function that transforms an instance of the resource allocation problem into an knapsack instance such that we can obtain the solution to the **RMP** instance by solving the transformed instance.

Given a membrane we define: the capacity c of the knapsack, the number n of objects, the weight w_i and value p_i for each object i:

$$
\begin{aligned}
&c = |w|;\\
&n = |\{w_{i_k} \mid w_{i_k} \text{ defined object}\}|\,;\\
&w_{i_k} = k \cdot |u_i|, \text{ where } R = \{r_1, \ldots, r_m\},\ r_i = u_i \to v_i,\ u_i \in O^* \wedge\\
&\quad k = \overline{1,p} \text{ where } p = \max\{j \in \mathbb{N} \mid w' \subseteq w \ \wedge\ w' = j \cdot u_i\};\ i = \overline{1,|R|};\\
&p_{i_k} = w_{i_k}
\end{aligned}
\tag{6}
$$

The transformation f is defined by the equations (6). Note that we do not need to use all the contents of the membrane. We denote by w the contents of the membrane, and by v the multiset which we intend to consume. We now have $w = v + v'$ which links w and v, where $\forall u \to v \in R$ we have that $u \nsubseteq v'$. This restriction assures us that we cannot apply a rule with the remaining contents of the membrane. In the knapsack problem an item can be used only once, but in membrane systems, because of maximal parallelism, a rule can be applied several times to all objects which it can process. Thus we need to define for every rule a "class" of items which represent all the possible ways in which a rule could be used. We denote by W the set of items defined; thus we have $|W| \leq c \cdot |R|$. The profit of an object is defined as the number of symbols it consumes – because we are interested in consuming as many symbols as possible. The transformation f can be computed in polynomial time with respect to $|w|$.

In the first stage we transform an instance of the resource allocation problem to an instance of the knapsack problem by using the function f. Then we solve the created instance, and obtain the rules which can be applied in parallel, together with the multiplicity of each rule. We can now express the computational complexity of each stage with respect to the input, represented by the multiset of the membrane.

For the first stage we need to express a relation between the number of objects created and the size of the multiset. From equations (6) we have that each $u_i \to v_i$ rule can introduces a maximum of $\frac{|w|}{|u_i|}$ objects. Summing these relations for each rule we have that:

$$n \leq \sum_{i=1}^{|R|} \frac{|w|}{|u_i|} = |w| \sum_{i=1}^{|R|} \frac{1}{|u_i|}$$

Note that $\sum_{i=1}^{|R|} \frac{1}{|u_i|}$ is a constant associated with the membrane, because the rules of a membrane do not change in the process of evolution. We denote this constant with S, and have that $n \leq |w| \cdot S$. Thus the complexity of this stage is $O(n) = O(|w| \cdot S)$.

For the second stage we use a pseudo-polynomial algorithm for knapsack with a complexity of $O(n \cdot c)$, where n is the number of objects, and c is the capacity of the knapsack. By using the relations (from the first stage) between the number of objects and the capacity, we have that the second stage has a complexity of $O(|w| \cdot S \cdot |w|) = O(|w|^2 \cdot S)$.

The third stage consists of applying the rules according to the solution from the knapsack problem. Note that a maximal parallel evolution does not imply a maximum profit. According to this, we chose nondeterministically a solution that corresponds to such a behaviour. Using the knapsack algorithm we compute all valid profits, so we need to chose only the ones which correspond to such an evolution. To achieve this, we define for each symbol a minimum allowed profit expressed as the difference between the multiplicity of the symbol from the input multiset and the minimum multiplicity of a rule that uses the symbol. Formally, $p_a^{min} = w(a) - \min_{u \to v \in R}\{u(a) | u(a) > 0\} + 1$. We introduce a lower bound in terms of profit, defined by summing the minimum allowed profit for each symbol. This assures that no rule can be applied using the remaining multiset. We know that for the pseudo-polynomial algorithm for knapsack we can retrieve the selected objects in $O(n)$, thus the complexity of this step is $O(|w| \cdot S)$. During this backtracking process we nondeterministically choose a rule that corresponds to the object chosen. Thus we obtain a multiset of rules that corresponds to maximal parallel evolution because no rule can be applied with the remaining contents and the evolution is nondeterministic because of the way the rules were chosen.

Using the example in Section 4, we show how we can distribute $10a$ as $3 \cdot 3a$(remaining $1a$). Using equations (6) we obtain the items in Table 3. By applying the knapsack algorithm we get the results in Table 4. In this case, the value of $p_a^{min} = 10 - 2 + 1 = 9$. Using the recurrence relation defined in equation (1) we obtain the items used of the solution. At each step $i = \overline{1, n}$ we test whether the item $n - i + 1$ was included in the knapsack or not. According to p_a^{min} the starting value can be 9 or 10. Suppose we start with the value 9. To test if object 7 was used we find the maximum of $f_6(9) = 9$ and $f_6(9-10)+10 = -\infty$. The maximum is $f_6(9)$. We continue until we reach the f_0 line.

i	1	2	3	4	5	6	7
w_i	2	3	4	6	8	9	10
p_i	2	3	4	6	8	9	10

Table 3. Items obtained using the transformation for the example in Section 4

This process is illustrated in Table 5, where X represents the remaining weight in the knapsack, $f_{i-1}(X)$ and $f_{i-1}(X - w_i) + p_i$ represent the alternatives between including or not the object and max represent the chosen value. The recurrence is also illustrated in Table 6, where the value chosen at step i is highlighted with a box and a subscript indicating the step. The solution is given by object 6 which has weight 9 and profit 9. This object corresponds to the multiset of rules composed of three times the rule with $3a$ as left-hand side. In conclusion, the algorithm tells us we can apply the rule with $3a$ three times and process only $9a$ out of $10a$.

Thus a complexity of a single evolution step for a membrane is

$$O(|w| \cdot S) + O(|w|^2 \cdot S) + O(|w| \cdot S).$$

X	0	1	2	3	4	5	6	7	8	9	10
f_0	0	0	0	0	0	0	0	0	0	0	0
f_1	0	0	2	2	2	2	2	2	2	2	2
f_2	0	0	2	3	3	5	5	5	5	5	5
f_3	0	0	2	3	4	5	6	7	7	9	9
f_4	0	0	2	3	4	5	6	7	8	9	10
f_5	0	0	2	3	4	5	6	7	8	9	10
f_6	0	0	2	3	4	5	6	7	8	9	10
f_7	0	0	2	3	4	5	6	7	8	9	10

Table 4. Solution to the knapsack instance in Table 3

i	7	6	5	4	3	2	1
X	9	9	0	0	0	0	0
$f_{i-1}(X)$	9	9	0	0	0	0	0
$f_{i-1}(X-w_i)+p_i$	$-\infty$	9	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
max	$f_6(9)$	$f_5(0)$	$f_4(0)$	$f_3(0)$	$f_2(0)$	$f_1(0)$	$f_0(0)$

Table 5. The backtracking process

X	0	1	2	3	4	5	6	7	8	9	10
f_0	$\boxed{0}_7$	0	0	0	0	0	0	0	0	0	0
f_1	$\boxed{0}_6$	0	2	2	2	2	2	2	2	2	2
f_2	$\boxed{0}_5$	0	2	3	3	5	5	5	5	5	5
f_3	$\boxed{0}_4$	0	2	3	4	5	6	7	7	9	9
f_4	$\boxed{0}_3$	0	2	3	4	5	6	7	8	9	10
f_5	$\boxed{0}_2$	0	2	3	4	5	6	7	8	9	10
f_6	0	0	2	3	4	5	6	7	8	$\boxed{9}_1$	10
f_7	0	0	2	3	4	5	6	7	8	$\boxed{9}_0$	10

Table 6. Finding the solution to the knapsack instance in Table 3

After all membranes have evolved through these three stages, we need to distribute the resources produced by them to show how the multiset of each membrane evolves. Thus we seek to find a relation between the contents of two consecutive configurations. Formally for two configurations $C_1 = (\mu, w_{i_1}, \ldots, w_{i_k})$, $C_2 = (\mu', w'_{j_1}, \ldots, w'_{j_l})$, where $C_1 \Rightarrow C_2$ we need to express the relation between $w'_{j_p} p = \overline{1,l}$ and $w_{i_q} q \in \overline{1,k}$. The contents of a membrane change from the application of rules. Following the definition of a rule, we see that we have four different situations for a new produced symbol: the symbol remains in the membrane, the symbol goes to the parent membrane, the symbol goes to a specific membrane, and the membrane dissolves passing all its contents to the parent. We know that μ' is obtained from μ by dissolving some of the membranes, thus $l \leq k$. We introduce a function $t : \overline{1,k} \to \overline{1,l}$ where $t(p) = 1$ if the membrane with label p from μ is not dissolved and $t(p) = 0$ otherwise. We introduce the following notations:

- w_i^{alloc} the multiset allocated by the resource allocator for membrane i;
- $selected : \mathbb{N}^2 \to \mathbb{N}$, $selected(i,j) = k$, where $k \in \{k \cdot u_j \mid \exists u_j \to v_j \in R_i \wedge w_j = k \cdot u_j \wedge w_j$ has been allocated to rule $j\}$ representing the number of times rule $u_j \to v_j \in R_i$ has been selected by the resource allocator;
- $w_i^{here} = \bigcup\{k \cdot v_j(a_{here}) \cdot a \mid \exists u_j \to v_j \in R_i \wedge k = selected(i,j)\}$ representing the multiset of objects produced by the selected rules which remain in i;
- $w_i^{in} = \bigcup\{k \cdot s \cdot a \mid \exists u_j \to v_j \in R_l \wedge k = selected(l,j) \wedge s = v_j(a_{in_i}) + v_j(a_{out}),\ i$ is the parent of $l\}\}$ representing the multiset of objects which have been produced by other membranes and have been transported to membrane i;
- $w_i^{dis} = \bigcup\{w \mid \exists u_j \to v_j\delta \in R_l \wedge selected(l,j) > 0 \wedge w$ the contents of $l \wedge i$ is the parent of $l\}$ representing the multiset of objects produced by rules which dissolve a membrane.

Thus we have the following relations:

$$w'_{j_p} = \begin{cases} w_{j_p} - w_{j_p}^{alloc} + w_{j_p}^{here} + w_{j_p}^{in} + w_{j_p}^{dis} & , \quad t(j_p) = 1 \\ 0 \quad , \quad \text{otherwise} \end{cases} \tag{7}$$

This means that the local symbols are first transported, followed by the symbols from other membranes, and finally the symbols produced by dissolving a membrane. We do this in order to ensure that the produced symbols reach their destination membrane. If we do not consider dissolution as the last operation, the symbols that were supposed to reach other membranes would pass on to the parent of the dissolving membrane.

6 Conclusion

In this paper we describe the computational complexity of the simple P system in classical complexity theory, extending the approach shortly presented in [5]. We show that the complexity of certain membrane systems called *simple P systems* can be studied using the classical complexity theory. The evolution of such a system is studied by using a resource allocator which solves the resource allocation problem using a well-known combinatorial problem.

The allocation of the resources to the rules is an important step in the nondeterministic and maximally parallel evolution of a simple P system. We consider the static allocation of resources towards the parallel application of the rules, and study the computational complexity of a subclass of P systems by reducing the resource allocation problem to the knapsack problem.

Trading space for time (as many models of natural computing), one can show that **PMC** = **PSPACE**, where **PMC** is the class of problems which can be solved in polynomial time by P systems of a given type [8, 9]. Membrane computing brings **PSPACE** to polynomial time in the sense that given a problem $X \in$ **PSPACE** there exists a deterministic Turing machine that constructs in polynomial time Π_X,

a P system that solves X in polynomial time. Computing **PSPACE** in polynomial time means that we have a family of membrane systems for a given **PSPACE** problem such that the n-th membrane system solves the problem in polynomial time for inputs of size less than or equal to n. Recently, Sosik and Rodriguez-Paton provide a characterization of **PSPACE** by showing that confluent P systems with active membranes solve in polynomial time exactly the class of problems **PSPACE** [10].

References

1. O. Andrei, G. Ciobanu, D. Lucanu. A Rewriting Logic Framework for Operational Semantics of Membrane Systems, *Theoretical Computer Science* vol. 373, 163-181, 2007.
2. O. Agrigoroaiei, G. Ciobanu. Rewriting Logic Specification of Membrane Systems with Promoters and Inhibitors, to appear in *Electronic Notes of Theoretical Computer Science*, 2008.
3. C.S. Calude, Gh. Păun. Bio-steps Beyond Turing. *BioSystems* vol.77, 175-194, 2004.
4. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez. *Application of Membrane Computing*, Springer, 2006.
5. G. Ciobanu, M. Gontineac. Mealy Membrane Automata and P Systems Complexity. In M.A.Gutierrez-Naranjo, Gh.Păun, M.J.Perez-Jimenez (Eds.): *Cellular Computing; complexity aspects*, ESF PESC Exploratory Workshop, Fenix Editora, Sevilla, 149-164, 2005.
6. T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms*, MIT Press, 1990.
7. Gh. Păun. *Membrane Computing. An Introduction*, Springer, 2002.
8. Gh. Păun. P Systems with Active Membranes: Attacking NP Complete Problems, *J. Autom. Lang. Comb.* vol.6(1), 75-90, 2001.
9. M. Perez Jimenez, A.R. Jimenez, F. Sancho-Caparrini. Complexity Classes in Models of Cellular Computing with Membranes, *Natural Computing* vol.2, 265-285, 2003.
10. P. Sosik, A. Rodriguez-Paton. Membrane Computing and Complexity Theory: A characterization of PSPACE. *Journal of Computer and System Sciences* vol 73(1), 137-152, 2007.

Solving the Partition Problem by Using Tissue-like P Systems with Cell Division

Daniel Díaz-Pernil, Miguel A. Gutiérrez-Naranjo,
Mario J. Pérez-Jiménez, Agustín Riscos-Núñez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mails: {sbdani,magutier,marper,ariscosn}@us.es

Summary. Tissue-like P systems with cell division is a computing model in the framework of Membrane Computing that shares with the spiking neural P system model a similar biological inspiration. Namely, both models are based on the intercellular communication and cooperation between neurons, respectively. Due to this fact, in both models the devices have the same structure: a network of elementary units (cells in a tissue and interconnected neurons, respectively). Nonetheless, the two models are quite different. One of the differences is the ability of tissue-like P systems with cell division for increasing the number of cells during the computation. In this paper we exploit this ability and present a polynomial-time solution for the (**NP**-complete) Partition problem via a uniform family of such P systems.

1 Introduction

Tissue-like P systems with cell division [13] is a computing model in the framework of membrane computing based on inter-cellular communication and cooperation between neurons. It shares some common features with another emerging membrane computing model based on spiking neurons, the *spiking neural P systems* [15]. Their main common feature is that in the computational devices of both models we have certain processor units (called *cells* or *neurons*, respectively) that process in parallel some pieces of information and send signals to other processor units along links that connect some of them. Such links do not follow any scheme, and this is one of the features which distinguishes these models from the initial model in membrane computing, the cell-like model, where membranes are hierarchically arranged in a tree-like structure (see [10]). The biological inspiration for this cell-like model is the morphology of cell, where small vesicles are surrounded by larger ones.

In spiking neural P systems and in tissue-like P systems with cell division the membrane structure is tissue-like and the links between cells form a general graph

(directed graph for spiking neural P systems and undirected graph for tissue-like P systems). Nonetheless there are important differences between both models. For instance, in spiking neural P systems only one type of object (called *spike*) is used to encode the information in the cells. Specific rules are used for evolving populations of spikes and the time is used as a support of information[1].

As we said above, in tissue-like P systems we can picture the cells as nodes of a general undirected graph. The edges of such graph are not given explicitly, but they are deduced from the set of rules, as it will be explained later. The communication among cells is based on symport/antiport rules in P systems[2]. 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 definition of tissue P systems [7, 8], several research lines have been developed and other variants have arisen (see, for example, [1, 2, 3, 4, 6, 14]). One of the most interesting variants of tissue P systems was presented in [13]. In that paper, tissue P systems are endowed with the ability of getting new cells based on the *mitosis* or cellular division, yielding *tissue-like P systems with cell division*, and the underlying graph is implicitly described by the rules.

This cellular division is other of the main differences between the model followed in this paper and spiking neural P systems. The ability of cell division allows us to obtain an exponential amount of cells in linear time and to design cellular solutions to **NP**-complete problems in polynomial time. Nonetheless, the solutions to **NP**-complete problems in the spiking neural P systems literature need an exponential amount of pre-computed devices (see [5]).

In this paper we present a solution to the Partition problem via a family of recognizing tissue-like P systems with cell division. In the literature we can find uniform solutions to this problem in the cell-like model of P systems with active membranes, but this is the first solution to Partition in the framework of tissue-like P systems with cell division.

The paper is organized as follows: first we recall some preliminaries and the definition of tissue-like P systems with cell division. Next, recognizing tissue-like P systems with cell division are briefly described in section 3. A linear–time solution to the Partition problem is presented in the section 4, including a short overview of the computation and of the necessary resources. Finally, some conclusions and new open research lines are presented.

2 Preliminaries

In this section we briefly recall some of the concepts used later on in the paper.

An *alphabet*, Σ, is a non empty set, whose elements are called *symbols*. An ordered sequence of symbols is a *string*. 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

[1] A detailed description can be found in [16] and the references therein.
[2] This way of communication for P systems was introduced in [12].

length 0) will be denoted by λ. The set of strings of length n built with symbols from the alphabet Σ is denoted by Σ^n and $\Sigma^* = \cup_{n\geq 0}\Sigma^n$. A *language* over Σ is a subset from Σ^*.

A *multiset* 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 *s*upport is defined as $supp(m) = \{x \in A \mid f(x) > 0\}$ and its *size* is defined as $\sum_{x\in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite).

If $m = (A, f)$ is a finite multiset over A, then it will be denoted by $m = a_1^{f(a_1)}a_2^{f(a_2)}\cdots a_k^{f(a_k)}$, where $supp(m) = \{a_1, \ldots, a_k\}$, and for each element a_i, $f(a_i)$ is called the multiplicity of a_i.

A *undirected graph* G is a pair $G = (V, E)$ where V is the set of vertices and E is the set of edges, each one of which is a (unordered) pair of (different) vertices. If $\{u, v\} \in E$, we say that u is *adjacent* to v (and also v is *adjacent* to u). The *degree* of $v \in V$ is the number of adjacent vertices to v.

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems. For details, see [11].

3 Tissue-like P Systems with Cell Division

In the first definition of the model of tissue P systems [7, 8] the membrane structure did not change along the computation. Based on the cell-like model of P systems with active membranes, Gh. Păun et al. presented in [13] a new model of tissue P systems *with cell division*. The biological inspiration is clear: alive tissues are not *static* network of cells, since cells are duplicated via mitosis in a natural way.

The main features of this model, from the computational point of view, are that cells have not polarizations (the contrary holds in the cell-like model of P systems with active membranes, see [11]); 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 blocked during the mitosis process. In some sense, this means that while a cell is dividing it closes the communication channels with other cells and with the environment.

Formally, a *tissue-like P system with cell division* of degree $q \geq 1$ is a tuple of the form

$$\Pi = (\Gamma, \mathcal{E}, w_1, \ldots, w_q, \mathcal{R}, i_0),$$

where:

1. Γ is a finite *alphabet*, whose symbols will be called *objects*.
2. $w_1, \ldots, w_q$ are strings over Γ representing the multisets of objects associated with the cells in the initial configuration.
3. $\mathcal{E} \subseteq \Gamma$.
4. $\mathcal{R}$ is a finite set of rules of the following form:
   (a) *Communication rules*: $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$.
   (b) *Division rules*: $[a]_i \to [b]_i[c]_i$, where $i \in \{1, 2, \ldots, q\}$ and $a, b, c \in \Gamma$.

5. $i_0 \in \{0, 1, 2, \ldots, q\}$.

A tissue-like P system with cell division of degree $q \geq 1$ can be seen as a set of q cells (each one consisting of an elementary membrane) labelled by $1, 2, \ldots, q$. We shall use 0 to refer to the label of the environment, and i_0 denotes the output region (which can be the region inside a cell or the environment).

The communication rules determine a virtual graph, where the nodes are the cells and the edges indicated if it is possible for pairs of cells to communicate directly. This is a dynamical graph, because of new nodes can appear produced by the application of division rules.

The strings $w_1, \ldots, w_q$ describe the multisets of objects placed in the q cells of the system. We interpret that $\mathcal{E} \subseteq \Gamma$ is the set of objects placed in the environment, each one of them in an arbitrary large amount of copies.

The communication rule $(i, u/v, j)$ can be applied over two cells i and j such that u is contained in cell i and v is contained in cell j. The application of this rule means that the objects of the multisets represented by u and v are interchanged between the two cells.

The division rule $[a]_i \rightarrow [b]_i[c]_i$ is applied over a cell i containing object a. The application of this rule divides this cell into two new cells with the same label. All the objects in the original cell are replicated and copied in each of the new cells, with the exception of the object a, which is replaced by the object b in the first one and by c in the other one.

Rules are used as usual in the framework of membrane computing, that is, in a maximally parallel way (a universal clock is considered). In one step, each object in a membrane can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can participate in a rule of any form must do it, i.e, in each step we apply a maximal set of rules. This way of applying rules has only one restriction when a cell is divided, the division rule is the only one which is applied for that cell in that step; the objects inside that cell do not evolve in that step.

3.1 Recognizing Tissue-like P Systems with Cell Division

NP-completeness has been usually studied in the framework of *decision problems*. 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.

In order to study the computing efficiency for solving **NP**-complete decision problems, a special class of tissue P systems with cell division is introduced in [13]: *recognizing tissue P systems*. The key idea of such recognizing systems is the same one as from recognizing P systems with cell-like structure.

Recognizing cell-like P systems were introduced in [9] and they are the natural framework to study and solve decision problems within Membrane Computing, since deciding whether an instance of a given problem has an affirmative or negative

answer is equivalent to deciding if a string belongs or not to the language associated with the problem.

In the literature, recognizing cell-like P systems are associated with P systems with *input* in a natural way. The data encoding to an instance of the decision problem has to be provided to the P system in order to compute the appropriate answer. This is done by codifying each instance as a multiset placed in an *input membrane*. The output of the computation (`yes` or `no`) is sent to the environment, and in the last step of the computation. In this way, cell-like P systems with input and external output are devices which can be seen as black boxes, in the sense that the user provides the data before the computation starts, and then waits *outside* the P system until it sends to the environment the output in the last step of the computation.

A recognizing tissue-like P system with cell division of degree $q \geq 1$ is a tuple

$$\Pi = (\Gamma, \Sigma, \mathcal{E}, w_1, \ldots, w_q, \mathcal{R}, i_{in}, i_0)$$

where

- $(\Gamma, \mathcal{E}, w_1, \ldots, w_q, \mathcal{R}, i_0)$ is a tissue-like P system with cell division of degree $q \geq 1$ (as defined in the previous section), $i_0 = env$ and $w_1, \ldots, w_q$ strings over $\Gamma \setminus \Sigma$.
- The working alphabet Γ has two distinguished objects `yes` and `no`, present in at least one copy in some initial multisets $w_1, \ldots, w_q$, but not present in $\mathcal{E}$.
- Σ is an (input) alphabet strictly contained in Γ.
- $i_{in} \in \{1, \ldots, q\}$ is the input cell.
- All computations halt.
- If $\mathcal{C}$ is a computation of Π, then either the object `yes` or the object `no` (but not both) must have been released into the environment, and only in the last step of the computation.

The computations of the system Π with input $w \in \Sigma^*$ start from a configuration of the form $(w_1, w_2, \ldots, w_{i_{in}} w, \ldots, w_q; \mathcal{E})$, that is, after adding the multiset w to the contents of the input cell i_{in}. We say that the multiset w is *recognized* by Π if and only if the object `yes` is sent to the environment, in the last step of the corresponding computation. We say that $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object `yes` (respectively, `no`) appears in the environment associated to the corresponding halting configuration of $\mathcal{C}$.

Definition 1. *We say that a decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ of recognizing tissue-like P systems with cell division 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 (called a polynomial encoding of I_X in $\mathbf{\Pi}$) 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* $\Pi(s(u))$*;*
- *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, moreover, 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.*

In the above definition we have imposed to every P system $\Pi(n)$ to be *confluent*, in the following sense: every computation of a system with the *same* input multiset must always give the *same* answer.

We denote by $\mathbf{PMC}_{TD}$ the set of all decision problems which can be solved by means of recognizing tissue-like P systems with cell division in polynomial time. This class is closed under polynomial reduction and under complement.

4 A solution for the Partition Problem

Let us recall that a partition of a set V is a family of non-empty pairwise disjoint subsets of V such that the union of the subsets of the family is equal to V.

The Partition Problem (`PART`) can be settled as follows: Let V be a finite set and let w be a weight function on V, $w : V \to \mathbb{N}$ (that is, an additive function). Decide whether or not there exists a partition $\{V_1, V_2\}$ of V such that $w(V_1) = w(V_2)$.

Next, we shall prove that the Partition problem can be solved in a linear time (in $\{n, \lg k\}$ where $k = \omega_1 + \cdots + \omega_n$) by a family of recognizing tissue-like P systems with cell division (in the sense of Definition 1).

Given an instance $u = (V, w)$ of the Partition Problem, we will denote $V = \{v_1, v_2, \ldots, v_n\}$. Such instance will be represented by $u = (n, (w_1, \ldots, w_n))$, where $w_i = w(v_i)$, for each i $(1 \leq i \leq n)$.

Next, we present a family of recognizing tissue-like P systems with cell division where at the initial configuration each system of the family has two cells (labelled by 1 and 2). We shall address the resolution via a brute force algorithm, which consists in the following stages:

- *Generation Stage*: All the possible subsets of V are generated by the application of cell division rules.
- *Pre–checking Stage*: In this stage, the weight of each of the subsets of V is calculated.
- *Checking Stage*: We compare for each subset if its weight and the weight of its complementary set are equal.

- *Answer Stage*: According to the previous stage, an affirmative or negative response is obtained.

For each $n, k \in \mathbb{N}$ we will consider the recognizing tissue-like P system with cell division and symport/antiport rules

$$\Pi(< n, k >) = (\Gamma, \Sigma, \mathcal{E}, w_1, w_2, R, i_{in})$$

defined as follows

- $\Gamma = \{A_i, \overline{A}_i, B'_i, B_i : 1 \leq i \leq n\} \cup$
  $\{a_i : 1 \leq i \leq \lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 14\} \cup \{c_i, v_i : 1 \leq i \leq n\} \cup$
  $\{d_i, g_i : 1 \leq i \leq \lceil \lg\, n \rceil + 1\} \cup \{e_i : 1 \leq i \leq \lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 5\} \cup$
  $\{A_{ij}, B_{ij} : 1 \leq i \leq n \wedge 1 \leq j \leq \lceil \lg\, k \rceil + 1\} \cup$
  $\{b, D, D_1, p, q, E_1, F_1, F_2, T, S, N, \texttt{yes}, \texttt{no}\}$
- $\Sigma = \{v_1, \ldots, v_n\}$
- $\mathcal{E} = \Gamma \setminus \{a_1, b, c_1, \texttt{yes}, \texttt{no}, D, A_1, ..., A_n, \overline{A}_1, \ldots, \overline{A}_n\}$.
- $w_1 = a_1\, b\, c_1\, \texttt{yes}\, \texttt{no}$ and $w_2 = D\, A_1 \ldots A_n, \overline{A}_1\, \ldots, \overline{A}_n$.
  Also, we consider that in the environment there are infinitely many copies of each object from $\mathcal{E}$, and no copies of any element in $\Gamma \setminus \mathcal{E}$.
- R is the following set of rules:
  1. *Division rules:*
     $r_{1,i} \equiv [A_i]_2 \rightarrow [B_i]_2[\lambda]_2$, for $i = 1, \ldots, n$
  2. *Communication rules:*
     $r_{2,i} \equiv (1, a_i/a_{i+1}, 0)$, for $i = 1, \ldots, n + \lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 11$
     $r_{3,i} \equiv (1, c_i/c_{i+1}^2, 0)$, for $i = 1 \ldots, n$
     $r_4 \equiv (1, c_{n+1}/D, 2)$
     $r_5 \equiv (2, c_{n+1}/D_1 g_1, 0)$
     $r_{6,i} \equiv (2, g_i/g_{i+1}^2, 0)$, for $i = 1, \ldots, \lceil \lg\, n \rceil$
     $r_7 \equiv (2, D_1/d_1 e_2, 0)$
     $r_{8,i} \equiv (2, d_i/d_{i+1}^2, 0)$, for $i = 1, \ldots, \lceil \lg\, n \rceil$
     $r_9 \equiv (2, d_{\lceil \lg\, n \rceil}/d_{\lceil \lg\, n \rceil + 1}, 0)$
     $r_{10,i} \equiv (2, e_i/e_{i+1}, 0)$, for $i = 1, \ldots, \lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 4$
     $r_{11,i} \equiv (2, g_{\lceil \lg\, n \rceil + 1} B_i/B'_i, 0)$, for $i = 1, \ldots, n$
     $r_{12,i} \equiv (2, B'_i \overline{A}_i/B_{i1}, 0)$, for $i = 1, \ldots, n$
     $r_{13,i} \equiv (2, d_{\lceil \lg\, n \rceil + 2} \overline{A}_i/A_{i1}, 0)$, for $i = 1, \ldots, n$
     $r_{14,ij} \equiv (2, B_{ij}/B_{ij+1}^2, 0)$, for $i = 1, \ldots, n$ and $j = 1, \ldots, \lceil \lg\, k \rceil$
     $r_{15,ij} \equiv (2, A_{ij}/A_{ij+1}^2, 0)$, for $i = 1, \ldots, n$ and $j = 1, \ldots, \lceil \lg\, k \rceil$
     $r_{16,i} \equiv (2, B_{i,\lceil \lg\, k \rceil + 1} v_i/p, 0)$, for $i = 1, \ldots, n$
     $r_{17,i} \equiv (2, A_{i,\lceil \lg\, k \rceil + 1} v_i/q, 0)$, for $i = 1, \ldots, n$
     $r_{18} \equiv (2, pq/\lambda, 0)$
     $r_{19} \equiv (2, e_{\lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 5}/E_1 F_1, 0)$
     $r_{20} \equiv (2, E_1 p/\lambda, 0)$
     $r_{21} \equiv (2, E_1 q/\lambda, 0)$
     $r_{22} \equiv (2, F_1/F_2, 0)$
     $r_{23} \equiv (2, E_1 F_2/T, 0)$

$r_{24} \equiv (2, T/\lambda, 1)$
$r_{25} \equiv (1, bT/S, 0)$
$r_{26} \equiv (1, S\texttt{yes}/\lambda, 0)$
$r_{27} \equiv (1, a_{n+\lceil \lg\, n\rceil+\lceil \lg\, k\rceil+12}b/N, 0)$
$r_{28} \equiv (1, noN/\lambda, 0)$

- $i_{in} = 2$, is the label of the input cell.

This family of recognizing tissue-like P systems with cell division and symport/antiport rules consists of *non–deterministic* systems, since several division rules can be applied in the cells labelled by 2. Nonetheless, if a division rule has not been applied yet to a cell labelled by 2, then it will be applied in the next steps since in the initial configuration, the unique cell labelled by 2 contains the objects $A_1, A_2, \ldots, A_n$, i.e., with respect to the division rules, the systems are confluent.

In order to justify that the family $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbb{N}}$ defined above provides a linear solution to the Partition problem we need a polynomial encoding (cod, s) of the set of instances of such a problem in the family $\mathbf{\Pi}$.

We will consider a polynomial enconding (cod, s) defined as follows: for each instance $u = (n, (w_1, \ldots, w_n))$ we define $s(u) =< n, w_1 + \cdots + w_n >$ and $cod(u) = v_1^{w_1}, \ldots, v_n^{w_n}$.

In this way, the instance $u = (n, (w_1, \ldots, w_n)) \in I_{\texttt{PART}}$ will be processed by the tissue-like P system $\Pi(s(u))$ with the multiset $cod(u)$ provided in the corresponding input cell.

Next, we will provide an informal description of the computations of the system $\Pi(s(u))$ with input $cod(u)$ for a generic instance u of the Partition problem, and we justify that the family defined above is polynomially uniform by deterministic Turing machines.

4.1 An overview of the computation

We informally describe here how the recognizing tissue-like P system with cell division $\Pi(s(u))$ with input $cod(u)$ works.

Let us start with the *generation stage*. In this stage we have two parallel processes.

- On the one hand, in the cell labelled by 1 we have two counters: a_i, which will be used in the output stage, and c_i, which will be multiplied until step n, where 2^n copies of c_{n+1} are obtained.
- On the other hand, in the cell labelled by 2, the division rules are applied. For each object A_i we produce two cells labelled by 2, one of them containing a new object B_i and the other one not.
  After the appropriate divisions, in the step n we obtain exactly 2^n cells with label 2, and each of them encode a different subset of V.

The pre–checking stage starts at the step $(n + 1)$, where each cell labelled by 2 trades the object D against the counter c_{n+1} from the cell 1 (by applying in

parallel the rule r_4). From that moment on, only the evolution of the counter a_i will be performed in cell 1, till the step $n + \lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 13$, via the rules $r_{1,i}$ ($n + 2 \leq i \leq n + \lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 12$).

Note that in the next step, the objects c_{n+1} in the cells labelled by 2 will trigger the rules r_5 and r_7 in the next two steps, thus bringing in the counter g_i in the step $n + 2$, and the counters d_i and e_i in the step $n + 3$.

From the step $n + 3$ to the step $n + \lceil \lg\, n \rceil + 3$ the counter g_i duplicates itself (with the rules $r_{6,i}$) until producing at least n copies of the object $g_{\lceil \lg\, n \rceil + 1}$, and in a further step, it yields the trading of the objects B_i in each cell with label 2 against the objects B'_i from the environment (by the application of the rules $r_{11,i}$).

In the step $n + \lceil \lg\, n \rceil + 5$, each pair of objects B'_i and $\overline{A}_i$ that appear in a cell labelled by 2 are traded against an object B_{i1} by applying the rules $r_{12,i}$.

In parallel, from the step $n+4$ to the step $n + \lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 8$ the counter e_i is evolving until reaching the object $e_{\lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 5}$ (by applying the rules $r_{10,i}$)). Moreover, from the step $n + 4$ to the step $n + \lceil \lg\, n \rceil + 4$ the counter d_i duplicates itself (by the rules $r_{8,i}$) until getting at least n copies of the object $d_{\lceil \lg\, n \rceil + 1}$. In the next step, the rule r_9 trades the objects $d_{\lceil \lg\, n \rceil + 1}$ in the cells with label 2 against the objects $d_{\lceil \lg\, n \rceil + 2}$. The arrival of these objects to a cell with label 2 produces the trading of the objects $\overline{A}_i$ (which remain in the cell after the application of the rules $r_{12,i}$) against objects A_{i1} in the step $n + \lceil \lg\, n \rceil + 6$ (by applying the rules $r_{13,i}$).

In this way, we have in each cell with label 2 a pair of complementary subsets, encoded by the objects B_{i1} and A_{i1}, respectively.

From the step $n + \lceil \lg\, n \rceil + 7$ to the step $n + \lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 7$ the number of objects B_{i1} and A_{i1} are multiplied by 2 (by application of the rules $r_{14,ij}$ and $r_{15,ij}$, respectively) to reach, at least, k copies of the objects $B_{i,\lceil \lg\, k \rceil + 1}$ and $A_{i,\lceil \lg\, k \rceil + 1}$ ($1 \leq i \leq n$). Recall that $k = w_1 + \cdots + w_n$ represents the total weight of the initial set.

In order to obtain the weight of each one of the subsets, we take each pair of objects $B_{i,\lceil \lg\, k \rceil + 1}$ and v_i (respectively, $A_{i,\lceil \lg\, k \rceil + 1}$ and v_i) that appear in a cell with label 2, and they are traded against an object p (respectively, against an object q) according to the rules $r_{16,i}$ (respectively, $r_{17,i}$).

The *checking stage* starts in the step $n + \lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 8$ with the application of the rule r_{18} which removes from the cells labelled by 2 as many pairs of objects p and q as possible. Therefore, if a cell 2 encodes a pair of subsets of weight k, then all the objects p and q will be deleted in this cell. Otherwise, at least one object p or q will remain in this cell.

The answer stage starts in the step $n + \lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 9$. In the cells with label 2, the object $e_{\lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 5}$ is traded against the objects E_1 and F_1 by the rule r_{19}. From this step on, there are two possible situations:

- Let us suppose that there exists a couple of complementary subsets of V with weight k. In this case, there will exist a cell 2 such that it does not contain any object p or q after the step $n + \lceil \lg\, n \rceil + \lceil \lg\, k \rceil + 9$. Therefore, in the next step, neither rule r_{20} nor r_{21} can be applied in such cell. However, rule r_{22} is applied,

allowing the evolution of the counter F_1 to F_2. This object together with the object E_1 produces the object T that, in the step $n + \lceil \lg\ n \rceil + \lceil \lg\ k \rceil + 12$ goes to the cell labelled by 1.
In the next step, the objects T and b that initially were in the cell 1 produce the object S. This object allows to send an object `yes` to the environment in the step $n + \lceil \lg\ n \rceil + \lceil \lg\ k \rceil + 14$, which ends the computation. In this case, we have an accepting computation.

- Let us suppose now that there does not exist a pair of complementary subsets of V such that its weights are both equal to k. In this case, all the cells labelled by 2 contain either objects p or q (but not both of them simultaneously). Then, in the step $n + \lceil \lg\ n \rceil + \lceil \lg\ k \rceil + 10$, the object E_1 is removed from these cells labelled by 2 together with a copy of p or q (by application of the rules r_{20} or r_{21}). In the meantime, the object F_1 evolves to F_2 (by the rule r_{22}). In this way, after the step $n + \lceil \lg\ n \rceil + \lceil \lg\ k \rceil + 13$ the object b remains in the cell 1. This object together with the object $a_{n+\lceil \lg\ n \rceil + \lceil \lg\ k \rceil + 14}$ produces an object N, which is sent to the environment together with an object `no` in the step $n + \lceil \lg\ n \rceil + \lceil \lg\ k \rceil + 15$. This step ends the computation with a negative answer.

Polynomial Uniformity of the Family

In order to stablish that the family $\mathbf{\Pi} = (\Pi(t))_{t \in \mathbb{N}}$ is polynomially uniform by deterministic Turing machines firstly we note that the set of rules associated with the system $\Pi(< n, k >)$ is described in a recursive way. Hence, we only need to justify that the amount of necessary resources for defining the system is polynomial in $\max\{n, \lceil \lg\ k \rceil\}$. The necessary resources for building $\Pi(< n, k >)$ are the following:

- Size of the alphabet: $2n \cdot \lceil \lg\ k \rceil + 7n + 2\lceil \lg\ k \rceil + 3\lceil \lg\ n \rceil + 36 \in \theta(n \cdot \lceil \lg\ k \rceil)$,
- Initial number of cells: $2 \in \theta(1)$,
- Initial number of objects: $2n + 6 \in \theta(n)$,
- Number of rules: $2n \cdot \lceil \lg\ k \rceil + 6n + 2\lceil \lg\ k \rceil + 5\lceil \lg\ n \rceil + 33 \in \theta(n \cdot \lceil \lg\ k \rceil)$,
- Upper bound for the length of the rules: $3 \in \theta(1)$.

Then, we have the following result:

Theorem 1. `PART`$\in \mathbf{PMC}_{TD}$.

Taking into account that `PART` is an **NP**-complete problem, we can deduce the following result.

Corollary 1. $\mathbf{NP} \subseteq \mathbf{PMC}_{TD}$.

5 Conclusion and Future Work

Tissue-like P systems with cell division is a computing model in the framework of Membrane Computing that shares with the spiking neural P system model the tissue-like structure of cells and the biological inspiration, since both models are based on the intercellular communication and cooperation between neurons. Nonetheless, both models are quite different. One of the main differences is the treatment of the information and how the flow of information between rules is handled. The second main difference is the ability of tissue-like P systems with cell division for increasing the number of cells during the computation. In a similar way to other P system models, this ability can be used for trading space against time and obtaining polynomial-time solutions for **NP** problems by obtaining an exponential amount of new cells during the computation.

One of the main drawbacks of spiking neural P systems in order to design solutions for **NP** problems is that the cell structure cannot change along the computation, so in order to get solutions of hard problems, the design needs to use precomputed resources. An open research line for the future is to study if some of the features of tissue-like P systems can be adapted to spiking neural P systems in order to get new applications to these new systems.

Acknowledgment

The authors wish to acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the project of excellence TIC-581 of the Junta de Andalucía.

References

1. A. Alhazov, R. Freund, M. Oswald: Tissue P Systems with Antiport Rules and Small Numbers of Symbols and Cells. *Lecture Notes in Computer Science* **3572**, (2005), 100–111.
2. F. Bernardini, M. Gheorghe: Cell Communication in Tissue P Systems and Cell Division in Population P Systems. *Soft Computing* **9**, 9, (2005), 640–649.
3. R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue P Systems with Channel States. *Theoretical Computer Science* **330**, (2005), 101–116.
4. S.N. Krishna, K. Lakshmanan, R. Rama: Tissue P Systems with Contextual and Rewriting Rules. *Lecture Notes in Computer Science* **2597**, (2003), 339–351.
5. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving Numerical NP-Complete Problems with Spiking Neural P Systems. 336-352. *Lecture Notes in Computes Science* **4860**, (2007), 336–352.
6. K. Lakshmanan, R. Rama: 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), 304–318.

7. 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.
8. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón: Tissue P Systems. *Theoretical Computer Science*, **296**, (2003), 295–326.
9. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: A Polynomial Complexity Class in P Systems Using Membrane Division. *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003*, (E. Csuhaj-Varjú, C. Kintala, D. Wotschke, Gy. Vaszyl, eds.), (2003), 284–294.
10. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences*, **61**, 1, (2000), 108–143.
11. Gh. Păun: *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, (2002).
12. A. Păun, Gh. Păun: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing*, **20**, 3, (2002), 295–305.
13. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez: Tissue P System with Cell Division. In Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez and F. Sancho-Caparrini (eds.), *Second Brainstorming Week on Membrane Computing*, Sevilla, Report RGNC 01/2004, (2004), 380–386.
14. V.J. Prakash: 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), 356–364.
15. M. Ionescu, Gh. Păun and T. Yokomori: Spiking Neural P Systems. *Fundamenta Informaticae*, **71**, 2-3 (2006), 279–308.
16. Gh. Păun: Twenty Six Research Topics About Spiking Neural P Systems. In *Fifth Brainstorming Week on Membrane Computing*, (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez, A. Riscos-Núñez, eds.) Fénix Editora, Sevilla, 2007, 263–280.

P-Lingua: A Programming Language for Membrane Computing

Daniel Díaz–Pernil, Ignacio Pérez–Hurtado,
Mario J. Pérez–Jiménez, Agustín Riscos–Núñez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mails:{sbdani,perezh,marper,ariscosn}@us.es

Summary. Software development for cellular computing has already been addressed, yielding a first generation of applications. In this paper, we develop a new programming language: P-Lingua. Furthermore, we present a simulator for the class of recognizing P systems with active membranes. We illustrate it by giving a solution to the SAT problem as an example.

1 Introduction

Membrane computing (or cellular computing) is an emerging branch within natural computing that was introduced by Gh. Păun [4]. The main idea is to consider biochemical processes taking place inside living cells from a computational point of view, in a way that gives us a new nondeterministic model of computation by using cellular machines.

Since the model was presented, many software applications have been produced (see [2], [10]). The common purpose of all of these software applications is to simulate P systems devices (cellular machines), and hence the designers have faced similar difficulties. However, these systems were usually focused on, and adapted for, particular cases, making it difficult to work on generalizations.

In order to give the first steps towards a next generation of applications, it is convenient to agree on some standards (specifications that regulate the performance of specific processes in order to guarantee their interoperability) and to implement the necessary tools and libraries.

When designing software for membrane computing, one has to describe precisely the P systems specification that is to be used. This task is hard if we need to handle families of P systems where the set of rules, the alphabet, the initial contents and even the membrane structure depend on the value assigned to some initial parameters. In existing software, several options have been implemented:

plain text files with a determined format, XML documents, graphical user interfaces, etc. As mentioned above, most of these solutions are adapted to specific models or to the specific purpose of the software.

In this paper we propose a programming language, called P-Lingua, whose programs define families of P systems in a parametric and modular way. After assigning values to the initial parameters, the compilation tool generates an XML document associated with the corresponding P system from the family, and furthermore it checks possible programming errors (both lexical/syntactical and semantical). Such documents can be integrated into other applications, thus guaranteeing interoperability. More precisely, in the simulators framework, the XML specification of a P system can be translated into an executable representation.

We present a practical application of P-Lingua in this paper. We give a simulator for recognizing P systems with active membranes that accepts as input an XML document generated by the compiler and that allows us to simulate a computation of the P system, obtaining the answer that the system outputs to its environment, plus a text file with a detailed step-by-step report of the computation.

The paper is structured as follows. In Section 2 several definitions and concepts are given for the sake of completeness of the paper. Section 3 introduces the P-Lingua programming language, and the syntax for P systems with active membranes is specified. In Section 4 we implement a solution to the SAT problem using P-Lingua. In Section 5 the compilation tool for the language is presented. Finally, Section 6 presents a simulator for recognizing P systems with active membranes. The paper ends with some conclusions and ideas for future work in Section 7.

2 Preliminaries

Polynomial time solutions to **NP**-complete problems in membrane computing are produced by trading time for space. This is inspired by the capability of cells to produce an exponential number of new membranes (new workspace) in polynomial time. Basically, there are two ways of producing new membranes in living cells: *mitosis* (membrane division) and *autopoiesis* (membrane creation). Both ways of generating new membranes have given rise to different variants of P systems: *P systems with active membranes*, where the new workspace is generated by membrane division, and *P systems with membrane creation*, where the new membranes are created from objects. Both models were proved to be computationally universal.

In this paper, we use the first variant mentioned above. Recall that a P system with active membranes is a construct of the form $\Pi = (O, H, \mu, \omega_1, \ldots, \omega_m, R)$, where $m \geq 1$ is the initial degree of the system; O is the alphabet of *objects*, and H is a finite set of *labels* for membranes; μ is a membrane structure, consisting of m membranes injectively labelled with elements of H, and $\omega_1, \ldots, \omega_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ; R is a finite set of *rules*, where each rule is of one of the following forms:

(a) $[a \rightarrow v]_h^\alpha$ where $h \in H$, $\alpha \in \{+, -, 0\}$ (electrical charges), $a \in O$ and v is a string over O describing a multiset of objects associated with membranes and depending on the label and the charge of the membranes (*object evolution rules*).

(b) $a\,[\,]_h^\alpha \rightarrow [b]_h^\beta$ where $h \in H$, $\alpha, \beta \in \{+, -, 0\}$, $a, b \in O$ (*send-in communication rules*). An object is introduced in the membrane, possibly modified, and the initial charge α is changed to β.

(c) $[a]_h^\alpha \rightarrow [\,]_h^\beta b$ where $h \in H$, $\alpha, \beta \in \{+, -, 0\}$, $a, b \in O$ (*send-out communication rules*). An object is sent out of the membrane, possibly modified, and the initial charge α is changed to β.

(d) $[a]_h^\alpha \rightarrow b$ where $h \in H$, $\alpha \in \{+, -, 0\}$, $a, b \in O$ (*dissolution rules*). A membrane with a specific charge is dissolved in reaction with a (possibly modified) object.

(e) $[a]_h^\alpha \rightarrow [b]_h^\beta\,[c]_h^\gamma$ where $h \in H$, $\alpha, \beta, \gamma \in \{+, -, 0\}$, $a, b, c \in O$ (*division rules*). A membrane is divided into two membranes. The objects inside the membrane are replicated, except for a, that may be modified in each membrane.

Rules are applied according to the following principles:

- Rules from (a) to (e) are used as is usual in the framework of membrane computing, i.e. in a maximal parallel way. In one step, each object in a membrane can only be used for one rule (non-deterministically chosen), but any object which can evolve by a rule must do it (with the restrictions indicated below).
- If a membrane is divided each object a in a membrane labelled with h and with charge α is divided into two membranes with label h, and one membrane has charge β and the second membrane has charge γ. The objects are replicated, but a can be modified in each membrane.
- If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external one. The skin is never dissolved.
- All the elements which are not involved in any of the operations to be applied remain unchanged.
- Rules associated with label h are used for all membranes with this label, irrespective of whether the membrane is an initial one or whether it was created.
- Rules (b) to (e) can not be applied simultaneously in a membrane in one computation step.

Recognizing P systems were introduced in [5], and are the natural framework to study and solve decision problems, since deciding whether an instance has an affirmative or negative answer is equivalent to deciding if a string belongs or not to the language associated with the problem.

In the literature, recognizing P systems are associated in a natural way with P systems with *input*. The data related to an instance of the decision problem has to be provided to the P system in order for it to compute the appropriate answer. This is done by codifying each instance as a multiset placed in an *input membrane*. The output of the computation, *yes* or *no*, is sent to the environment.

A *P system with input* is a tuple (Π, Σ, i_Π), where: (a) Π is a P system, with working alphabet Γ, with p membranes labelled by $1, \ldots, p$, and initial multisets $\omega_1, \ldots, \omega_p$ associated with them; (b) Σ is an (input) alphabet strictly contained in Γ; the initial multisets are over $\Gamma \setminus \Sigma$; and (c) i_Π is the label of a distinguished (input) membrane.

Let m be a multiset over Σ. The *initial configuration* of (Π, Σ, i_Π) with input m is $(\mu, \omega_1, \ldots, \omega_{i_\Pi} + m, \ldots, \omega_p)$.

A *recognizing P system* is a P system with input, (Π, Σ, i_Π), and with external output such that:

(a) The working alphabet contains two distinguished elements, *yes* and *no*.
(b) The system always halts.
(c) If C is a computation of Π, then either some object *yes* or some object *no* (but no both) must be released into the environment, and only in the last step of the computation.

We say that C is an accepting computation (respectively, rejecting computation) if the object *yes* (respectively, *no*) appears in the external environment associated with the corresponding halting configuration of C.

In this paper, we present a programming language to define P systems with active membranes. A programming language is an artificial language that can be used to control the behavior of a machine, particularly a computer, but it can be used also to define a model of a machine that can be translated into an executable representation by a simulation tool. The act of simulating something generally entails representing certain key characteristics or behaviours of some physical, or abstract, system. Do not confuse a simulation tool with an emulation tool: the second one duplicates the functions of one system by using a different system, so that the second system behaves like (and appears to be) the first system. With the actual technology, we can not emulate the functionality of a cellular machine by using a conventional computer to resolve **NP** problems in polynomial time, but we can simulate these cellular machines, not necessarily in polynomial time, in order to aid researchers.

Programming languages, like natural languages, are defined by syntactic and semantic rules which describe their structure and meaning respectively. Usually, they are asociated with compilation tools that are computer programs that translates text written in a programming language into another language. The original sequence is usually called the source code whereas the output called the object code. Commonly the output has a form suitable for being processed by other programs or for being executed by the computer, but it may be a human-readable text file. In this paper, we use an XML language-like object code. The Extensible Markup Language (XML) is a general-purpose specification for creating custom markup languages. It is classified as an extensible metalanguage because it allows its users to define their own elements. Its primary purpose is to facilitate the sharing of structured data across different information systems. The files written by using a specific XML language are called XML documents.

The P system computations are massively parallel. One of the most common programming methods to simulate real parallelism in a conventional computer with a single processor is to use multithreading. A thread in this sense is a thread of execution. Threads are a way for a program to fork (or split) itself into two or more simultaneously (or pseudo-simultaneously) running tasks. Multiple threads can be executed in parallel on a single computer. This multithreading generally occurs by time-division multiplexing where the processor switches between different threads. This context switching can happen so fast as to give the illusion of parallelism to an end-user. On a multiprocessor or multi-core system, threading can be achieved via multiprocessing, wherein different threads can literally run simultaneously on different processors or cores.

3 The P-Lingua programming language

3.1 Syntax for P systems with active membranes

What follows is the syntax of the language for P systems with active membranes (whose description can be found in [6] and [1] among others.)

Valid identifiers

We say that a sequence of characters forms a `valid identifier` if it does not begin with a numeric character and it is composed by characters from the following:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 _
```

Valid identifiers are widely used in the language: to define module names, parameters, indexes, membrane labels and alphabet objects.

The following text strings are reserved words in the language: `def, call, @mu, @ms, main, -->, #` and they cannot be used as valid identifiers.

Identifiers for electrical charges

In P-Lingua, we can consider electrical charges by using the `+` and `-` symbols for positive and negative charges respectively, and no one for neutral charge. It is worth mentioning that polarizationless P systems are included.

Data types

Two data types exist in P-Lingua:

- `Integer numbers:` We use 32 bits (signed) to store integer values, this allows a range from -2,147,483,648 to 2,147,483,647 for indexes and parameters.
- `Text strings:` These are valid identifiers that are used to define the alphabet objects and the membrane labels of a P system.

Variables

Two kind of variables are permitted in P-Lingua:

- `indexes`
- `Parameters`

Variables are used to store numeric values and their names are valid identifiers.

Numeric expressions

Numeric expressions can be written by using the `*` (multiplication), `/` (division), `%` (module), `+` (addition), `-` (subtraction) operators with integer numbers or variables, along with the use of parentheses.

Objects

The objects of the alphabet of a P system are written using valid identifiers, and the inclusion of sub-indexes is permitted. For example, $x_{i,2n+1}$ and Yes are written as `x{i,2*n+1}` and `Yes` respectively.

The multiplicity of an object is represented by using the `*` operator. For example, x_i^{2n+1} is written as `x{i}*(2*n+1)`.

Modules definition

Similarities between various solutions to **NP**-complete numerical problems by using families of recognizing P systems are discussed in [3]. Also, a cellular programming language is proposed based on libraries of subroutines. Using these ideas, a P-Lingua program consists of a set of programming modules that can be used more times by the same, or other, programs.

The syntax to define a module is the following.

```
def module_name(param1,..., paramN)
{
  sentence0;
  sentence1;
  ...
  sentenceM;
}
```

The name of a module, `module_name`, must be a valid and unique identifier.The parameters must be valid identifiers and cannot appear repeated. It is possible to define a module without parameters. Parameters have a numerical value that is assigned at the module call (see below).

All programs written in P-Lingua must contain a `main` module without parameters. The compiler will look for it when generating the XML file.

In P-Lingua there are sentences to define the membranes configuration of a P system, to specify multisets, to define rules and to make calls to other modules. Next, let us see how such sentences are written.

Module calls

In P-Lingua, modules are executed by using calls. The format of an sentence that calls a module for some concrete values of its parameters is given next:

```
call module_name(value1, ..., valueN);
```

where `value`i is an integer number or a variable.

Definition of the initial membrane structure of a P system

In order to define the initial membrane structure of a P system, the following sentence must be written:

```
@mu = expr;
```

where `expr` is a sequence of matching square brackets representing the membrane structure, including some identifiers that specify the label and the electrical charge of each membrane.

Examples:

1. $[[\,]^0_2]^0_1 \equiv$ `@mu = [[]'2]'1`
2. $[[\,]^0_b[\,]^-_c]^+_a \equiv$ `@mu = +[[]'b, -[]'c]'a`

Definition of multisets

Next sentence defines the initial multiset associated to the membrane labelled by `label`.

```
@ms(label) = list_of_objects;
```

where `label` is a valid identifier or a natural number that represents a label of the structure of membranes and `list_of_objects` is a comma-separated list of objects. The character `#` is used to represent the empty multiset.

Union of multisets

P-Lingua allows to define the union of two multisets (recall that the input multiset is "added" to the initial multiset of the input membrane) by using an sentence with the following format.

```
@ms(label) += list_of_objects;
```

Definition of rules

1. The format to define evolution rules of type $[a \rightarrow v]_h^\alpha$ is given next:

   α`[a --> v]'h`

2. The format to define send-in communication rules of type $a\,[\,]_h^\alpha \rightarrow [b]_h^\beta$ is given next:

   `a`α`[]'h -->`β`[b]`

3. The format to define send-out communication rules of type $[a]_h^\alpha \rightarrow b[\,]_h^\beta$ is given next:

   α`[a]'h -->` β`[]b`

4. The format to define division rules of type $[a\,]_h^\alpha \rightarrow [b]_h^\beta[c]_h^\gamma$ is given next:

   α`[a]'h -->`β`[b]`γ`[c]`

5. The format to define dissolution rules of type $[a]_h^\alpha \rightarrow b$ is given next:

   α`[a]'h --> b`

where:

- α, β and γ are identifiers for electrical charges.
- `a`, `b` and `c` are objects of the alphabet.
- `v` is a comma-separated list of objects that represents a multiset.
- `h` is a label.

Some examples:

- $[x_{i,1} \rightarrow r_{i,1}^4]_2^+ \equiv$ `+[x{i,1} --> r{i,1}*4]'2`
- $d_k[\,]_2^0 \rightarrow [d_{k+1}]_2^0 \equiv$ `d{k}[]'2 --> [d{k+1}]`
- $[d_k]_2^+ \rightarrow []_2^0 d_k \equiv$ `+[d{k}]'2 --> []d{k}`
- $[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- \equiv$ `[d{k}]'2 --> +[d{k}]-[d{k}]`
- $[a]_2^- \rightarrow b \equiv$ `-[a]'2 --> b`

Parametric sentences

In P-Lingua, it is possible to define parametric sentences by using the next format:

```
sentence : range1, ..., rangeN;
```

where `sentence` is a sentence of the language, or a sequence of sentences in brackets, and `range1, ..., rangeN` is a comma-separated list of ranges with the format:

```
min_value <= index <= max_value
```

where `min_value` and `max_value` are numeric expressions, integer numbers or variables, and `index` is a variable that can be used in the context of the sentence. It is possible to use the operator < instead of <=.

The sentence will be repeated for each possible values of each `index`.

Some examples of parametric sentences:

1. $[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- : 1 \leq k \leq n \equiv$
   `[d{k}]'2 --> +[d{k}]-[d{k}] : 1<= k <= n;`
2. $[x_{i,j} \rightarrow x_{i,j-1}]_2^+ : 1 \leq i \leq m, 2 \leq j \leq n \equiv$
   `+[x{i,j} --> x{i,j-1}]'2 : 1<=i<=m,2<=j<=n;`

Inclusion of comments

The programs in P-Lingua can be commented by writing phrases into the text strings `/*` and `*/`.

4 Implementation of a solution to SAT problem

SAT problem is the following: *Given a boolean formula in conjunctive normal form (CNF), to determine whether or not it is satisfiable, that is, whether there exists an assignment to its variables on which it evaluates to true.*

4.1 A solution to SAT

In this section, we present a solution to the **SAT** problem using recognizing P systems with active membranes, given by M.J. Pérez–Jiménez et al. [6].

For each $(m, n) \in \mathbb{N}^2$, we consider the P system

$$(\Pi(\langle m, n\rangle), \Sigma(m, n), i(m, n))$$

where

- $\Sigma(m, n) = \{x_{i,j}, \bar{x}_{i,j} : 1 \leq i \leq m, 1 \leq j \leq n\}$
- $i(m, n) = 2$
- $\Pi(\langle m, n\rangle) = (\Gamma(m, n), \{1, 2\}, [[\,]_2]_1, w_1, w_2, R)$, is defined as follows:
  - $\Gamma(m, n) = \Sigma(m, n) \cup \{c_k : 1 \leq k \leq m + 2\} \cup$
    $\{d_k : 1 \leq k \leq 3n + 2m + 3\} \cup$
    $\{r_{i,k} : 0 \leq i \leq m, 1 \leq k \leq m + 2\} \cup \{e, t\} \cup \{Yes, No\}$
  - $w_1 = \emptyset$
  - $w_2 = \{d_1\}$

– The set of rules, R, is given by:

$\{[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- : 1 \leq k \leq n\}$

$\{[x_{i,1} \rightarrow r_{i,1}]_2^+, [\bar{x}_{i,1} \rightarrow r_{i,1}]_2^- : 1 \leq i \leq m\}$

$\{[x_{i,1} \rightarrow \lambda]_2^-, [\bar{x}_{i,1} \rightarrow \lambda]_2^+ : 1 \leq i \leq m\}$

$\{[x_{i,j} \rightarrow x_{i,j-1}]_2^+, [x_{i,j} \rightarrow x_{i,j-1}]_2^- : 1 \leq i \leq m, 2 \leq j \leq n\}$

$\{[\bar{x}_{i,j} \rightarrow \bar{x}_{i,j-1}]_2^+, [\bar{x}_{i,j} \rightarrow \bar{x}_{i,j-1}]_2^- : 1 \leq i \leq m, 2 \leq j \leq n\}$

$\{[d_k]_2^+ \rightarrow []_2^0 d_k, [d_k]_2^- \rightarrow []_2^0 d_k : 1 \leq k \leq n\}$

$\{d_k[]_2^0 \rightarrow [d_{k+1}]_2^0 : 1 \leq k \leq n-1\}$

$\{[r_{i,k} \rightarrow r_{i,k+1}]_2^0 : 1 \leq i \leq m, 1 \leq k \leq 2n-1\}$

$\{[d_k \rightarrow d_{k+1}]_1^0 : n \leq k \leq 3n-3\}; [d_{3n-2} \rightarrow d_{3n-1}e]_1^0$

$e[]_2^0 \rightarrow [c_1]_2^+; [d_{3n-1} \rightarrow d_{3n}]_1^0$

$\{[d_k \rightarrow d_{k+1}]_1^0 : 3n \leq k \leq 3n+2m+2\}$

$[r_{1,2n}]_2^+ \rightarrow []_2^- r_{1,2n} \;;\; \{[r_{i,2n} \rightarrow r_{i-1,2n}]_2^- : 1 \leq i \leq m\}$

$r_{1,2n}[]_2^- \rightarrow [r_{0,2n}]_2^+$

$\{[c_k \rightarrow c_{k+1}]_2^- : 1 \leq k \leq m\}$

$[c_{m+1}]_2^+ \rightarrow []_2^+ c_{m+1} \;;\; [c_{m+1} \rightarrow c_{m+2}t]_1^0$

$[t]_1^0 \rightarrow []_1^+ t \;;\; [c_{m+2}]_1^+ \rightarrow []_1^- Yes \;;\; [d_{3n+2m+3}]_1^0 \rightarrow []_1^+ No$

4.2 Implementation

The following is the code of the program written in P-Lingua that encodes a solution to the SAT problem.

Objects of the form $\bar{x}_{i,j}$ are written as `nx{i,j}`.

```
/* Module that defines a family of recognizing P systems
   to solve the SAT problem */
def Sat(m,n)
{
 /* Initial configuration */
 @mu = [[]'2]'1;

 /* Initial multisets */
 @ms(2) = d{1};

 /* Set of rules */
 [d{k}]'2 --> +[d{k}]-[d{k}] : 1 <= k <= n;
```

```
 {
  +[x{i,1} --> r{i,1}]'2;
  -[nx{i,1} --> r{i,1}]'2;
  -[x{i,1} --> #]'2;
  +[nx{i,1} --> #]'2;
 } : 1 <= i <= m;

 {
  +[x{i,j} --> x{i,j-1}]'2;
  -[x{i,j} --> x{i,j-1}]'2;
  +[nx{i,j} --> nx{i,j-1}]'2;
  -[nx{i,j} --> nx{i,j-1}]'2;
 } : 1<=i<=m, 2<=j<=n;

 {
  +[d{k}]'2 --> []d{k};
  -[d{k}]'2 --> []d{k};
 } : 1<=k<=n;

 d{k}[]'2 --> [d{k+1}] : 1<=k<=n-1;
 [r{i,k} --> r{i,k+1}]'2 : 1<=i<=m, 1<=k<=2*n-1;
 [d{k} --> d{k+1}]'1 : n <= k<= 3*n-3;
 [d{3*n-2} --> d{3*n-1},e]'1;
 e[]'2 --> +[c{1}];
 [d{3*n-1} --> d{3*n}]'1;
 [d{k} --> d{k+1}]'1 : 3*n <= k <= 3*n+2*m+2;
 +[r{1,2*n}]'2 --> -[]r{1,2*n};
 -[r{i,2*n} --> r{i-1,2*n}]'2 : 1<= i <= m;
 r{1,2*n}-[]'2 --> +[r{0,2*n}];
 -[c{k} --> c{k+1}]'2 : 1<=k<=m;
 +[c{m+1}]'2 --> +[]c{m+1};
 [c{m+1} --> c{m+2},t]'1;
 [t]'1 --> +[]t;
 +[c{m+2}]'1 --> -[]Yes;
 [d{3*n+2*m+3}]'1 --> +[]No;

} /* End of Sat module */

/* Main module */
def main()
{
 /* Call to Sat module for m=4 and n=6 */
 call Sat(4,6);
 /* Expansion of the input multiset */
```

```
 @ms(2) += x{1,1}, nx{1,2}, nx{2,2}, x{2,3},
           nx{2,4}, x{3,5}, nx{4,6};
} /* End of main module */
```

The module `main` is instantiated with the formula

$$\varphi \equiv (x_1 + \overline{x}_2)(\overline{x}_2 + x_3 + \overline{x}_4)\, x_5\, \overline{x}_6$$

where $n = 6$, $m = 4$ and the input multiset: $x_{1,1}, \overline{x}_{1,2}, \overline{x}_{2,2}, x_{2,3}, \overline{x}_{2,4}, x_{3,5}, \overline{x}_{4,6}$.

5 The P-Lingua compiler

A compiler is a program that translates code written in some computer language to another language. We have developed a compiler that is able to translate programs written in P-Lingua into XML documents, after having assigned values to some initial parameters. Recall that a P-Lingua program can, in a flexible way, encode a family of P systems (with the help of some parameters), whereas the XML document generated by the compiler specifies only a single P system of the family. In this way, the applications do not need to process parametric systems, and hence their implementation is much easier.

The choice of the metalanguage XML is due to the fact that it is a broadly known standard, that has the following advantages:

- It is extensible. After having an XML specification designed, one can extend it by adding new labels, allowing in this way compatibility with earlier versions.
- The analyzer is a generic component, it is not necessary to create a new one for each XML specification. This avoids errors and speeds up the development of applications.
- The structure of the language is easy to understand and to process, facilitating compatibility with earlier versions.

5.1 An XML language for P systems with active membranes

The structure of the XML documents generated by the P-Lingua compiler for P systems with active membranes is as follows:

```
<?xml version="1.0"?>
<active_membrane_psystem version="1.0">

  <init_config>
   ...
  </init_config>
```

```
  <multisets>
   ...
  </multisets>

  <rules>
   ...
  </rules>

</active_membrane_psystem>
```

The main element is named `active_membrane_psystem`, and it has an attribute indicating the version of the specification. There are three internal elements:

- `init_config`: defines the membrane structure of the initial configuration.
- `multisets`: defines the initial multisets.
- `rules`: defines the set of rules.

Definition of the membrane structure of the initial configuration

Next, we describe the element `init_config` corresponding to the membrane structure $[[\,]_e^+[\,]_r^-]_s^0$.

```
<init_config>
  <membrane label="s" charge="0">
    <membrane label="e" charge="+1"/>
    <membrane label="r" charge="-1"/>
  </membrane>
</init_config>
```

`init_config` allows a recursive representation of a membrane structure. The element `membrane` has two attributes: `label`, which indicates the label of the membrane, and `charge`, which can take values `0`, `+1` or `-1` and indicates the membrane polarization.

Definition of initial multisets

Multisets of objects present in membranes are defined through the element `multisets`. Let us consider the following example: $w_e = e_0, g_1$, $w_s = z_1^3$ and $w_r = h_0, b_0$.

```
<multisets>
  <multiset label="e">
    <object name="e{0}" multiplicity="1"/>
```

```
    <object name="g{1}" multiplicity="1"/>
  </multiset>
  <multiset label="s">
    <object name="z{1}" multiplicity="3"/>
  </multiset>
  <multiset label="r">
    <object name="h{0}" multiplicity="1"/>
    <object name="b{0}" multiplicity="1"/>
  </multiset>
</multisets>
```

As it can be seen in the example, the element `multisets` is composed of several elements of type `multiset`, each of them having an attribute `label` indicating the label of the membrane where the multiset is contained. The objects present in the multiset are represented by elements of type `object` with two attributes: `name` indicates the symbol naming the object, and `multiplicity` indicates the multiplicity of the object in the multiset.

Definition of the set of rules

Let us consider the following set of rules:

- $[c_9 \rightarrow c_{10}t]_1^0$
- $[r_{1,16}]_2^+ \rightarrow [\,]_2^- r_{1,16}$
- $r_{1,16}[\,]_2^- \rightarrow [r_{0,16}]_2^+$
- $[d_0]_2^0 \rightarrow [d_0]_2^+ [d_0]_2^-$
- $[a]_e^0 \rightarrow b$

The element `rules` is described as follows:

```
<rules>
  <evolution_rule label=1" charge="0">
    <left_hand_rule object="c{9}"/>
    <right_hand_rule object="c{10}" multiplicity="1"/>
    <right_hand_rule object="t" multiplicity="1"/>
  </evolution_rule>
  <send_out_rule label="2" charge="+1">
    <left_hand_rule object="r{1,16}"/>
    <right_hand_rule object="r{1,16}" charge="-1"/>
  </send_out_rule>
  <send_in_rule label="2" charge="-1">
    <left_hand_rule object="r{1,16}"/>
    <right_hand_rule object="r{0,16}" charge="+1"/>
```

```
    </send_in_rule>
    <division_rule label="2" charge="0">
      <left_hand_rule object="d{0}"/>
      <right_hand_rule object="d{0}" charge="+1"/>
      <right_hand_rule object="d{0}" charge="-1"/>
    </division_rule>
    <dissolution_rule label="e" charge="0">
      <left_hand_rule object="a"/>
      <right_hand_rule object="b"/>
    </dissolution_rule>
  </rules>
```

Within the element `rules` we can find five different types of elements: `evolution_rule`, `send_in_rule`, `send_out_rule`, `division_rule` and `dissolution_rule`. All of them contain two attributes: `label` and `charge`, indicating the label and polarization of the membranes to which the rule can be applied.

Besides, there exists an internal element called `left_hand_rule` with an attribute called `object` containing the name of the object that triggers the rule.

For the case of evolution rules, the compiler generates one or more elements of type `right_hand_rule`, each of them having two attributes `object` and `multiplicity` expressing the name of the object produced by the rule, and the number of copies obtained.

Communication rules have only one element `right_hand_rule` with the name of the resulting object and the polarization that the membrane gets after applying the rule.

For division rules, there are two elements `right_hand_rule`, indicating the objects obtained in the two resulting membranes, as well as their respective polarizations.

Finally, for dissolution rules, only one element `right_hand_rule` showing the name of the object that is obtained.

5.2 The compilation tool

The P-Lingua compiler (version 1.0) and its source code can be freely downloaded from the *software* section in the website of the Research Group on Natural Computing [11]. The compiler is under GPL license [7] and is written in Java [8] using the lexical and syntactical analyzers provided by JavaCC [9]. The minimum system requirements are having a Java virtual machine (JVM) version 1.6.0 running in a Pentium III computer.

The compilation tool is a program that may be exectuted from the command line as follows:

```
plingua input_file -xml output_file [-v verbosity_level] [-h]
```

The text file `input_file` contains the program (written in P-Lingua) that we want to be compiled, and `output_file` is the name of the XML file that is generated. Optional arguments are in brackets: `verbosity_level` is a number between 0 and 5 indicating the level of detail of the messages shown during the compilation process, and the option `-h` displays some help information.

6 A simulator for recognizing P systems with active membranes

As a first practical application of the P-Lingua programming language, we have implemented a simulator for recognizing P systems with active membranes that takes as input an XML document generated by the P-Lingua compiler and runs one of the possible computations that the P system may follow, obtaining the answer that the system outputs to its environment, plus a text file with a detailed step-by-step report of the computation.

This simulator is again a Java program under GPL license that can be freely downloaded from the *software* section in the web of the Research Group on Natural Computing [11]. The system requirements are the same as in the case of the P-Lingua compiler.

The simulator is launched from the command line as follows:

```
plingua_sim input_xml [-o output_file]
```

where `input_xml` is an XML document formatted as discussed in this paper, and `output_file` is the name of the file where the report about the simulated computation will be saved.

6.1 Simulation of a solution to SAT problem

We now show an execution of the simulator running on the XML document obtained after compiling the P-Lingua program described in Section 4.2. The results have been obtained on an AMD Sempron machine, at 2.8 Ghz and with 512Mb of RAM memory.

The command used to execute the simulation is:

```
plingua_sim sat.xml -o info.txt
```

The simulation ends when no more rules can be applied, and then the following information is displayed:

```
Environment: t, Yes
Steps: 41
Time: 1.971 s.
Halting configuration (No rule can be selected to be executed
in the next step)
```

Thus, the computation of the P system lasted 41 transition steps, and it took 1,971 seconds to simulate it until reaching a halting configuration (recall that we are simulating a parallel device on a sequential computer).

The file `info.txt` keeps detailed information about each configuration of the simulated computation. More precisely, the multisets and polarizations of all the membranes are listed, as well as the rules selected for execution at each transition step. The configurations are numbered (starting at 0), to keep track of the step of the computation that is being simulated. Some information about the CPU time is shown for each step, and the number of rules of each type that is executed. As an example, we give the information generated for the first two configurations.

```
### MEMBRANE ID: 1, Label: 2, Charge: 0
    Multiset: nx{1, 2}, d{1}, x{3, 5}, nx{2, 4}, nx{2, 2},
              nx{4, 6}, x{2, 3}, x{1, 1}
    Parent Membrane ID: 0
    Rules Selected:
    1*DIVISION RULE: [d{1}]'2 --> +[d{1}] -[d{1}]

@@@ SKIN MEMBRANE ID: 0, Label: 1, Charge: 0
    Multiset: #
    Internal membranes count: 1

Configuration: 0
Time: 0.0 s.
1 division rule(s) selected to be executed in the step 1
***********************************************
### MEMBRANE ID: 1, Label: 2, Charge: +
    Multiset: nx{1, 2}, d{1}, x{3, 5}, nx{2, 4}, nx{2, 2},
              nx{4, 6}, x{2, 3}, x{1, 1}
    Parent Membrane ID: 0
    Rules Selected:
    1*EVOLUTION RULE: +[nx{2, 2} --> nx{2, 1}]'2
    1*EVOLUTION RULE: +[nx{1, 2} --> nx{1, 1}]'2
    1*EVOLUTION RULE: +[x{3, 5} --> x{3, 4}]'2
    1*EVOLUTION RULE: +[x{1, 1} --> r{1, 1}]'2
    1*EVOLUTION RULE: +[nx{2, 4} --> nx{2, 3}]'2
    1*EVOLUTION RULE: +[nx{4, 6} --> nx{4, 5}]'2
    1*EVOLUTION RULE: +[x{2, 3} --> x{2, 2}]'2
    1*SEND-OUT RULE: +[d{1}]'2 --> []d{1}

### MEMBRANE ID: 2, Label: 2, Charge: -
    Multiset: nx{1, 2}, d{1}, nx{2, 4}, x{3, 5}, nx{2, 2},
              x{2, 3}, nx{4, 6}, x{1, 1}
    Parent Membrane ID: 0
```

```
    Rules Selected:
    1*EVOLUTION RULE: -[nx{2, 4} --> nx{2, 3}]'2
    1*EVOLUTION RULE: -[nx{2, 2} --> nx{2, 1}]'2
    1*EVOLUTION RULE: -[nx{4, 6} --> nx{4, 5}]'2
    1*EVOLUTION RULE: -[x{1, 1} --> #]'2
    1*EVOLUTION RULE: -[x{2, 3} --> x{2, 2}]'2
    1*EVOLUTION RULE: -[nx{1, 2} --> nx{1, 1}]'2
    1*EVOLUTION RULE: -[x{3, 5} --> x{3, 4}]'2
    1*SEND-OUT RULE: -[d{1}]'2 --> []d{1}

@@@ SKIN MEMBRANE ID: 0, Label: 1, Charge: 0
    Multiset: #
    Internal membranes count: 2

Configuration: 1
Time: 0.025 s.
14 evolution rule(s) selected to be executed in the step 2
2 send-out rule(s) selected to be executed in the step 2
***********************************************
```

After simulating 41 transition steps, the halting configuration is described as follows:

```
### MEMBRANE ID: 1, Label: 2, Charge: +
    Multiset: r{0, 12}*3, c{4}
    Parent Membrane ID: 0

### MEMBRANE ID: 2, Label: 2, Charge: +
    Multiset: c{1}, r{2, 12}, r{3, 12}
    Parent Membrane ID: 0

### MEMBRANE ID: 3, Label: 2, Charge: +
    Multiset: r{0, 12}*5, c{4}
    Parent Membrane ID: 0

### MEMBRANE ID: 4, Label: 2, Charge: +
    Multiset: r{0, 12}*4, c{4}
    Parent Membrane ID: 0

### MEMBRANE ID: 5, Label: 2, Charge: +
    Multiset: r{0, 12}, r{2, 12}, c{2}
    Parent Membrane ID: 0

### MEMBRANE ID: 6, Label: 2, Charge: +
    Multiset: c{1}, r{3, 12}
```

```
    Parent Membrane ID: 0

### MEMBRANE ID: 7, Label: 2, Charge: +
    Multiset: r{0, 12}*4, c{4}
    Parent Membrane ID: 0
```

⋮

```
@@@ SKIN MEMBRANE ID: 0, Label: 1, Charge: -
    Multiset: t*10, d{29}*64, c{6}*10
    Internal membranes count: 64

~~~ENVIRONMENT: t, Yes

Configuration 41
Time: 1.971 s.
Halting configuration (No rule can be selected to be
executed in the next step)

*********************************************
```

Note that there are 64 different membranes labelled by 2 in this configuration, although for the sake of simplicity we show only seven of them.

7 Conclusions and future work

In this paper we have presented the first programming language for membrane computing, *P-Lingua*, together with a compiler that generates XML documents, and a simulator for a class of P systems called recognizing P systems with active membranes.

Using a programming language to define cellular machines is a new concept in the development of applications for membrane computing that leads to a standardization with the following advantages:

- Users (researchers) can define cellular machines in a modular and parametric way by using an easy-to-learn programming language.
- It is possible to define libraries of modules that can be shared among researchers to facilitate the design of new programs.
- This method to define P systems is decoupled from its applications and the same P-Lingua programs can be used in different software enviroments.
- By using compiling tools, the P-Lingua programs are translated to other file formats that can be interpreted by a large number of different applications.

The first version of P-Lingua is presented for P systems with active membranes. In forthcoming versions we intend to generalize the language so that other types of

cellular devices can be also specified, for instance transition P systems and tissue P systems.

Currently, the compiler is an application that is executed from the command line, but the possibility of a graphical programming environment remains open.

We have chosen an XML language as the output format because of the reasons exposed above. However, we are aware that for some applications it is not the most suitable format, due to the fact that XML does not include any method for compressing data, and therefore the text files can eventually become too large, which is a clear disadvantage for applications running on networks of processors. It would be convenient to modify the compiler so that it generates a larger variety of output formats, of special interest are compressed binary files or executable code (either in C or Java).

It is important to recall that the simulator presented here is designed to run in a conventional computer, having limited resources (RAM, CPU), and this leads to a bound on the size of the instances of **NP**-complete problems whose solutions can be successfully simulated. Moreover, conventional computers are not massively parallel devices, and therefore it seems that the inherent parallelism of P systems must be simulated by means of multithreading techniques.

These shortcomings lead us to the possibility of implementing a distributed simulator running on a network or cluster of processors, where the need of resources arising during the computation could be solved by adding further nodes to the network, thus moving towards massive parallelism.

Acknowledgement

The authors acknowledge the valuable assistance given by Damien Woods who helped us to write this paper.

The authors also wish to acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, as well as the support of the project of excellence TIC-581 of the Junta de Andalucía.

References

1. A. Alhazov, M.J. Pérez–Jiménez. Uniform solution of QSAT using polarizationless active membranes. In J. Durand–Lose and M. Margenstern (eds.) *Machines, Computations, and Universality. Lecture Notes in Computer Science*, 4664 (2007), pp. 122–133.
2. M. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez. Available membrane computing software. In G. Ciobau, Gh. Păun, M.J. Péréz–Jiménez (eds.) *Applications of Membrane Computing, Natural Computing Series*, Springer–Verlag, Berlin, 15 (2006), pp. 411–436.
3. M.A. Gutiérrez, M.J. Pérez–Jiménez, A. Riscos–Núñez. Towards a programming language in cellular computing. *Electronic Notes in Theoretical Computer Science*, Elsevier B.V., 123 (2005), pp. 93–110.

4. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), pp. 108–143.
5. M.J. Pérez-Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini. Complexity classes in cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265-285.
6. M.J. Pérez–Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini. A polynomial complexity class in P systems using membrane division. In E. Csuhaj Varjú, C. Kintala, D. Wotschke, G. Vaszil (eds.), *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003*, Computer and Automation Research Institute of the Hungarian Academy of Sciences, Budapest, (2003), pp. 284–294.
7. The GNU General Public License: `http://www.gnu.org/copyleft/gpl.html`
8. Java web page: `http://www.java.com/`
9. JavaCC web page: `https://javacc.dev.java.net/`
10. P systems web page: `http://ppage.psystems.eu/`
11. Research Group on Natural Computing web page: `http://www.gcn.us.es/`

No Cycles in Compartments. Starting from Conformon-P Systems

Pierluigi Frisco[1], Gheorghe Păun[2]

[1] School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, EH14 4AS, UK
E-mail: pier@macs.hw.ac.uk

[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
and
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mails: george.paun@imar.ro, gpaun@us.es

Summary. Starting from proofs of results about the computing power of conformon-P systems, we infer several results about the power of certain classes of tissue-like P systems with (cooperative) rewriting rules used in an asynchronous way, without cycles in compartments. This last feature is related to an important restriction appearing when dealing with lab implementations of P systems, that of avoiding local evolution loops of objects.

1 Introduction

This note addresses a technical issue which appeared in the framework of the recent attempt to implement a P system in biochemical terms, at Technion institute, Haifa, Israel, namely of avoiding cyclical evolution of chemicals in any compartment of the system – see a more precise description of the problem in [9]. Here we consider a class of tissue-like P systems, namely as introduced in [16], with rewriting rules present in membranes, and with target indications of the forms *here, go* associated with the "products of reactions": rules of the form $u \to v$, where u and v are multisets of objects and the objects in v have associated target indications *here, go* (actually, *here* is omitted) indicating that the respective object remains in the same compartment or it has to go to any of the adjacent compartments, non-deterministically choosing the destination. We also consider an evolution-communication (EC) version of these systems, following the ideas of [1], i.e., using evolution rules without target indications and using separate communication rules (of the form (a, go), with the obvious meaning: object

a is communicated to any of the adjacent membranes). In order to transfer in a direct way to these systems results from conformon-P systems area, we add to the definition in [16] several "non-standard" ingredients: we work asynchronously (in any step, in any compartment, a rule may be used or not), maybe with a priority relation among rules, of a global type (in each compartment, evolution rules have priority over communication rules: if an object can evolve and, at the same time, communicated, an evolution rule is applied first), an acknowledging membrane (the computation stops when any object is sent to this membrane, which is empty in the beginning of the computation). The number of membranes we use is arbitrary (rather high, if we take into account the number of membranes used in conformon-P systems simulating register machines), but, on the good side, the evolution rules we need to simulate a conformon-P system are of a very restrictive form: each of the multisets u, v from a rule $u \to v$ has exactly two objects.

Although, for the sake of readability, we recall here the definitions of conformon-P systems and of P systems with a graph structure, we do not enter into details, and we assume the reader to be familiar with basic elements of membrane computing. However, we indicate a series of papers related to conformons. This concept was introduced independently in [10] and [17]. Following the definition given in [10] conformons and conformon-like entities have been classified into 10 families according to their biological functions [12]. To know more about the Bhopalator refer to [11, 13]. The term *conformon* was adopted in [14, 15] where the authors started to develop a quantum mechanical theory based on this concept. Conformon-P systems have been introduced in [3] and later studied, among others, in [4, 6]. Conformon-P systems have also been successfully used as a platform to model biological process. The interested reader can refer to [8, 2, 7].

2 Basic Definitions

Let V be an alphabet (a finite set of abstract symbols), and $\mathbb{N}$ be the set of natural numbers, including 0. A multiset over V is a function $M : V \longrightarrow \mathbb{N} \cup \{+\infty\}$. The support of M (the set of elements $a \in V$ for which $M(a) > 0$) is denoted by $supp(M)$ and the cardinality of M (the sum of multiplicities of all elements in $supp(M)$) is denoted by $|M|$.

2.1 Conformon-P Systems

In what follows, a *conformon* is an element of $V \times \mathbb{N}$, denoted by $[a, n]$. We refer to a as the *name* of the conformon $[a, n]$ and to n as its *value*.

Two conformons can interact according to an *interaction rule*. An interaction rule is of the form $a \xrightarrow{e} b$, where $a, b \in V$ and $e \in \mathbb{N}$, and it says that a conformon with name a can give e from its value to the value of a conformon having name b. If, for instance, there are conformons $[a, 5]$ and $[b, 9]$ and the rule $a \xrightarrow{3} b$, one application this rule leads to $[a, 2]$ and $[b, 12]$. As here we consider that the value

of a conformon cannot be a negative number, the rule $a \stackrel{3}{\rightarrow} b$ cannot be applied to $[a, 2]$.

Each membrane present in a conformon-P system has associated a label, different from the labels of other membranes. These membranes are placed in the nodes of a directed graph, hence they are connected in a unidirectionally way. Each connection has associated a *predicate*, which is an element of the set $pred(\mathbb{N}) = \{\geq n, \leq n \mid n \in \mathbb{N}\}$. If, for instance, there are two compartments (with labels) m_1 and m_2 and there is an connection from m_1 to m_2 having predicate ≥ 4, then conformons having value greater than or equal to 4 can pass from m_1 to m_2.

A *conformon-P system* is a construct

$$\Pi = (V, \mu, \omega_z, ack, L_1, \ldots, L_m, R_1, \ldots, R_m),$$

where:

V is a finite alphabet;
$\mu = (Q, E)$ is a *directed labelled graph* underlying Π, where
  $Q = \{1, \ldots, m\}$ is the set of *membranes* (we also say *compartments*) of Π;
  $E \subseteq Q \times Q \times pred(\mathbb{N})$ defines directed labelled *edges* between vertices, indicated by $(i, j, pred)$, $i, j \in Q, i \neq j$, where $pred \in pred(\mathbb{N})$ is a *predicate*;
ω_z with $\omega \in \{in, out\}$ and $z \in Q$ indicates whether Π is an accepting ($\omega = in$) or generating ($\omega = out$) device; the compartment z contains the input or output, respectively;
$ack \in Q$ indicates the *acknowledging* compartment;
$L_i : (V \times \mathbb{N}) \rightarrow \mathbb{N} \cup \{+\infty\}$, $i \in Q$, are multisets of conformons initially associated with the vertices in Q;
R_i, $i \in Q$, are finite sets of interaction rules associated with the vertices in Q, with $supp(L_{ack}) = \emptyset$.

Let M_i and R_i be the multiset of conformons and the set of rules, respectively, associated with the compartment $i \in Q$. Two conformons present in compartment i can interact according to a rule in R_i such that the multiset of conformons M_i changes into M_i'. If, for instance, $[a, p], [b, q] \in M_i$, $a \stackrel{e}{\rightarrow} b \in R_i$ and $p \geq e$, then $M_i' = (M_i - \{[a, p], [b, q]\}) \cup \{[a, p - e], [b, q + e]\}$.

A conformon $[a, p]$ present in compartment i can *pass* to compartment j if $(i, j, pred) \in E$ and $pred(p)$ holds. This passage changes the multisets of conformons M_i and M_j into M_i' and M_j', respectively, such that $M_i' = M_i - \{[a, p]\}$ and $M_j' = M_j \cup \{[a, p]\}$.

At the moment we do not assume any requirement (such as maximal parallelism, priorities, etc.) on the application of operations. If a conformon can pass to another compartment or interact with another conformon according to an interaction rule, then one of the two operations or none of them is non-deterministically chosen.

The possibility to carry out one of the two allowed operations in a compartment or none of them lets conformon-P systems to be non-deterministic. Non-determinism can also arise from the configurations of a conformon-P system if in a compartment a conformon can interact with more than one conformon and also from the graph underlying Π if a compartment has edges with the same predicate going to different compartments.

A *configuration* of Π is an m-tuple $(M_1, \ldots, M_m)$ of multisets over $V \times \mathbb{N}$. The m-tuple $(L_1, \ldots, L_m)$, is called *initial configuration* (remember that $supp(L_{ack}) = \emptyset$, so in the initial configuration the acknowledging compartment does not contain any conformon) while any configuration having $supp(M_{ack}) \neq \emptyset$ is called *final configuration*. In a final configuration no operation is performed even if it could.

For two configurations $(M_1, \ldots, M_m), (M'_1, \ldots, M'_m)$ of Π we write $(M_1, \ldots, M_m) \Rightarrow (M'_1, \ldots, M'_m)$ indicating a *transition* from $(M_1, \ldots, M_m)$ to $(M'_1, \ldots, M'_m)$, that is, the application of one operation to at least one conformon. In other words, in any configuration in which $supp(L_{ack}) = \emptyset$ any conformon present in a compartment can either interact with another conformon present in the same compartment or pass to another compartment or remain in the same compartment unchanged. If no operation is applied to a multiset M_i, then $M'_i = M_i$. The reflexive and transitive closure of $\Rightarrow$ is indicated by $\Rightarrow^*$.

A *computation* is a finite sequence of transitions between configurations of a system Π starting from $(L_1, \ldots, L_m)$.

In case Π is an accepting device ($\omega = in$), then the input is given by the number of conformons (counted with their multiplicity) present in L_z. The input is accepted by Π if it reaches a configuration in which any conformon is present in ack, halting in this way the computation.

Formally:

$$N(\Pi) = \{|L_z| \mid (L_1, \ldots, L_m) \Rightarrow^* (M'_1, \ldots, M'_m) \Rightarrow (M_1, \ldots, M_m), \\ supp(M'_{ack}) = \emptyset, supp(M_{ack}) \neq \emptyset\}.$$

In case Π is a generating device ($\omega = out$), then $supp(L_z) = \emptyset$. The result of a computation is given by M_z when any conformon is present in ack. When this happens the computation is halted and the number of conformons (counted with their multiplicity) present in M_z defines the *number generated* by Π.

Formally:

$$N(\Pi) = \{|M_z| \mid (L_1, \ldots, L_m) \Rightarrow^* (M'_1, \ldots, M'_m) \Rightarrow (M_1, \ldots, M_m), \\ supp(M'_{ack}) = \emptyset, supp(M_{ack}) \neq \emptyset\}.$$

In the conformon-P systems area, in general one uses graphical representations instead of formal definitions in order to specify systems appearing in examples or proofs. We recall now some conventions used in these representations – details can be found in the papers mentioned in the end of Introduction.

Membranes/compartments are represented by labelled ovals, having inside the associated conformons and interaction rules. Conformons present in the initial

configuration of a system are written in **bold** inside a membrane while the ones written in normal font are present in that compartment in one of the possible configurations of the system. A slash (/) between values in a conformon indicates that a conformon can have any of the indicated values. The multiplicity is indicated only for conformons which appear in more than one copy. Directed edges between compartments are represented as arrows with their predicate indicated close to them. Several edges connecting two compartments are depicted as just one edge with different predicates separated by a slash (/). For instance, Figure 1 presents a conformon-P system which accepts any positive even number (the input membrane is the one with label 1 and the acknowledging one is membrane 11).

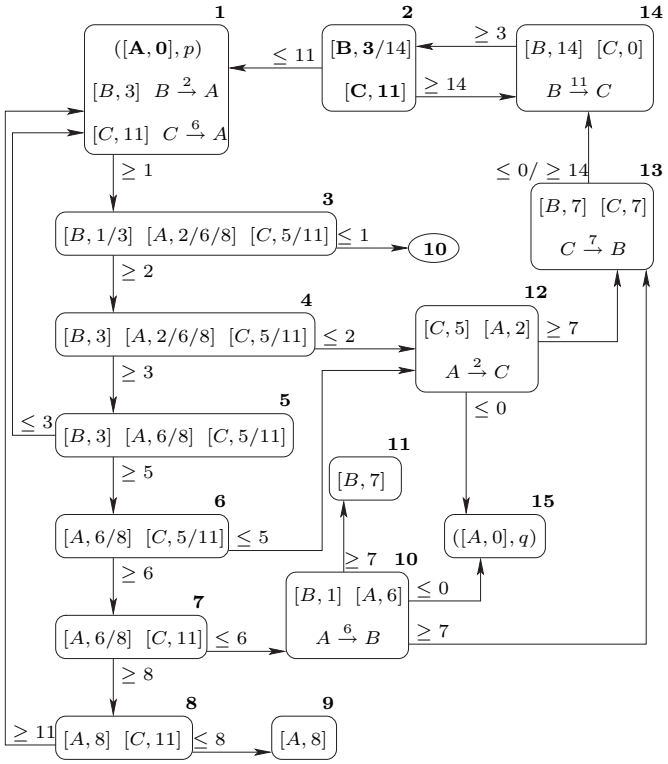


Fig. 1. A conformon-P system accepting even numbers.

In proofs there appear large conformon-P systems, that is why it is useful to consider *modules* which are sort of shortcuts of graphical representations. Such modules are explained in detail in several papers, e.g., in [3].

The basic modules are the *splitter* (it selects conformons depending on their values; specifically, when conformons of type $[a, p_i]$, $1 \leq i \leq h$, are present in a

given compartment, they can pass to specific different compartments depending on values p_i) and the *separator* (it selects conformons depending on their name; specifically, when conformons of type $[a_i, p]$, $1 \leq i \leq h$, are present in a compartment, they can pass to specific different compartments depending on a_i).

In the pictorial representations of conformon-P systems the modules are indicated by tick ovals, linked by arrows marked with predicates, which are of the form $= n_i$ in the case of splitters and of the form $[a, p_i]$ in the case of separators; usual membranes and arrows marked with predicates can be interleaved with modules. For instance, in Figure 2 we give a version of the system represented in Figure 1 where a splitter is also involved.

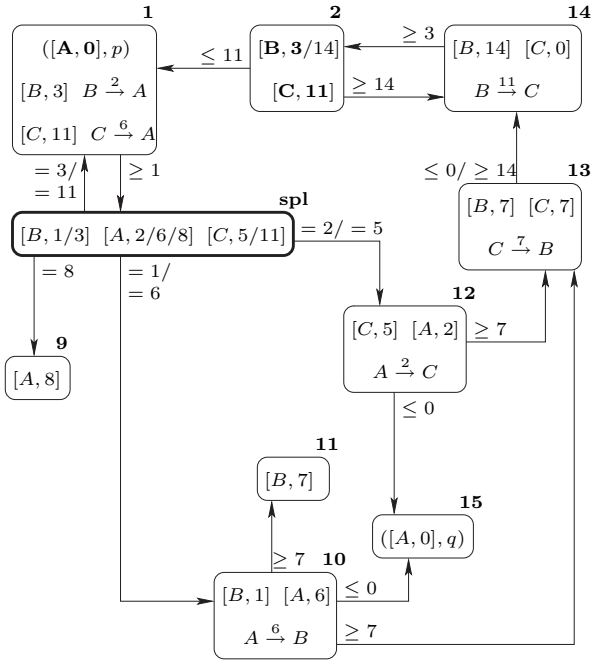


Fig. 2. The conformon-P system with a splitter associated to the system in Figure 1.

2.2 Asynchronous Tissue-like P Systems

We introduce the P systems of the form we have described in the Introduction, with a series of ingredients as presented before for conformon-P systems. Because we work only with asynchronous systems, from now on we omit mentioning this feature.

An *EC tissue-like P system* of degree m is a tuple

$$\Pi = (V, \mu, \omega_z, ack, L_1, \ldots, L_m, R_1, \ldots, R_m, P_1, \ldots, P_m),$$

where:

V is a finite alphabet whose elements are called *objects*;
$\mu = (Q, E)$ is a graph indicating the underlying compartment structure of Π, where
  $Q = \{1, \ldots, m\}$ is the set of *membranes/compartments*;
  $E \subseteq Q \times Q$ is the set of directed *edges* between compartments;
ω_z with $\omega \in \{in, out\}$ and $z \in Q$ indicates if Π is an accepting ($\omega = in$) or generating ($\omega = out$) device; the compartment z contains the input or output, respectively;
$ack \in Q$ indicates the *acknowledging* compartment;
$L_i : V \to \mathbb{N} \cup \{+\infty\}$, $1 \leq i \leq m$, are multisets of objects in V, with $supp(L_{ack}) = \emptyset$;
$R_i, 1 \leq i \leq m$, are sets of *evolution* rules of the form $ab \to cd$ with $a, b, c, d \in V$;
$P_i, 1 \leq i \leq m$, are sets of *communication* rules of the form (a, go) with $a \in V$.

A tissue-like P system is *cycle-free* if $ab \to cd \in R_i$ implies that $cd \to ab$ does not belong to R_i (with some abuse of notation we represent multisets by strings and all their permutations).

A *configuration* of Π is an m-tuple $(M_1, \ldots, M_m)$ of multisets over V. The m-tuple $(L_1, \ldots, L_m)$, is called *initial configuration* (in the initial configuration the acknowledge compartment does not contain any object) while any configuration having $supp(M_{ack}) \neq \emptyset$ is called *final configuration*. In a final configuration no operation is performed even if it could.

For two configurations $(M_1, \ldots, M_m), (M'_1, \ldots, M'_m)$ of Π we write $(M_1, \ldots, M_m) \Rightarrow (M'_1, \ldots, M'_m)$ indicating a *transition* from $(M_1, \ldots, M_m)$ to $(M'_1, \ldots, M'_m)$, that is, the application of one rule in a compartment according to the following. If $a, b \in M_i$ and $ab \to cd \in R_i$, then $M'_i = M_i - \{a, b\} \cup \{c, d\}$. If $a \in M_i$ and $(a, go) \in P_i$, then $M'_i = M_i - \{a\}$, $M'_j = M_j \cup \{a\}$ if $(i, j) \in E$. If no rule is applied to a multiset M_i, then $M'_i = M'_i$. The reflexive and transitive closure of $\Rightarrow$ is indicated by $\Rightarrow^*$. If in a configuration a symbol can be subject to more than one rule, then one of them is non-deterministically applied.

A *computation* is a finite sequence of transitions between configurations of the system Π starting from $(L_1, \ldots, L_m)$.

In case Π is an accepting device ($\omega = in$), then the input is given by the number of symbols (counted with their multiplicity) present in L_z. The input is accepted by Π if it reaches a configuration in which any conformon is present in ack, halting in this was the computation.

Formally,

$$N(\Pi) = \{|L_z| \mid (L_1, \ldots, L_m) \Rightarrow^* (M'_1, \ldots, M'_m) \Rightarrow (M_1, \ldots, M_m), \\ supp(M'_{ack}) = \emptyset, supp(M_{ack}) \neq \emptyset\}.$$

In case Π is a generating device ($\omega = out$), then $supp(L_z) = \emptyset$. The result of a computation is given by M_z when any symbol is present in ack. When this

happens the computation is halted and the number of symbols (counted with their multiplicity) present in M_z defines the *number generated* by Π.

Formally,

$$N(\Pi) = \{|M_z| \mid (L_1, \ldots, L_m) \Rightarrow^* (M'_1, \ldots, M'_m) \Rightarrow (M_1, \ldots, M_m), \\ supp(M'_{ack}) = \emptyset, supp(M_{ack}) \neq \emptyset\}.$$

As usual in P systems (i.e., without separating evolution from communication), we avoid rules of the form (a, go) and associate target indication directly to evolution rules: an object which has to be communicated will appear in the right hand side of a rule paired with *go* (the objects without such a pair remain in the same membrane). Note the important detail that this time the communication of an object c appearing in the form (c, go) in a rule must be done immediately, this does not mean application of a rule, but it is just part of using the evolution rule. This is a difference with respect to conformon-P systems and to EC tissue-like P systems, but in the proofs below we will not have to take care of this aspect: communication will be done by evolution rules of the form $a \to (a, go)$ which are directly associated with communication rules of the form (a, go).

3 Computing with Conformon-P Systems

We recall now some results concerning the computing power of conformon-P systems. Proofs can be found, e.g., in [3].

A conformon-P systems is called *value-restricted* (in short, VR) if in its initial configuration all conformons present in an unbounded number of copies have value 0. In this way, the total value of conformons present in the system at any step of a computation is finite.

Theorem 1. *The family of sets of numbers generated by VR conformon-P systems coincides with the family of sets of numbers generated by partially blind register machines.*

The conformon-P system which can simulate a partially blind register machine is based on the construction indicated in Figure 3. We recall it because later we will point out some basic features of this construction useful in inferring results about (asynchronous) tissue-like P systems.

From Theorem 2 in [5] we know that if in the conformon-P system described in the previous theorem either priorities, maximal concurrency, or maximal parallelism are added, then the resulting systems are computationally complete.

Theorem 2. *The family of sets of numbers generated by VR conformon-P systems where evolution has priority on communication (if a conformon can be subject of an interaction rule and it can also pass to another membrane, then the interaction should be done) coincides with the family of sets of numbers generated by register machines (hence with the family of Turing computable sets of numbers).*

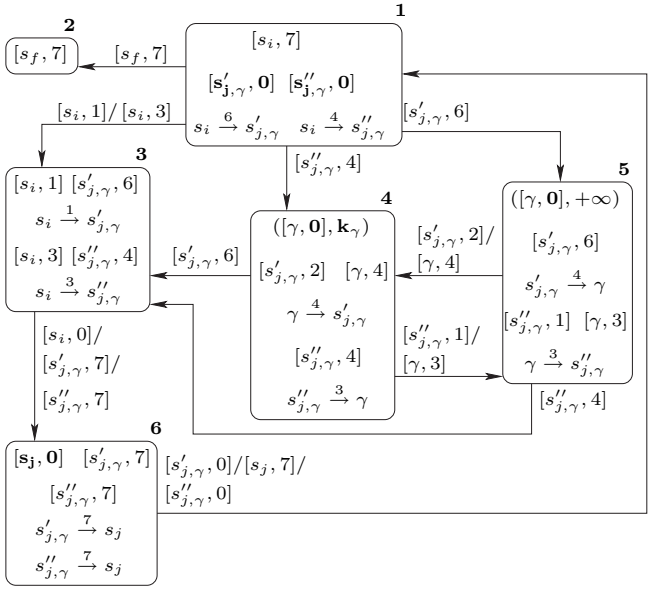


Fig. 3. The conformon-P system related to Theorem 1.

Also for this case we recall – in Figure 4 – the construction used in proving that a conformon-P system with priority as above can simulate a register machine.

4 From Conformon- to Tissue-like P Systems

First, let us point out a direct passage from conformon-P systems to EC tissue-like P systems.

Theorem 3. *Given any VR conformon-P system* $\Pi = (V, \mu, \omega_z, ack, L_1, \ldots, L_m, R_1, \ldots, R_m)$*, we can construct an EC tissue-like P system* $\Pi' = (V', \mu', \omega_z, ack, L'_1, \ldots, L'_m, R'_1, \ldots, R'_m, P'_1, \ldots, P'_m)$ *such that* $N(\Pi') = N(\Pi)$.

Proof. Consider a conformon-P system Π as above, with $\mu = (Q, E)$; denote by S the sum of the values of the conformons in Π. We construct the tissue-like P system Π' with:

$V' = \{a_p \mid a \in V, 0 \leq p \leq S\}$;
$\mu' = (Q, E')$ with $(i, j) \in E'$ for each $(i, j, pred) \in E$;
$L'_i(a_p) = k$ if $L_i([a, p]) = k$ for $1 \leq i \leq m$;
$a_p b_q \to a_{p-e} b_{q+e} \in R'_i$ if $a \xrightarrow{e} b \in R_i$, $0 \leq p, q \leq S, p \geq e$;
$(a_p, go) \in P'_i$ if $(i, j, \geq r) \in E$ for $r \leq p \leq S$ or $(i, j, \leq r) \in E$ for $0 \leq p \leq r$.

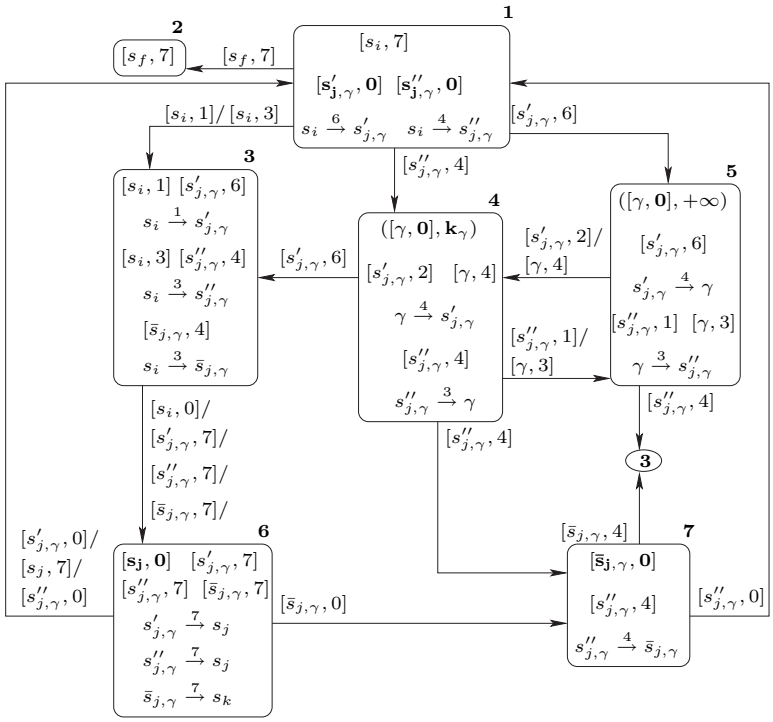


Fig. 4. The conformon-P system with priorities related to Theorem 2.

As V is by definition a finite alphabet and Π has finite total value, then the cardinality of V' is finite and equivalent to $S|V|$. Similarly, also the cardinality of the sets of rules is finite. It should be clear that the rules in the sets R'_i simulate the interaction between conformons, while the rules in the sets P'_i simulate the communication of conformons.

The initial configuration of Π is closely related to the one of Π': if n copies of the conformon $[a, p]$ are present in compartment i in Π, then n copies of the object a_p are present in compartment i of Π'. The system Π' simulates Π faithfully either if $\omega = in$ or $\omega = out$. An object will be present in the compartment ack of Π' if and only if a conformon can be present in the compartment ack of Π.

We conclude that $N(\Pi') = N(\Pi)$. □

The transcription-P system Π' constructed as in the proof of the previous theorem from a conformon-P system Π is cycle-free as soon as none of the compartments in Π contains rules of the form $a \xrightarrow{e} b$ and $b \xrightarrow{e} a$ for a and b conformons and $e \in \mathbb{N}$, and this is indeed the case for the systems constructed in the proofs of Theorems 1 and 2. Consequently, we have:

Corollary 1. *The family of sets of numbers generated by cycle-fee EC tissue-like P systems coincides with the family of sets of numbers generated by partially blind register machines.*

Like in the case of conformon-P systems, we can consider that also in tissue-like P systems the evolution rules have priority on communication rules. Then, from Theorem 2 we have:

Corollary 2. *The family of sets of numbers generated by cycle-free EC tissue-like P system with priorities coincides with the family of sets of numbers generated by register machines (hence with the family of Turing computable sets of numbers).*

The communication rules of tissue-like P systems can be avoided by adding targets to objects produced by evolution rules. In systems constructed as above, starting from conformon-P systems, we will get rules of the following forms: $ab \to (c, tar_1)(d, tar_2)$, with a, b, c, d symbols of the alphabet of Π and $tar_1, tar_2 \in \{here, in\}$, with the mentioning that *here* is omitted when specifying the rules.

Theorem 4. *Given any VR conformon-P system Π, we can construct a tissue-like P system Π' such that $N(\Pi') = N(\Pi)$.*

Proof. Given $\Pi = (V, \mu, \omega_z, ack, L_1, \ldots, L_m, R_1, \ldots, R_m)$, we construct $\Pi' = (V', \mu', \omega_z, ack, L'_1, \ldots, L'_m, U'_1, \ldots, U'_m)$ such that $N(\Pi') = N(\Pi)$ as follows. Let as assume that $\mu = (Q, E)$ and S is the sum of the values of the conformons in Π. Then:

$V' = \{a_p \mid a \in V, 0 \leq p \leq S\}$;
$\mu' = (Q, E')$ with $(i, j) \in E'$ for each $(i, j, pred) \in E$;
$L'_i(a_p) = k$ if $L_i([a, p]) = k$ for $1 \leq i \leq m$;
$a_p b_q \to a_{p-e} b_{q+e} \in U'_i$ if $a \xrightarrow{e} b \in R_i$, $0 \leq p, q \leq S, p \geq e$;
$a_p \to (a_p, go) \in U'_x$ if $(i, j, \geq r) \in E$ for $r \leq p \leq S$ or $(i, j \leq r) \in E$ for $0 \leq p \leq r$.

The system Π' simulates Π in a very similar way to the simulation described in the proof of Theorem 3. In the present proof, rules having *here* as both target indicators are equivalent to the interaction rules present in the proof of Theorem 3, and the remaining rules are equivalent to the communication rules present in the proof of Theorem 3.

Consequently, we obtain $N(\Pi') = N(\Pi)$. □

Similarly as before we have:

Corollary 3. *The family of sets of numbers generated by cycle-fee tissue-like P system coincides with the family of sets of numbers generated by partially blind register machines.*

If we assume that evolution rules having *here* as both target indicators have priority on the remaining rules, then we have:

Corollary 4. *The family of sets of numbers generated by cycle-fee tissue-like P system with priorities coincides with the family of sets of numbers generated by register machines (hence with the family of Turing computable sets of numbers).*

5 Concluding Remarks

Starting from some simple observations about the conformon-P systems which simulate (partially blind or arbitrary) register machines, we have inferred a series of results about the computing power of asynchronous tissue-like P systems, of the "standard" form (with targets associated with reaction products) and of the EC (evolution separated from communication) form. Cycle-free systems are obtained, which is an important feature for implementing P systems in a biochemical framework.

Some open problems remains to be considered. For instance, the tissue-like P systems deriving from the four corollaries we stated in the previous section have the same underlying graph (compartment structure) as the conformon-P systems depicted in Figure 3 and Figure 4, which, in turn, depend on the number of instructions of the register machines simulated by the respective conformon-P systems. Can the number of membranes be bounded (by a small number)? It is also worth trying to find interesting sets of numbers which can be computed (generated or accepted) by cycle-free systems as above.

References

1. M. Cavaliere: Evolution-communication P systems. In Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds., *Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, August 2002, Revised Papers*, LNCS 2597, Springer-Verlag, 2003, 134–145.
2. D.W. Corne, P. Frisco: Dynamics of HIV infection studied with cellular automata and conformon-P systems. *BioSystems*, 91, 3 (2008), 531–544.
3. P. Frisco: The conformon-P system: A molecular and cell biology-inspired computability model. *Theoretical Computer Science*, 312, 2-3 (2004), 295–319.
4. P. Frisco: Infinite hierarchies of conformon-P systems. In H.J. Hoogeboom, Gh. Păun, G. Rozenberg, and A. Salomaa, eds., *Membrane Computing, 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 17-21, 2006, Revised Selected, and Invited Papers*, LNCS 4361, Springer, Berlin, 2006, 395–408.
5. P. Frisco: A hierarchy of computational processes. *Technical report HW-MACS-TR-0059*, Heriot-Watt University, 2008.
6. P. Frisco: Conformon-P systems with negative values. In G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa, eds., *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 2007, Revised Selected and Invited Papers*, LNCS 4860, Springer, Berlin, 2007, 21–32.
7. P. Frisco, D.W. Corne: Modeling the dynamics of HIV infection with conformon-P systems and cellular automata. In G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa, eds., *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 2007, Revised Selected and Invited Papers*, LNCS 4860, Springer, Berlin, 2007, 21–32.
8. P. Frisco, R.T. Gibson: A simulator and an evolution program for conformon-P systems. In *SYNASC 2005, 7th International Symposium on Simbolic and Numeric Algorithms for Scientific Computing*, Workshop on Theory and Applications of P

Systems, TAPS, Timişoara (Romania), September 26-27, 2005, IEEE Computer Society, 2005, 427–430.
9. R. Gershoni, E. Keinan, Gh. Păun, R. Piran, T. Ratner, S. Shoshani: Research topics arising from the (planned) P systems implementation experiment in Technion. In this volume.
10. D.E. Green, S. Ji: The electromechanical model of mitochondrial structure and function. In J. Schultz, B. F. Cameron, eds., *Molecular Basis of Electron Transport*, Academic Press, New York, 1972, 1–44.
11. S. Ji: The Bhopalator: a molecular model of the living cell based on the concepts of conformons and dissipative structures. *Journal of Theoretical Biology*, 116 (1985), 395–426.
12. S. Ji: Free energy and information contents of conformons in proteins and DNA. *BioSystems*, 54 (2000), 107–214.
13. S. Ji: The Bhopalator: an information/energy dual model of the living cell (II). *Fundamenta Informaticae*, 49, 1-3 (2002), 147–165.
14. G. Kemeny, I.M. Goklany: Polarons and conformons. *Journal of Theoretical Biology*, 40 (1973), 107–123.
15. G. Kemeny, I.M. Goklany: Quantum mechanical model for conformons. *Journal of Theoretical Biology*, 48 (1974), 23–38.
16. Gh. Păun, Y. Sakakibara, T. Yokomori: P systems on graphs of restricted forms. *Publ. Math. Debrecen*, 60 (2002), 635–660
17. M.V. Volkenstein: The conformon. *Journal of Theoretical Biology*, 34 (1972), 193–195.

Testing Einstein's Formula on Brownian Motion Using Membrane Computing

Manuel A. Gálvez-Santisteban, Miguel A. Gutiérrez-Naranjo,
Daniel Ramírez-Martínez, Elena Rivero-Gil

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mails: mangalsan@alum.us.es, magutier@us.es,
thebluebishop@gmail.com, elen.rg@gmail.com

Summary. Brownian motion refers to erratic movements of small particles of solid matter suspended in a fluid and it is the basis of the development of many fractals found in Nature. In this paper we use the Membrane Computing model of P systems with membrane creation and the software tool JPLANT [15] in order to check the Einstein's theory on the Mean Square Displacement of Brownian motion.

1 Introduction

In [5], a first study was presented by showing the relation between fractals and P systems. On the one hand, a *fractal* is a shape made of parts similar to the whole in some way. This self-similarity occurs over an infinite range of scales in pure mathematical structures but over a finite range in many natural objects such as clouds, coastlines or snowflakes. In many plants and also organs of animals, this has led to fractal branching structures. For example, in a tree the branching structure allows the capture of a maximum amount of sun light by the leaves; the blood vessel system in a lung is similarly branched so that the maximum amount of oxygen can be assimilated (see [11]).

On the other hand, as pointed out in [5], cell-like P systems have several properties which make them suitable for the study of fractals:

- P systems can be considered as a structure of nested processors placed in a tree-structure, i.e., we can consider computations on many scales.
- If we consider P systems where membranes can be dissolved, divided or created, we usually obtain a geometrical shape too irregular to be described in traditional geometrical language, both locally and globally.

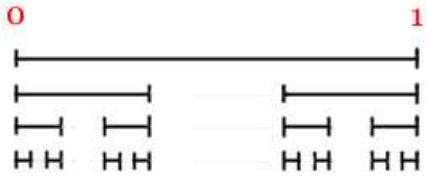


Fig. 1. First steps for the middle third Cantor set

- Computations in P systems are obtained by the application of a fixed (often only a few) set of rules. The application of these rules allows to obtain a configuration C_{n+1} from C_n.
- The computation of a P system is discrete, i.e., it is a process performed *step* by *step*.

In that paper a pair of examples were provided based on the cell-like model of P systems with membrane creation: the middle third Cantor set [2] (see Fig. 1) and the Koch curve [7, 8]. If we put together three Koch curves we have the fractal known as *Koch Snowflake* (see Fig. 2).

Self-similar fractals as Koch curve differs from natural fractals in one significant aspect. They are *exactly* self similar, and they cannot be considered as *realistic* models of natural fractals. In [5], *statistically self-similar* objects were also considered. The property that objects can look statistically similar while at the same time different in detail at different length scales, is the central feature of fractals in Nature. Randomizing a deterministic classical fractal is the first approach generating a realistic natural shape. Figure 3 shows a random Koch snowflake. Note that this fractal represents a *realistic* shape of a fractal from Nature.

In this paper we follow this research line and we present a study on the Brownian motion. Brownian motion refers to the erratic movements of small particles of solid matter suspended in a fluid and it is the basis of many random fractals found in Nature. The study of fractals and P systems needs, in the same way that other studies with P systems that involve a large number of configurations, the appropriate software in order to do the corresponding simulations. Our study considers a large amount of branches in the computational tree of a P system and for that we have used JPLANT, which is a software tool[1] that computes the first configurations of a computation and draws the corresponding graphical representation. This graphical representation provides the necessary information for carrying out our experiments.

The paper is organized as follows. First we recall the stochastic restricted P system model, its graphical representation and the software tool JPLANT used for its representation. In section 3, a brief introduction to the Brownian motion

[1] A detailed description of JPLANT can be found in [15].

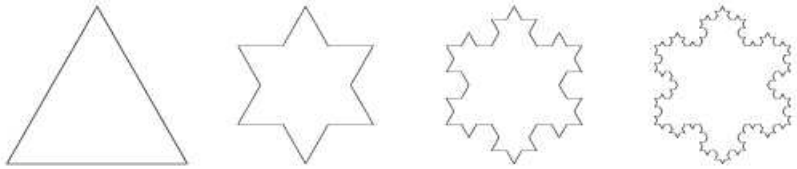


Fig. 2. First steps for the Koch Snowflake

together with our experiments are presented. Some conclusions and lines for future research are given in the last section.

2 P Systems with Membrane Creation

In this paper we will consider stochastic restricted P systems with membrane creation. This P system model has already been used for the study of graphics with P systems (see [16]). This model follows a research line in Membrane Computing that incorporates randomness into membrane systems (see [1, 10, 13] and the references therein). In this model, to pass from a configuration of the system to the next one we apply to every object present in the configuration a rule chosen at random, according to given probabilities, among all the rules whose left–hand side coincides with the object[2]. The second ingredient in this model is membrane creation, which was first introduced in [6, 9]. However, our needs are far simpler than the models found in the literature. In this *restricted* model we only consider object-evolution rules and creation rules.

The non-determinism is one of the main features of P systems and the possibility of reaching different configurations leads us to consider different graphical representations in the evolution of a P system.

A restricted P system with membrane creation is a tuple $\Pi = (O, \mu, w_1, \ldots, w_m, R)$ where:

1. O is the alphabet of *objects*. There exist two distinguished objects, F and W that always belong to the alphabet.
2. μ is the initial *membrane structure*, consisting of a hierarchical structure of m membranes (all of them with the same label; for the sake of simplicity we omit the label).
3. $w_1, \ldots, w_m$ are the multisets of objects initially placed in the m regions delimited by the membranes of μ.
4. R is a finite set of *evolution rules* associated with every membrane, which can be of the two following kinds:
   a) $a \xrightarrow{p} v$, where $a \in O$, v is a multiset over O, and $0 \leq p \leq 1$ is a real number representing the probability of the rule. This rule replaces an object a present in a membrane of μ by the multiset of objects v.

[2] This idea was also presented in [16].

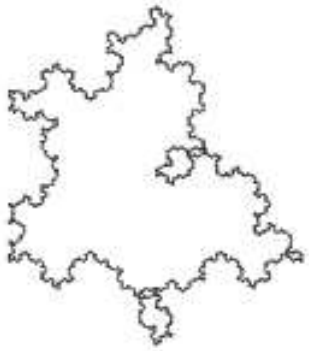


Fig. 3. Random Koch Snowflake

b) $a \xrightarrow{p} [v]$, where $a \in O$, v is a multiset over O, and $0 \leq p \leq 1$ is a real number representing the probability of the rule. This rule replaces an object a present in a membrane of μ by a new membrane with the same label and containing the multiset of objects v.

The addition of the probabilities of the rules with the same left-hand side must be one. If there is only one rule for a given left-hand side, then its probability must be one and, for the sake of simplicity, we omit it.

A membrane structure (extending the membrane structure μ) together with the objects contained in the regions defined by its membranes constitute a configuration of the system. A computation step is performed applying to a configuration the evolution rules of the system in a non-deterministic maximally parallel way.

A rule in a region is applied if and only if the object occurring in its left–hand side is available in that region; this object is then consumed and the objects indicated in the right–hand side of the rule are created inside the membrane. The rules are applied in all the membranes simultaneously, and all the objects in them that can trigger a rule must do it. When there are several possibilities to choose the evolution rules to apply, non-determinism takes place.

2.1 Graphical Representation

In this section we show how to use, through a suitable graphical representation, restricted P systems with membrane creation to model branching structures. The key point of the representation relies on the fact that a membrane structure is a *rooted tree of membranes*, whose root is the skin membrane and whose leaves are the elementary membranes. It seems therefore a perfect frame to encode the branching structure.

Let us suppose that the alphabet O of objects contains the objects F and W and let us fix the lengths l and w.

A simple model to graphically represent a membrane structure is to make a depth-first search of it, drawing, for each membrane containing the object F, a segment of length $m \times l$, where m is the multiplicity of F. If the number of copies of F in a membrane increases along the computation, the graphical interpretation is that the corresponding segment is lengthening. Analogously, the multiplicity of the symbol W specify the width of the segments to be drawn as follows: if the number of objects W present in a membrane is n, then the segment corresponding to this membrane must be drawn with width $n \times w$.

Each segment is drawn rotated with respect to the segment corresponding to its parent membrane. In order to determine the rotation angle we need to fix a third parameter δ. Such angle δ together with the length l and the width w will determine the picture of the P system.

In order to compute the rotation angle of a segment with respect to its parent membrane we consider two new objects that can appear in the alphabet: $+$ and $-$. The rotation angle will be $n \times \delta$, where n is the multiplicity of objects "+" minus the multiplicity of objects "−" in the membrane. That is, each object "+" means that the rotation angle is increased by δ whereas each object "−" means that it is decreased by δ.

Inside the membranes other objects can appear that do not have geometrical interpretation. They are related to the development of the graph in time.

For example, let us consider Π_2 the following restricted P system with membrane creation:

- The alphabet of objects is $O = \{F, W, B_l, B_s, B_r, L, L_1, E\}$.
- The initial membrane structure together with the initial multiset of objects is $[F^2\, W\, B_l\, B_s\, L_1\, E]$.
- The rules are:

$$\begin{array}{ll} B_l \xrightarrow{1/2} [+\, F\, W\, B_l\, B_s\, L\, E] & L \to L\, F \\ B_l \xrightarrow{1/2} [-\, F\, W\, B_l\, B_s\, L\, E] & L_1 \to L_1\, F^2 \\ B_r \xrightarrow{1/2} [+\, F\, W\, B_l\, B_s\, L\, E] & E \to E\, W \\ B_r \xrightarrow{1/2} [-\, F\, W\, B_l\, B_s\, L\, E] & B_s \to [F\, W\, B_l\, B_r\, L_1\, E] \end{array}$$

There exist two rules for the evolution of the object B_l and two possibilities for the evolution of the object B_r. The probability for each choice is $1/2$. Notice that we do not make explicit the probability of the rule when this is one.

Figure 4 shows four different configurations after the second step of this P system with the angle $\delta = 15$.

2.2 Software

As usual, the hand-made simulation of the evolution of a P system is a heavy task. In this paper we use a new software tool called JPLANT[3]. It computes

[3] A detailed description with examples can be found in [15].

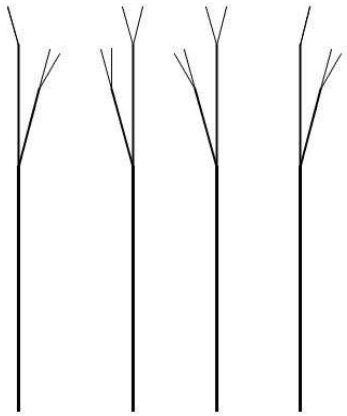


Fig. 4. Four configurations after the second step

the first configurations of a computation of a stochastic restricted P system with membrane creation and draws the corresponding graphical representation of the configurations of such computation.

The program has been written in Java and it has a nice intuitive user-friendly graphical interface. The output is a picture with a set of connected segments drawn according to the rules described in Section 2. For each new configuration, a new picture is drawn, so the output of this tool is a sequence of pictures which can be saved in several computer graphic formats.

The graphical representation of one configuration is not unique. It depends on the parameters l, w and δ which determine the length and width of the segments as well as the rotation angle with respect to the segment corresponding to the parent membrane. These parameters must be also provided by the user and with the initial configuration and the rules, they are the input of the tool.

3 Brownian motion

Brownian motion refers to erratic movements of small particles of solid matter suspended in a liquid. These movements can only be seen under microscope. After the discovery of such movement of pollen it was believed that the cause of the motion was biological in nature. However, about 1828, the botanist Robert Brown realized that a physical explanation, rather than the biological one, was correct. The effect is due to the influence of very light collisions with the surrounding molecules. The standard theory of Brownian motion due to Einstein, Smoluchowski, Langevin, Fokker and Planck is based on the model where a particle moves in a dense medium which generates friction and random collisions.

In 1905 Einstein published a mathematical study of this motion, which eventually led to Perrin's Nobel prize-winning calculation of Avogadro's number. A rigorous probabilistic model of Brownian motion was proposed by Wiener in [18]. He constructed a process which exhibits random behavior very similar to that of Brownian motion. The theoretical problems connected with Brownian motion have many interesting applications in different fields, such as in the theory of sound [14], in physical chemistry [4] and biophysics [17].

In this paper we will consider the special case in which the particle moves a constant distance in each time unit (constant speed) and after each time unit the particle randomly chooses a new direction. The question is to know if we can make any prediction about the total displacement after n steps.

Instead of asking for the total expected displacement, i.e., the displacement of a particle averaged over many samples, the specialized literature focuses on the average of the square of the displacements, the *mean square displacement*. In 1905, Einstein showed that the mean square displacement is *proportional* to time[4]. The factor of proportionality depends on the speed, the step length and the dimension of the space. This is the fundamental property of Brownian motion, verified experimentally in 1908 by the French physicist Jean Perrin (see [12]).

Next, we shows the result of our experiments in order to check the fundamental property of the Brownian motion. We use the probabilistic P system

$$
\begin{array}{l}
\texttt{Initial configuration:} \quad [F\,W\,H] \\
\qquad\qquad \texttt{Rules:} \quad H \xrightarrow{1/24} [-\,F\,W\,H] \\
\qquad\qquad\qquad\quad H \xrightarrow{1/24} [-^{2}\,F\,W\,H] \\
\qquad\qquad\qquad\quad H \xrightarrow{1/24} [-^{3}\,F\,W\,H] \\
\qquad\qquad\qquad\qquad\quad \cdots \\
\qquad\qquad\qquad\quad H \xrightarrow{1/24} [-^{23}\,F\,W\,H] \\
\qquad\qquad\qquad\quad H \xrightarrow{1/24} [F\,W\,H]
\end{array}
$$

where all the segments have the same length and each new step can a deviation of $n \times 15$ degrees, where n is non-deterministically chosen in $\{0, 1, 2, \ldots, 23\}$.

Since JPLANT is able to simulate discrete Brownian trajectories, we can try to link classical applications of such movements with this new modeling software.

3.1 The experiment

A particle dropped into a fluid describes a Brownian trajectory. Because of the impacts along its path with other particles, several different routes can be traced for this single traveler. For each impact, the trajectory of this moving particle changes its direction. If we assume that no energy is lost due to the impacts,

[4] See [3] for details.

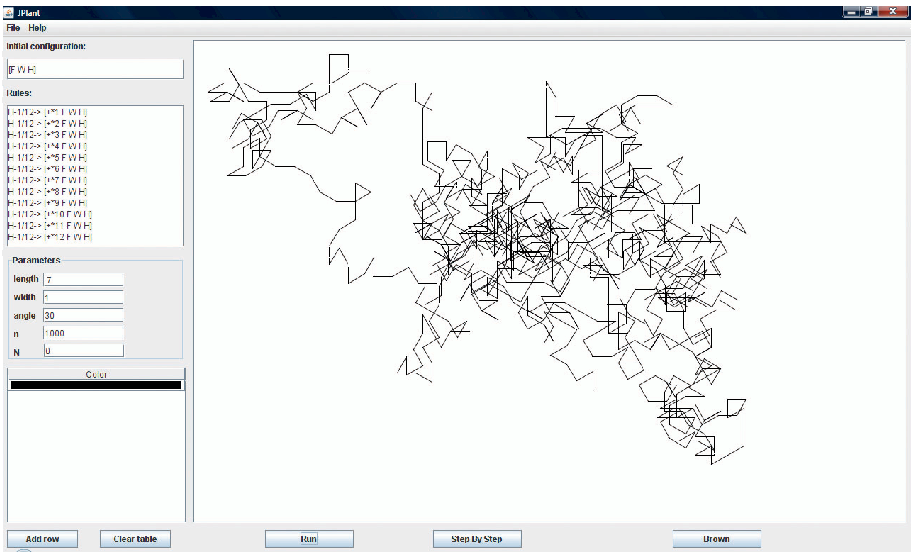


Fig. 5. Snapshot from JPLANT showing a Brownian trajectory.

the set of possible trajectories can be modeled by JPLANT, just by giving some interpretation to the parameters.

This model of trajectory is quite common in physical and biological systems; from solid state electronics to cell membrane dynamics, even in stochastic signal processes there are many examples of Brownian motions describing several noisy behaviors.

When not only one particle but many of them (a gas cloud, for example) are moving into a fluid, this is called a diffusion process. Depending on the conditions and materials, these processes have different behaviors. The same happens when a black ink drop falls into a glass of liquid; depending on the densities, the shapes of the molecules, the viscosities, the temperature, etc., the black cloud will spread faster or slower. In this way, a diffusion process is the result of an overlapping of many Brownian motions evolving in a parallel way.

Albert Einstein studied Brownian motions and extracted some essential mathematical properties from them. For example, he showed that the mean square of the distance traveled by a particle is proportional to the elapsed time. Using the square of the distances instead of the distances themselves is a key point. The mean of the distances does not give any information because of the uniform distribution of the possible orientations for each step in the path. If we consider a one-dimensional Brownian motion, the expected value for the position is the origin (50% for positive step and 50% for negative step). Nonetheless we have a positive magnitude for each step with squared distances, which admits some parameter definitions. Einstein proposed the following expected value for the Mean Square displacement (MSD, for short):

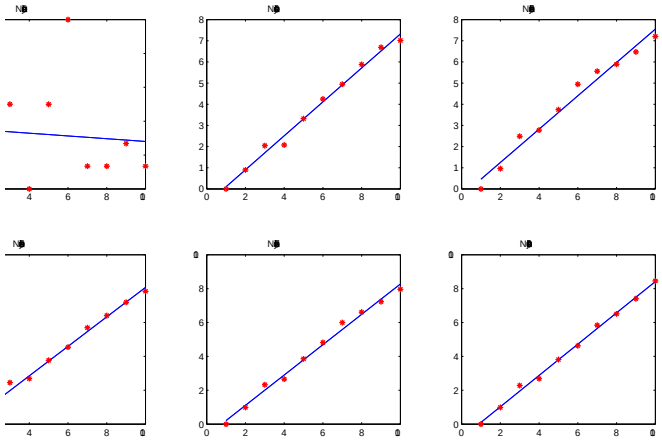


Fig. 6. When N increases, the fitting gets better.

$$MSD =< s^2 >= kDt + C \tag{1}$$

This expression can be calculated from a set of Brownian particles which are diffusing into a fluid. When the MSD data fits into a linear function, the slope is a measurement of how fast the particles are expanding into the fluid. The parameter k is a constant which depends on how many dimensions are considered (for three-dimensional diffusions $k = 6$, for two-dimensional diffusions $k = 4$). The parameter C is just an offset adjustment needed for the regression and D is the diffusion rate.

After a statistical analysis of the data provided by JPLANT, it is possible to model problems involving such diffusion processes. We propose two tasks to be accomplished by Brownian data from JPLANT:

- Check the Einstein's formula and its convergence. Einstein claimed that if the number of experiments grows, the linearity gets stronger, so the absolute error between raw data and the linear regression must tend to zero.
- If the probability of collision increases, that means that for a fixed time of diffusion, the ability to diffuse must decrease. As far as the diffusion rate measures this, the coefficient should get lower as probability increases.

Einstein's relation for the Mean Square Displacement claims that it is directly proportional to the elapsed time. Thus, if the number of observed trajectories under same conditions increases, the linear relation gets stronger, so the relation (1) fits into the data in a better way.

In order to check Einstein's claim we designed the following experiment:

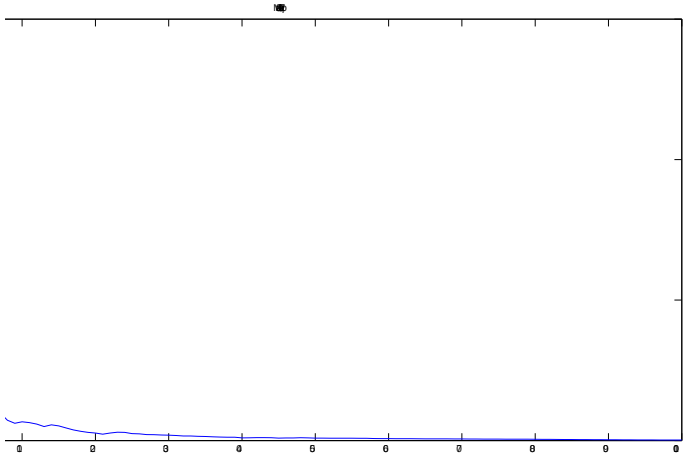


Fig. 7. MSE vs. number of trajectories

- We use JPLANT in order to get the data corresponding to one hundred bidimensional trajectories of Brownian particles.
- For each N with $1 \leq N \leq 100$, we calculate the MSD of the corresponding trajectories and its linear regression.
- In order to consider how the MSD fits to the regression line we calculate the Mean Squared Error.
- The experiments show that when the number of trajectories N considered increases, then the Mean Squared Error tends to zero as Einstein predicted.

Figure 6 illustrates the results of the experiment. In the first frame, only two trajectories are considered. It is easy to check that the points do not fit into the regression line. In the second frame, 10 trajectories are considered and the MSD fits better into the regression line. The following four frames shows the adjustment to the regression line for 25, 50, 75 and 100 trajectories so the data cloud gets tighter to the line as N gets higher.

Figure 7 shows the relation between the number of experiments and the Mean Squared Error. It is clear that when the number of trajectories increases, the MSE tends to zero.

4 Conclusions and Future Work.

In this paper we have used JPLANT as a Brownian simulation tool, testing some Einstein's results and generating new possible paths of study, starting from the idea of a partial recreation of a real experiment. The same methodology, extended appropriately, could be applied to other biological processes or electronic models.

This way, we provide a new application for membrane computing, being useful to model and, maybe, extend classical ways of simulation for such problems.

As a goal for future simulations, we propose the modeling of real experiments maybe by extending the P system model with new types of rules that capture the dynamics of the real experiments: division, cooperation, dissolution, ... In the same way, a deeper study of the use of probabilities in Membrane Computing can be useful in order to model experiments from the real world. Thermic noise or biological membranes are good candidates to be simulated in the immediate future because of the extensive bibliography and the practical use of these concepts.

Acknowledgement

The authors acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the project of excellence TIC-581 of the Junta de Andalucía.

References

1. I.I. Ardelean, M. Cavaliere: Modelling Biological Processes by Using a Probabilistic P System Software. Natural Computing, 2(2), (2003), 173–197.
2. G. Cantor: *Über unendliche, lineare Punktmannigfaltigkeiten V*. Mathematische Annalen, 21, (1883), 545–591.
3. W. Ebeling: Non Linear Brownian Motion – Mean Square Displacement. Condensed Matter Physics, 7, 539, (2004), 1–12.
4. P.-G. de Gennes: Wetting: Statics and Dynamics. Reviews of Modern Physics 57, (1985), 827–863.
5. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: Fractals and P Systems. In Graciani–Díaz, C., Păun, Gh., Romero–Jiménez, A., Sancho–Caparrini, F. (eds.): Proceedings of the Fourth Brainstorming Week on Membrane Computing, Vol. II. Fénix Editora, Sevilla, (2006), 65–86.
6. M. Ito, C. Martín–Vide, Gh. Păun: A Characterization of Parikh Sets of ET0L Languages in Terms of P Systems. In Ito, M., Păun, Gh., Yu, S. (eds.): Words, Semigroups, and Transducers. World Scientific, (2001), 239–254.
7. H. von Koch. Sur one courbe continue sans tangente, obtenue par une construction géometrique élémentaire. Arkiv för Matematik 1, (1904), 681–704.
8. H. von Koch. Une méthode géometrique élémentaire pour l'étude de certaines questions de la théorie des courbes planes. Acta Mathematica 30, (1906), 145–174.
9. M. Madhu, K. Krithivasan: P Systems with Membrane Creation: Universality and Efficiency. In Margenstern, M., Rogozhin, Y. (eds.): Proceedings of the Third International Conference on Universal Machines and Computations. Lecture Notes in Computer Science, 2055, (2001), 276–287.
10. A. Obtulowicz, Gh. Păun: (In Search of) Probabilistic P Systems. Biosystems 70(2), (2003), 107–121.
11. H.-O. Peitgen, H. Jürgens, D. Saupe: Fractals for the Classroom. Springer-Verlag (1992).

12. H.-O. Peitgen, H. Jürgens, D. Saupe: Chaos and Fractals. New Frontiers of Scisnce, 2nd Ed. Springer, (2004).
13. D. Pescini, D. Besozzi, G. Mauri, C. Zandron: Dynamical Probabilistic P Systems. International Journal of Foundations of Computer Science 17(1), (2006), 183–204.
14. J.W. Rayleigh: The Theory of Sound, vol I. 2nd edition. New York, Dover, (1945).
15. E. Rivero-Gil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: Graphics and P Systems: Experiments with JPLANT. In these Proceedings.
16. A. Romero-Jiménez, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, M.J.: Graphical Modelling of Higher Plants Using P Systems. In H.J. Hoogeboom, Gh. Paun, G. Rozenberg, A. Salomaa (eds.) Membrane Computing, Seventh International Workshop, Lecture Notes in Computer Science, 4361, (2006), 496–506.
17. M. Schienbein and H. Gruler: Langevin Equation, Fokker-Planck Equation and Cell Migration. Bulletin of Mathematical Biology, 55, (1993), 585-608.
18. N. Wiener. Differential Space. Journal of Mathematical Physics, 2, (1923), 131–174.

Research Topics Arising from the (Planned) P Systems Implementation Experiment in Technion

Renana Gershoni[1], Ehud Keinan[1,2], Gheorghe Păun[3],
Ron Piran[1], Tamar Ratner[1], Sivan Shoshani[1]

[1] Schulich Faculty of Chemistry
Technion – Israel Institute of Technology
Haifa 32000, Israel
[2] The Scripps Research Institute
Departments of Molecular Biology and the Skaggs Institute for Chemical Biology
10550 North Torrey Pines Road, La Jolla, California 92037, USA
E-mails: keinan@technion.ac.il, keinan@scripps.edu
[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
and Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mails: george.paun@imar.ro, gpaun@us.es

Summary. We formulate here a few technical (mathematical) open problems related to the *in vitro* bio-chemical experiment planned in Technion for computing the Fibonacci sequence in terms of P systems. So-called *local-loop-free* P systems are introduced and their universality for various types of P systems as well as other issues are mentioned as research questions.

1 Introduction

Although in the fall of this year membrane computing counts one decade since its beginnings (since the paper [1] was circulated as a technical report of TUCS, Finland), so far no attempt to implement P systems in a laboratory, using a bio-chemical support, was reported. Recently, such an experiment was planned, in the Chemical Faculty of Technion Institute, Haifa. This will be an *in vitro* experiment, using test tubes as membranes and DNA molecules as objects, evolving under the control of enzymes. The computation to implement was chosen to be the generation of a bunch of numbers in the famous Fibonacci sequence.

As expected, such an attempt raised a series of difficulties related to the type of P system which is possible to simulate/implement. After briefly mentioning

these difficulties, we introduce the type of P systems which seem to avoid them. The basic issue is to have no loops in the evolution of objects/substances present in a membrane/compartment, because this would lead to cycles which cannot be "read" from outside in a useful way, to equilibrium states which are not "useful" for the computation.

Local-loop-free (in short, LL-free) P systems are introduced and the power of this restriction is a natural issue to investigate from a mathematical and computational point of view. We only show here that LL-free tissue-like P systems with cooperative rules are universal, but the question remains open for other classes of P systems, in particular, for the catalytic ones. A few related questions are mentioned.

2 Difficulties and (Hopefully) Solutions Related to Implementing P Systems

The basic features of membrane computing are (1) *compartmentalization*, by means of cell-like membranes, (2) *multisets* (sets with multiplicity associated with their elements, which means *counting* the objects present in membranes), (3) bio-inspired *evolution rules*, which are reaction-like (for processing multisets), communication rules (e.g., symport and antiport rules), membrane handling rules, etc., (4) *synchronization* of compartment evolution, for instance, using the rules in a maximally parallel manner, (5) *communication* between compartments; we can also mention (6) defining the result of a computation mainly for *halting* computations, but this is not specific to membrane computing (and can also be avoided).

In order to implement a P system in a laboratory, all or most of these features should be implemented. Compartments can be obtained by using standard test tubes or similar labware, multisets are usual in bio-chemistry, but... without a precise counting. Still, by defining carefully some "moles" of substances, one can count in terms of such ad-hoc moles. Anyway, full synchronization and parallelism cannot be guaranteed by bio-chemical reactions, hence a certain degree of non-determinism/approximation should be allowed in the experiment. In particular, a good degree of synchronization can be obtained by "waiting enough", such that all reactions that can take place in a test tube actually take place – and this raises an important issue: these reactions should not cycle, the process should be finite in each compartment of the system. Counting is also needed when reading the result of a computation.

In the experiment planned in Technion, the above mentioned difficulties are solved as follows: (1) test tubes for membranes (compartments), (2) multisets of pre-defined "moles" of DNA molecules, (3) enzyme driven operations with the DNA molecules, with the precaution not to have any cycle in any compartment, (4) waiting enough for reactions to take place and then (5) moving all relevant objects to the next tube in a mechanical way, with (6) the result read by spectrophotometry (certain molecules are marked and their number is estimated).

These lab solutions request finding a suitable problem or class of problems for which no cycle of substances is possible in any compartment and the solution allows a degree of approximation. We said nothing above about halting, because in the experiment this feature is not taken into consideration: a sequence of numbers (the famous Fibonacci one) are computed, hence several outputs, at precise time moments, are produced.

3 Local-loop-free P Systems

From a theoretical point of view, the central issue is that of finding a non-trivial class of P systems such that the reactions from each compartment are completed in a finite (better: small) number of steps. Otherwise stated, no compartment can contain a cycle of objects which can run forever.

This intuitive goal can be reached in various formal ways. For instance, we can request that no local transition graph contains a cycle (the catalysts are ignored). Specifically, for each region i of a P system with the set of objects O and set R_i of rules in region i, the transition graph $\gamma_i = (O, E)$ associated with region i has the set of edges defined as follows: for each $a, b \in O$,

$$(a, b) \in E \text{ iff there is } u \to v \in R_i \text{ such that } |u|_a \geq 1, |v|_b \geq 1.$$

(For a string x and a symbol a we denote by $|x|_a$ the number of occurrences of a in x.)

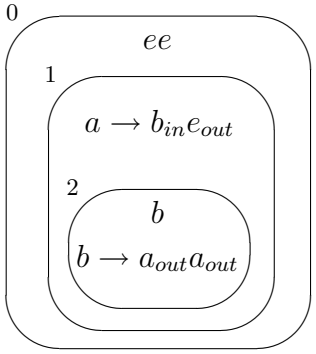


Fig. 1.

A stronger condition is to impose that no object produced in a compartment can evolve in the same compartment. In the case of non-cooperative systems, this means that the local transition graph contains no paths of length longer than or equal to two. (For cooperative systems this assertion is not true: having the rules $a \to b,\ bc \to cc$, the local transition graph contains the path (a, b, c), but it is

possible not to actually have two reactions in a row, because without c, the product b of the first reaction cannot evolve.) This latter condition is similar to the way the P systems with immediate communication are defined (see [2]): each product of a reaction is immediately communicated to one of the neighboring membranes.

Thus, formally, we can define several properties which ensure the local-loop-freeness. Defining such properties and investigating the P systems obeying them is one of the *research topics* we want to point out here.

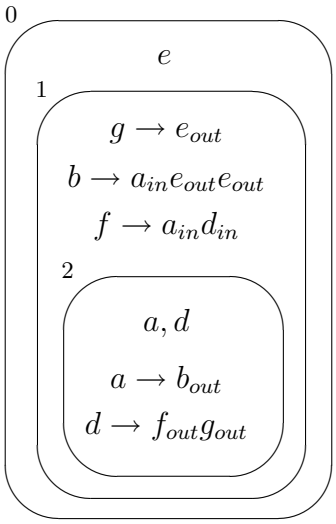


Fig. 2.

In what follows, we briefly discuss the P systems which are local-loop-free (in short, LL-free) in the sense of the previous definition: no cycle exists in any local transition graph.

4 Some Examples

We start by considering three (non-semilinear) sets of numbers which can be computed by P systems of a rather similar form. Figures 1, 2, and 3 present non-cooperative P systems (denoted by Π_1, Π_2, Π_3) generating, respectively, the following sets of numbers:

$$N(\Pi_1) = \{2^n \mid n \geq 1\},$$
$$N(\Pi_2) = \{n^2 \mid n \geq 1\},$$
$$N(\Pi_3) = \{1, 2, 3, 5, 8, 13, ...\}.$$

The third sequence is the Fibonacci one (each element is the sum of the previous two; here we start with 1 and 2 as the initial numbers), and this system Π_3 is planned to be implemented.

These systems can be represented in a more intuitive way (in what concerns the reactions taking place in compartments and the objects communicated) as tissue-like P systems with immediate communication. For systems Π_1 and Π_3 this is done in Figures 4 and 5, respectively; the case of Π_2 is left to the reader. On the arrows are indicated the objects which are communicated between the respective membranes.

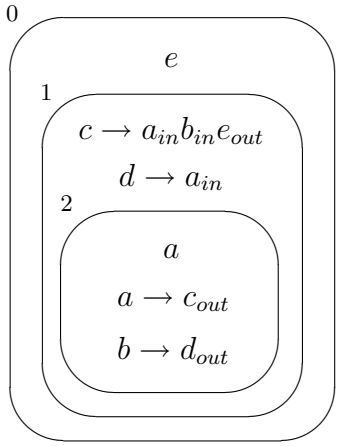


Fig. 3.

In all cases, of both cell-like and tissue-like systems, the result is collected in the form of the number of copies of a special object e in a designated membrane which has no other role in the system. We call it a *output membrane*; it contains no rule, hence no objects can evolve in it. In all cases, the environment can be used instead of this membrane, but it is "more practical" to work with a output membrane.

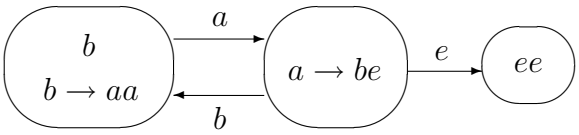


Fig. 4.

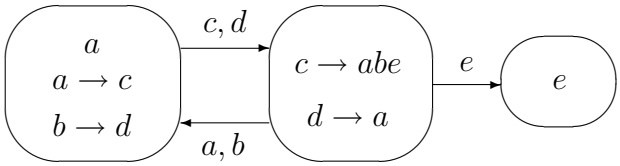


Fig. 5.

During the discussions in Technion in search of a suitable problem to implement, when we have arrived at the construction of the system in Figure 3, a generalized glee was expressed by the young members of the team, who exclaimed: "this is doable!" The idea was summarized at various stages in nice graphical forms – one of them is given in Figure 6 (the output membrane is here the inner one and one additional external membrane is considered as an infinite supplier of "raw materials") – and then a group photo was taken, to celebrate the moment (see Figure 7). Well, whether or not this moment deserves also to be celebrated with champagne it remains to be seen after trying the experiment...

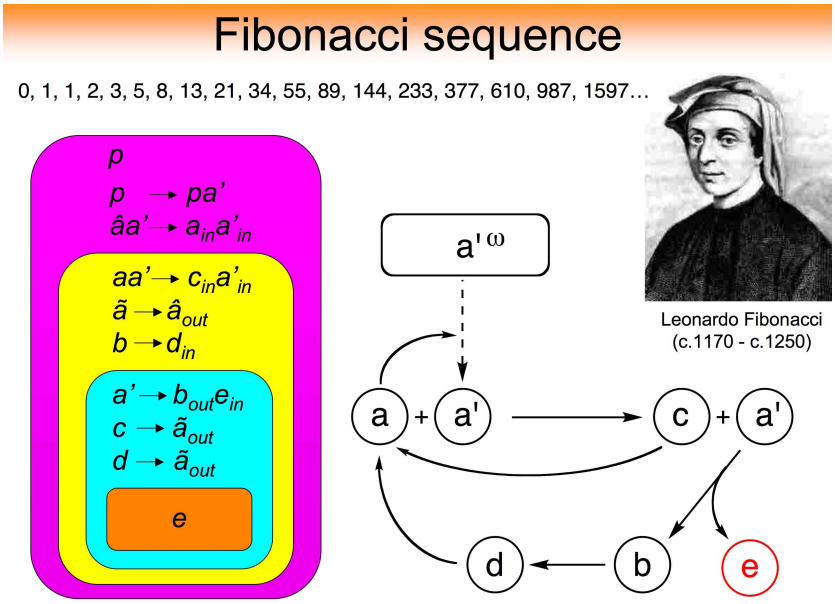


Fig. 6.

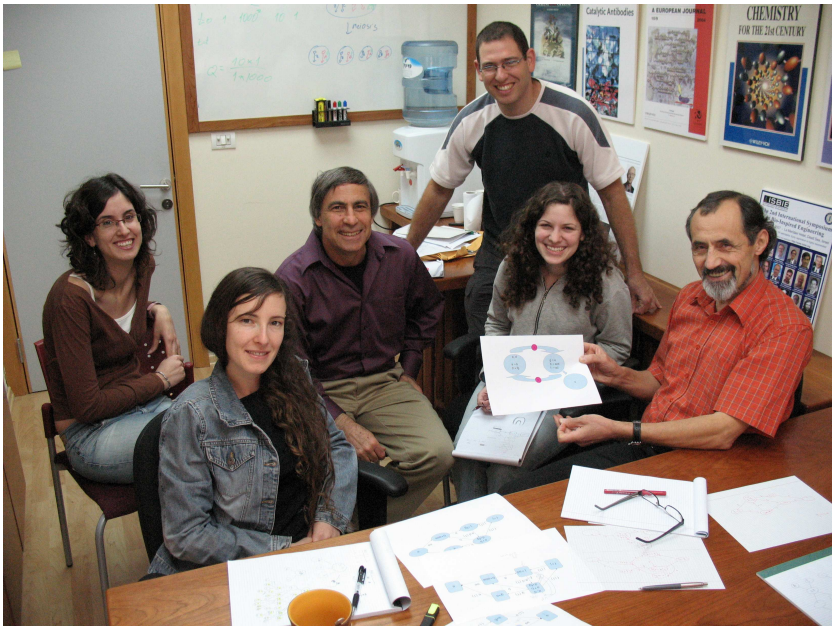


Fig. 7.

5 And Now Start the Problems

We list here only some of the most natural ones:

- Prove universality for LL-free P systems. Of course, in this case we need to consider as successful only halting computations. For tissue-like cooperative systems we give a proof of universality in the next section, but for other classes of P systems, in particular, for catalytic P systems (with two or more catalysts, or with one catalyst and various controls on using the rules) the problem remains open.
- What about considering systems with a membrane structure like those in Figures 1–3, i.e., with only two membranes for computing and one additional membrane for collecting the result of a computation? Are also such LL-free systems universal? Note that even the system in the next section, using cooperative rules, has three "computing" membranes.
- Find other examples of systems (of interest) with the LL-freeness property or with the membrane structure of the form in Figures 1–3.
- Note that the systems considered above are of a generative form, they start from an initial configuration and generate infinite sequences of numbers. Devise input-output systems, computing a function (of some interest).

- Is any chance to solve NP-complete problems in this framework?
- What about sorting, ranking, or other computer science applications of P systems (as reported in the literature), based on LL-free P systems?

6 Universality of Cooperative LL-free Systems

We denote by $tNOP_m^{llf}(coo)$ the family of sets of numbers $N(\Pi)$ generated by LL-free tissue-like P systems with cooperative rules having at most $m \geq 1$ membranes, with immediate communication and with the result collected in a special output membrane which has only this role (no object evolve in this membrane, it has no rule inside). If the result is collected in the environment, then this output membrane can be saved, but here we choose to consider it.

In this framework, we can immediately prove the following result (as usual, NRE denotes the family of recursively enumerable sets of natural numbers):

Theorem 1. *$tNOP_m^{llf}(coo) = NRE$ for all $m \geq 4$.*

The proof is based on simulating a register machine $M = (m, H, l_0, l_h, I)$ (number of registers, set of labels, initial label, halting label, set of instructions) by a P system Π constructed as suggested in Figure 8. Without any loss of the generality, we may assume that when halting, M has all registers different from register 1 empty.

All labels in H, primed versions of them (for each $l \in H$ we consider $l', l'', l''', l^{iv}, \bar{l}, \hat{l}$, too), as well as objects $a_r, 1 \leq r \leq m$, associated with the registers of M are objects in Π. We start with only one object in the system, namely l_0, present in membrane 1.

For each ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ in I we introduce the rules

$$l_i \to a_r \bar{l}_i \text{ in membrane 1,}$$
$$\bar{l}_i \to \hat{l}_i \text{ in membrane 2,}$$
$$\hat{l}_i \to l_j \text{ and}$$
$$\hat{l}_i \to l_k \text{ in membrane 3.}$$

The simulation of the ADD instruction is obvious: the increment of register r is done in membrane 1 and the non-deterministic choice of the next instruction to apply is done in membrane 3.

For each SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ in I we introduce the rules

$$l_i \to l_i' l_i'' \text{ in membrane 1,}$$
$$l_i' a_r \to l_i''' \text{ and}$$
$$l_i'' \to l_i^{iv} \text{ in membrane 2,}$$
$$l_i^{iv} l_i''' \to l_j \text{ and}$$
$$l^{iv} l_i' \to l_k \text{ in membrane 3.}$$

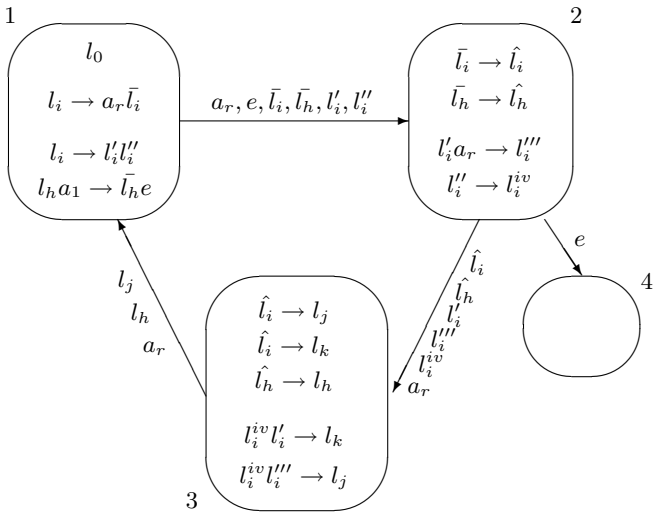


Fig. 8.

The simulation of the SUB instruction is done in three steps (a complete cycle through membranes 1, 2 and 3). In membrane 1 one introduces the objects l'_i, l''_i which are moved, together with all objects a_r to membrane 2. Here, l'_i tries to subtract one from the value of register r. If this is possible, then the object l'''_i is introduced, otherwise l'_i remains unchanged. Simultaneously, l''_i introduces the object l^{iv}_i. This object behaves as a checker in membrane 4: if it mets here l'''_i (hence the subtraction was possible), then one introduces the object l_j, otherwise one introduces the object l_k. In both cases the continuation is as necessary in the register machine M.

We also consider the rules

$$l_h a_1 \to \bar{l}_h e \text{ in membrane 1,}$$
$$\bar{l}_h \to \hat{l}_h \text{ in membrane 2,}$$
$$\hat{l}_h \to l_h \text{ in membrane 3.}$$

In the end of the computation with respect to M, the object l_h transforms all objects a_1 (hence the contents of the first register of M) into objects e, which are moved to membrane 4. Thus, the computation ends with a number of copies of e in membrane 4 equal to the number generated in the first register of M. Thus, $N(M) = N(\Pi)$ and we have the inclusion $NRE \subseteq tNOP_4^{llf}(coo)$.

The converse inclusion can be proved in a straightforward way or we can invoke for it the Turing-Church thesis. □

References

1. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (first circulated as TUCS Report 208 - Turku Center for Computer Science, November 1998, `www.tucs.fi`).
2. Gh. Păun: *Computing with Membranes: An Introduction*. Springer, Berlin, 2002.

Solving Numerical NP-complete Problems by Spiking Neural P Systems with Pre–computed Resources

Miguel A. Gutiérrez-Naranjo[1], Alberto Leporati[2]

[1] 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: magutier@us.es

[2] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
E-mail: alberto.leporati@unimib.it

Summary. Recently we have considered the possibility of using spiking neural P systems for solving computationally hard problems, under the assumption that some (possibly exponentially large) pre-computed resources are given in advance. In this paper we continue this research line, and we investigate the possibility of solving numerical **NP**-complete problems such as Subset Sum. In particular, we first propose a semi–uniform family of spiking neural P systems in which every system solves a specified instance of Subset Sum. Then, we exploit a technique used to calculate Iterated Addition with boolean circuits to obtain a uniform family of spiking neural P systems in which every system is able to solve all the instances of Subset Sum of a fixed size. All the systems here considered are deterministic, but their size generally grows exponentially with respect to the instance size.

1 Introduction

Spiking neural P systems (SN P systems, for short) have been introduced in [11] as a new class of distributed and parallel computing devices, inspired by the neurophysiological behavior of neurons sending electrical impulses (*spikes*) along axons to other neurons. SN P systems can also be viewed as an evolution of P systems [24, 25, 27, 28] (the latest information can be found in [34]) corresponding to a shift from *cell-like* to *neural-like* architectures. We recall that this biological background has already led to several models in the area of neural computation, e.g., see [19, 20, 8].

In SN P systems the cells (also called *neurons*) are placed in the nodes of a directed graph, called the *synapse graph*. The contents of each neuron consist of a number of copies of a single object type, called the *spike*. Every cell may also contain a number of *firing* and *forgetting* rules. Firing rules allow a neuron to send information to other neurons in the form of electrical impulses (also called spikes) which are accumulated at the target cell. The applicability of each rule is determined by checking the contents of the neuron against a regular set associated with the rule. In each time unit, if a neuron can use one of its rules, then one of such rules must be used. If two or more rules could be applied, then only one of them is nondeterministically chosen. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other. Observe that, as usually happens in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized. When a cell sends out spikes it becomes "closed" (inactive) for a specified period of time, that reflects the refractory period of biological neurons. During this period, the neuron does not accept new inputs and cannot "fire" (that is, emit spikes). Another important feature of biological neurons is that the length of the axon may cause a time delay before a spike arrives at the target. In SN P systems this delay is modeled by associating a delay parameter to each rule which occurs in the system. If no firing rule can be applied in a neuron, there may be the possibility to apply a *forgetting rule*, that removes from the neuron a predefined number of spikes.

Formally, a *spiking neural membrane system* (SN P system, for short) of degree $m \geq 1$, as defined in [10] in the computing version (i.e., able to take an input and provide and output), is a construct of the form

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
   b) R_i is a finite set of *rules* of the following two forms:
      (1) *firing* (also *spiking)* rules $E/a^c \rightarrow a; d$, where E is a regular expression over a, and $c \geq 1$, $d \geq 0$ are integer numbers; if $E = a^c$, then it is usually written in the following simplified form: $a^c \rightarrow a; d$;
      (2) *forgetting* rules $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, we have $a^s \notin L(E)$ (the regular language defined by E);
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of *synapses* between neurons;
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and the *output* neurons of Π.

A firing rule $E/a^c \rightarrow a; d \in R_i$ can be applied in neuron σ_i if it contains $k \geq c$ spikes, and $a^k \in L(E)$. The execution of this rule removes c spikes from σ_i (thus

leaving $k - c$ spikes), and prepares one spike to be delivered to all the neurons σ_j such that $(i, j) \in syn$. If $d = 0$ then the spike is immediately emitted, otherwise it is emitted after d computation steps of the system. As stated above, during these d computation steps the neuron is *closed*, and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost), and cannot fire (and even select) rules. A *forgetting* rule $a^s \to \lambda$ can be applied in neuron σ_i if it contains *exactly* s spikes, and no firing rules are applicable. The execution of this rule simply removes all the s spikes from σ_i.

The *initial configuration* of the system is described by the numbers $n_1, n_2, \ldots, n_m$ of spikes present in each neuron, with all neurons being open. During the computation, a configuration is described by both the contents of each neuron and its *state*, which can be expressed as the number of steps to wait until it becomes open (zero if the neuron is already open). Thus, $\langle r_1/t_1, \ldots, r_m/t_m \rangle$ is the configuration where neuron σ_i contains $r_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps, for $i = 1, 2, \ldots, m$; with this notation, the initial configuration of the system is $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$.

A *computation* starts in the initial configuration. In order to compute a function $f : \mathbb{N} \to \mathbb{N}$ (functions of the kind $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$, for any fixed pair of integers $\alpha \geq 1$ and $\beta \geq 1$, can also be computed by using appropriate bijections from $\mathbb{N}^\alpha$ and $\mathbb{N}^\beta$ to $\mathbb{N}$), a positive integer number is given in input to a specified *input neuron*. In the original model, as well as in some early variants, the number is encoded as the interval of time steps elapsed between the insertion of two spikes into the neuron. To pass from a configuration to another one, for each neuron a rule is chosen among the set of applicable rules, and is executed. Generally, a computation may not halt. However, in any case the output of the system is considered to be the time elapsed between the arrival of two spikes in a designated *output cell*. Other possibilities exist to encode input and output numbers, as discussed in [10]: as the number of spikes contained in a given neuron at the beginning (resp., the end) of the computation, as the number of spikes fired in a given interval of time, etc.

A useful extension to the standard model defined above, already considered in [15, 16, 17, 12], is to use several input neurons, so that the introduction of the encoding of an instance of the problem to be solved can be done in a faster way, introducing parts of the code in parallel in various input neurons. Formally, we can define an SN P system of degree (m, ℓ), with $m \geq 1$ and $0 \leq \ell \leq m$, just like a standard SN P system of degree m, the only difference being that now there are ℓ input neurons denoted by $in_1, \ldots, in_\ell$. A *valid input* for an SN P system of degree (m, ℓ) is a set of ℓ binary sequences, that collectively encode an instance of a problem.

The previous definitions cover many types of systems/behaviors. By neglecting the output neuron we can define *accepting* SN P systems, in which the natural number (or the vector of natural numbers, in the case of systems having $\ell > 1$ input neurons) given in input is accepted if the computation halts. On the other hand, by ignoring the input neuron (and thus starting from a predefined input configuration)

we can define *generative* SN P systems. In [11] it was shown that generative SN P systems are universal, that is, can generate any recursively enumerable set of natural numbers. Moreover, a characterization of semilinear sets was obtained by spiking neural P systems with a bounded number of spikes in the neurons. These results can be obtained also for some restricted forms of SN P systems: [9] shows that one of the following features can be avoided while keeping universality: time delay greater than 0, forgetting rules, outdegree of the synapse graph greater than 2, and regular expressions of complex form. In [6] it is shown that universality is kept even if we remove some combinations of two of the above features. Finally, in [29] the behavior of SN P systems on infinite strings and the generation of infinite sequences of 0 and 1 was investigated, whereas in [3] SN P systems were studied as language generators (over the binary alphabet $\{0, 1\}$).

Spiking neural P systems can also be used to solve decision problems, both in a *semi–uniform* and in a *uniform* way. When solving a problem Q in the semi–uniform setting, for each specified instance $\mathcal{I}$ of Q we build an SN P system $\Pi_{Q,\mathcal{I}}$, whose structure and initial configuration depend upon $\mathcal{I}$, that halts (or emits a specified number of spikes in a given interval of time) if and only if $\mathcal{I}$ is a positive instance of Q. On the other hand, a uniform solution of Q consists in a family $\{\Pi_Q(n)\}_{n\in\mathbb{N}}$ of SN P systems such that, when having an instance $\mathcal{I} \in Q$ of size n, we introduce a polynomial (in n) number of spikes in a designated (set of) input neuron(s) of $\Pi_Q(n)$ and the computation halts (or, alternatively, a specified number of spikes is emitted in a given interval of time) if and only if $\mathcal{I}$ is a positive instance. The preference for uniform solutions over semi–uniform ones is given by the fact that they are more strictly related to the structure of the problem, rather than to specific instances. If the instances of a problem Q depend upon two parameters (as is the case of Subset Sum, where $n + 1$ is the number of integer values contained into the generic instance $(V = \{v_1, v_2, \ldots, v_n\}, S)$, and k is the number of bits needed to represent each of these values), then we will denote the family of SN P systems that solves Q by $\{\Pi_Q(\langle n, k\rangle)\}_{n,k\in\mathbb{N}}$, where $\langle n, k\rangle$ indicates the positive integer number obtained by applying an appropriate bijection (for example, Cantor's pairing) from $\mathbb{N}^2$ to $\mathbb{N}$.

The present paper considers SN P systems for solving decision problems, continuing the papers [17], [16] and [15], where we dealt with the **NP**-complete decision problems Subset Sum, sat and 3-sat. For all these problems, constant time and polynomial time solutions were provided by using SN P systems constructed both in the semi-uniform and in the uniform setting, working in a non-deterministic way, and also using a series of ingredients added to SN P systems of the standard form: rules that produce several spikes at a time, the possibility to have a choice between spiking rules and forgetting rules, forgetting rules controlled by regular expressions, rules applied in the maximal parallel way, etc. Here we consider a different situation: we assume that a pre-computed (standard) SN P system is given in advance, possibly having an exponential size with respect to the size of the instances of the problem we want to solve, and we provide a semi–uniform and a uniform constructions that solve Subset Sum in a polynomial time. All the

systems we will propose work in a *deterministic* way. Note that this setting was already considered in [12], where polynomial time uniform solutions to SAT and 3-SAT were provided.

An important observation is that we will not specify how our pre-computed systems could be built. However, we require that such systems have a *regular* structure, and that they do not contain neither "hidden information" that simplify the solution of specific instances, nor an encoding of all possible solutions (that is, an exponential amount of information that allows to cheat while solving the instances of the problem). These requirements were inspired by open problem Q27 in [27]. Let us note in passing that the regularity of the structure of the system is related to the concept of *uniformity*, that in some sense measures the difficulty of constructing the system. For example, when considering families $\{C(n)\}_{n\in\mathbb{N}}$ of boolean circuits, or other computing devices whose number of inputs depends upon an integer parameter $n \geq 1$, it is required that for each $n \in \mathbb{N}$ a "reasonable" description (see [2] for further discussion on the meaning of the term "reasonable" in this context) of $C(n)$, the circuit of the family which has n inputs, can be produced in polynomial time and logarithmic space (with respect to n) by a deterministic Turing machine whose input is 1^n, the unary representation of n. In this paper we will not delve further into the details concerning uniformity; we just rely on reader's intuition, by stating that it should be possible to build the entire structure of the system using only a polynomial amount of information and a controlled replication mechanism, as it already happens in P systems with cell division.

The paper is organized as follows. In section 2 we recall the definition of the SUBSET SUM problem, as well as a classical solution algorithm based on the dynamic programming paradigm. In section 3 we elaborate such an algorithm to obtain a family of SN P systems that solves SUBSET SUM in a semi–uniform way. In section 4 we propose a completely different construction, that allows to uniformly solve all the instances of SUBSET SUM of any specified size; the instances are provided in input to the systems of the family by specifying their values in binary form. Finally, section 5 contains the conclusions and some directions for further research.

2 The SUBSET SUM Problem

SUBSET SUM is one of the most known **NP**-complete decision problems. We can state it as follows, in a form which is equivalent to the one given in [7, p. 223].

Problem 1. NAME: SUBSET SUM.

- INSTANCE: a (multi)set $V = \{v_1, v_2, \ldots, v_n\}$ of positive integer numbers, and a positive integer number S.
- QUESTION: is there a sub(multi)set $B \subseteq V$ such that $\sum_{b\in B} b = S$?

The following well known algorithm [5] solves SUBSET SUM by using the Dynamic Programming technique. In particular, the algorithm returns 1 on positive instances, and 0 on negative instances.

```
SUBSET SUM({v_1, v_2, ..., v_n}, S)
for j ← 0 to S
    do M[1, j] ← 0
M[1, 0] ← M[1, v_1] ← 1
for i ← 2 to n
     do for j ← 0 to S
              do M[i, j] ← M[i − 1, j]
                  if j ≥ v_i and M[i − 1, j − v_i] > M[i, j]
                     then M[i, j] ← M[i − 1, j − v_i]
return M[n, S]
```

In order to look for a subset $B \subseteq V$ such that $\sum_{b \in B} b = S$, the algorithm uses an $n \times (S + 1)$ matrix M whose entries are from $\{0, 1\}$. It fills the matrix by rows, starting from the first row. Each row is filled from left to right. The entry $M[i, j]$ is filled with 1 if and only if there exists a subset of $\{v_1, v_2, \ldots, v_i\}$ whose elements sum up to j. The given instance of SUBSET SUM is thus a positive instance if and only if $M[n, S] = 1$ at the end of the execution.

Since each entry is considered exactly once to determine its value, the time complexity of the algorithm is proportional to $n(S+1) = \Theta(nS)$. This means that the difficulty of the problem depends on the value of S, as well as on the magnitude of the values in V. In fact, let $K = max\{v_1, v_2, \ldots, v_n, S\}$. If K is polynomially bounded with respect to n, then the above algorithm works in polynomial time. On the other hand, if K is exponential with respect to n, say $K = 2^n$, then the above algorithm may work in exponential time and space. This behavior is usually referred to in the literature by telling that SUBSET SUM is a *pseudo–polynomial* **NP**–complete problem.

The fact that in general the running time of the above algorithm is not polynomial can be immediately understood by comparing its time complexity with the instance size. The usual size for the instances of SUBSET SUM is $\Theta(n \log K)$, since for conciseness every "reasonable" encoding is assumed to represent each element of V (as well as S) using a string whose length is $O(\log K)$. Here all logarithms are taken with base 2. Stated differently, the size of the instance is usually considered to be the number of bits which must be used to represent in binary S and all the integer numbers which occur in V. If we would represent such numbers using the unary notation, then the size of the instance would be $\Theta(nK)$. But in this case we could write a program which first converts the instance in binary form and then uses the above algorithm to solve the problem in polynomial time with respect to the new instance size. We can thus conclude that the difficulty of a numerical **NP**–complete problem depends also on the measure of the instance size we adopt. Indeed, SUBSET SUM is not **NP**-complete in the *strong sense*, meaning that it does

not remain **NP**-complete when we represent its instances in unary form [7]. Stated otherwise, strongly **NP**-complete problems remain **NP**-complete even when the numbers contained into their instances are small.

As a consequence of these observations, the SN P systems that we will consider in section 4 will take in input the instances of SUBSET SUM as $n + 1$ strings encoded in binary form, where the length of each string will be $k = \log K$. Before presenting the uniform solution of section 4, in the next section we first elaborate the above dynamic programming algorithm to provide a semi–uniform family of SN P systems that solves the SUBSET SUM problem.

3 A Semi–uniform Solution to SUBSET SUM

Let $SS(n, k)$ denote the set of instances of SUBSET SUM which can be built by using $n + 1$ positive k-bit integer numbers. In this section we present a semi–uniform family $\{\Pi(\mathcal{I})\}_{\mathcal{I} \in SS(n,k)}$ of SN P systems such that for every $\mathcal{I} \in SS(n, k)$ the system $\Pi(\mathcal{I})$ determines whether $\mathcal{I} = (\{v_1, v_2, \ldots, v_n\}, S)$ is a positive instance of SUBSET SUM. The size of $\Pi(\mathcal{I})$ will be $\Theta(nS)$, hence exponential with respect to the instance size. However, the computation time of $\Pi(\mathcal{I})$ will be linear in n and independent of k.

System $\Pi(\mathcal{I})$ is depicted in Figure 1 in a schematic way. The system is composed by n layers, horizontally arranged, one for each iteration of the dynamic programming algorithm illustrated in the previous section. The computation starts in the first (the uppermost) layer, and proceeds downwards until the lowest (i.e., the n-th) layer has been reached. The neurons of the first layer contain the firing rule $a \to a; 0$, that propagates the spikes eventually contained in these neurons to the appropriate neurons of the second layer. All the other neurons, from layer 2 down to layer n, contain two firing rules:

$$a \to a; 0 \qquad \text{and} \qquad a^2 \to a; 0$$

that make the neurons operate like OR boolean gates.

The connections among the neurons depend upon the instance $\mathcal{I} = (\{v_1, v_2, \ldots, v_n\}, S)$ of SUBSET SUM to be solved. Precisely, to determine the value of $M[i, j]$ in the above algorithm we need to compute the maximum between the values $M[i-1, j]$ and $M[i-1, j-v_i]$, provided that $j - v_i \geq 0$, otherwise we put $M[i, j]$ equal to $M[i-1, j]$. The rationale behind these formulas is the following: as stated above, $M[i, j]$ has to be set to 1 if and only if there exists a subset of $\{v_1, v_2, \ldots, v_i\}$ such that the sum of its elements is equal to j. Thus we have two possibilities: either the subset contains v_i, or not. In the former case, there must be a subset of $\{v_1, v_2, \ldots, v_{i-1}\}$ such that the sum of its elements is equal to $j - v_i$ (that is, $M[i-1, j-v_i]$ must be 1); in the latter case, there must be a subset of $\{v_1, v_2, \ldots, v_{i-1}\}$ whose elements sum up to j (that is, $M[i-1, j] = 1$). If $j < v_i$ then clearly v_i cannot be in any subset of $\{v_1, v_2, \ldots, v_i\}$ whose sum is equal to j, and thus in this case we only check the value of $M[i-1, j]$. If $i = 1$ then these

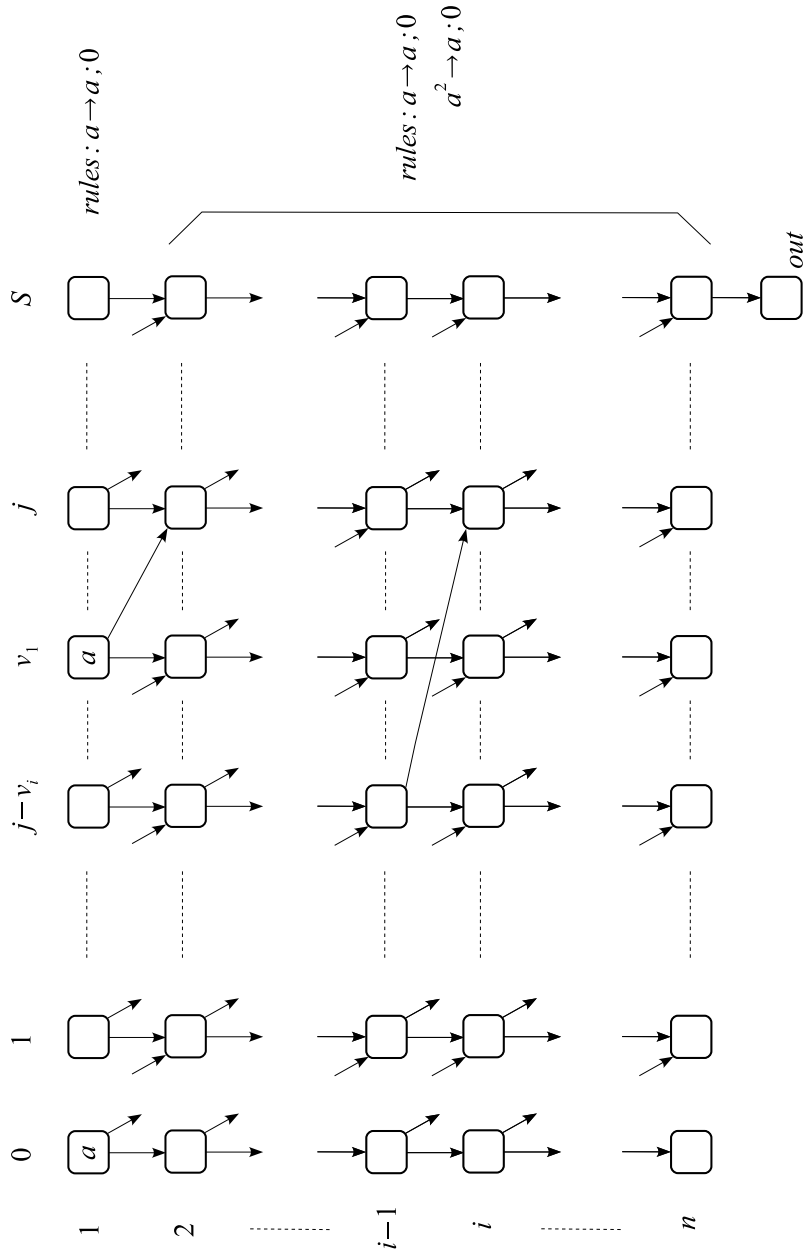


Fig. 1. A schematic view of the system $\Pi(\mathcal{I})$ used to solve a specified instance $\mathcal{I} = (\{v_1, v_2, \ldots, v_n\}, S)$ of Subset Sum, where each of the values $v_1, v_2, \ldots, v_n, S$ is a k-bit positive integer number

formulas cannot clearly be applied. However, we note that the only two subsets of $\{v_1\}$ we can build are the empty set $\emptyset$ and $\{v_1\}$ itself, hence $M[1,0] = M[1,v_1] = 1$ whereas $M[1,j] = 0$ for all $j \notin \{0, v_1\}$. Since the admissible values of $M[i-1,j]$ and of $M[i-1, j-v_i]$ are 0 and 1, computing the maximum is the same as computing a logical OR. In the system depicted in Figure 1, the j-th neuron from the left, $0 \leq j \leq S$, corresponds to $M[i,j]$. We denote 1 (resp., 0) by the presence (resp., absence) of a spike. Such a neuron, for $1 \leq i \leq n$, has a synapse going to the

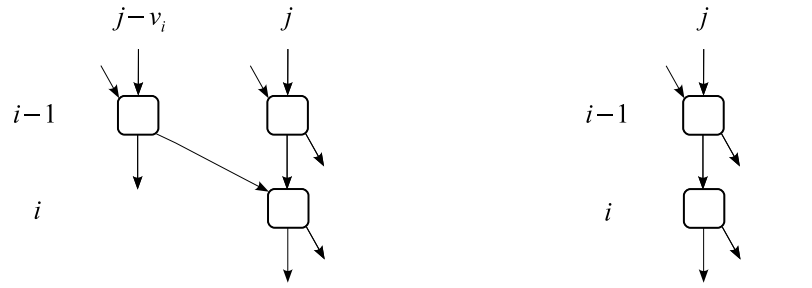


Fig. 2. The two cases to be considered to compute the value of $M[i,j]$

neuron that corresponds to $M[i+1,j]$, and possibly (if $v_{i+1} + j \leq S$) another synapse going to the neuron that corresponds to $M[i+1, j+v_{i+1}]$ (see Figure 2). In the last layer, only the neuron that corresponds to $M[n,S]$ has a synapse going to a neuron named *out*, which is the output neuron and does not contain any rule.

In the initial configuration of the system, one spike is put in the neurons that correspond to $M[1,0]$ and $M[1,v_1]$; all the other neurons are empty. During the i-th computation step, with $1 \leq i \leq n-1$, the neurons in the i-th layer perform their computation, and send the corresponding result to the appropriate neurons of the next layer. At the n-th computation step, all the neurons in the last layer send the spikes produced by them to the environment (where they are lost) but the rightmost neuron, that sends the result of its computation (0 or 1 spikes) to neuron *out*. Hence, the instance $\mathcal{I}$ of SUBSET SUM represented by the structure and the initial configuration of $\Pi(\mathcal{I})$ is positive if and only if one spike arrives in neuron *out* during the n-th computation step. After the result of the computation (0 or 1 spikes in neuron *out*) has been produced, the computation halts and the spike eventually contained in neuron *out* remains there. The computation time of $\Pi(\mathcal{I})$ is linear in n, independent of the values $v_1, v_2, \ldots, v_n$ and S contained in $\mathcal{I}$, but the number of neurons in the system is $n(S+1)+1$, which is exponential with respect to the instance size. This last fact would be considered unacceptable in traditional complexity theory, but recall that in this paper (as well as in [12]) we

are assuming that exponential size resources — encoded in exponential size SN P systems of regular structure — are admitted.

The structure of $\Pi(\mathcal{I})$ is indeed very regular: all the instances composed by n integer values plus a required sum equal to S produce systems having n layers, each composed by $S+1$ neurons. The values $v_1, v_2, \ldots, v_n$ determine some of the connections between the neurons (all the other connections go from every neuron in each layer to the neuron that occurs in the same position in the next layer); precisely, for all $i \in \{1, 2, \ldots, n-1\}$ the value v_i determines the presencce of a synapse from every j-th neuron in layer i, such that $j + v_{i+1} \leq S$, to the $(j+v_{i+1})$-th neuron of layer $i+1$. Value v_1 also determines the neuron in the first layer (apart from the leftmost) that will receive one spike in the initial configuration. An open question, that we will not address in this paper, is: what kind of operations are needed to augment the power of deterministic Turing machines so that, given any instance $\mathcal{I}$ of Subset Sum, the new machine is able to produce a "reasonable" description of $\Pi(\mathcal{I})$ in a polynomial time? Note that in this case we should also recast the meaning of the term "reasonable", since in [7] this notion concerns only polynomial size constructions.

4 A Uniform Solution to Subset Sum

Let us present now a uniform family $\{\Pi(\langle n, k\rangle)\}_{n,k\in\mathbb{N}}$ of SN P systems that solves the Subset Sum problem in a uniform way. Precisely, for all $n, k \in \mathbb{N}$ the system $\Pi(\langle n, k\rangle)$ will solve all the instances $\mathcal{I} \in SS(n, k)$ which are composed by $n+1$ positive k-bit integer numbers. Such instances are provided in input in binary form, as a sequence of $(n+1)k$ bits that are fed to the system in parallel (which means that each bit is inserted into an appropriate input neuron).

Figure 3 depicts the system $\Pi(\langle n, k\rangle)$ in a schematic way. The instance $\mathcal{I} \in SS(n, k)$ is inserted into the leftmost neurons, which are labelled with a name that indicates the bit which has to be inserted. These neurons simply propagate their spikes to subsystems $\textsc{Sum}_1$, $\textsc{Sum}_2$, ..., $\textsc{Sum}_{2^n-1}$ by using a firing rule of type $a \to a; 0$. The Sum subsystems are bijectively associated to every possible non-empty subset of $\{v_1, v_2, \ldots, v_n\}$. As the name indicates, every Sum subsystem computes the sum of the elements of the corresponding subset of $\{v_1, v_2, \ldots, v_n\}$, and thus the synapses outgoing from the leftmost neurons reflect this situation; that is, a synapse leaving from neuron $v_{i,j}$, $1 \leq i \leq n$ and $1 \leq j \leq k$, reaches the subsystem $\textsc{Sum}_\ell$ if and only if value v_i is involved in the sum computed by $\textsc{Sum}_\ell$. The sums are computed in binary (we will return later on this point) and hence every Sum subsystem produces a bit vector as a result. This vector is then compared with the sequence of bits that compose the value S; the comparison is performed by the Compare subsystems, that produce a 1 (that is, a spike) if and only if the two sequences given in input are equal. Recall that two integer numbers expressed in binary form are equal if and only if their binary expansions are equal; the comparison thus amounts to compute the following boolean function:

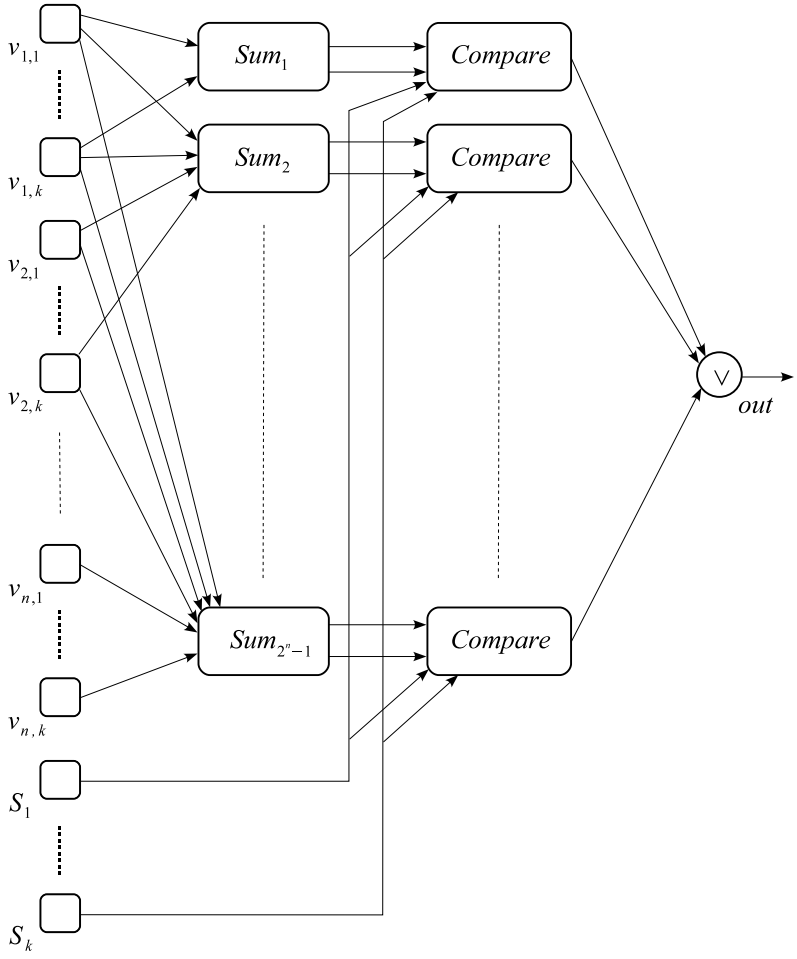


Fig. 3. A schematic view of the system $\Pi(\langle n, k\rangle)$ used to solve all the instances of SUBSET SUM which are composed by $n+1$ positive k-bit integer numbers

$$\text{COMPARE}(x_0, \ldots, x_{k-1}, y_0, \ldots, y_{k-1}) = \bigwedge_{i=0}^{k-1} \left(\neg(x_i \oplus y_i)\right) = \neg\left(\bigvee_{i=0}^{k-1} (x_i \oplus y_i)\right)$$

where $x = \sum_{i=0}^{k-1} x_i 2^i$ and $y = \sum_{i=0}^{k-1} y_i 2^i$ are the numbers to be compared, and $\vee, \wedge, \neg, \oplus$ denote the OR, AND, NOT and XOR (also PARITY) logical connectives, respectively. Figure 4 shows an SN P (sub)system which can be used to compute this function. This subsystem works as follows. Bits x_i and y_i are XORed by the neurons depicted on the left of the figure. The neuron labelled with $\vee$ computes the logical OR of its inputs: precisely, it emits one spike if and only if at least one spike enters into the neuron. Neuron *res* receives the output produced by $\vee$ and

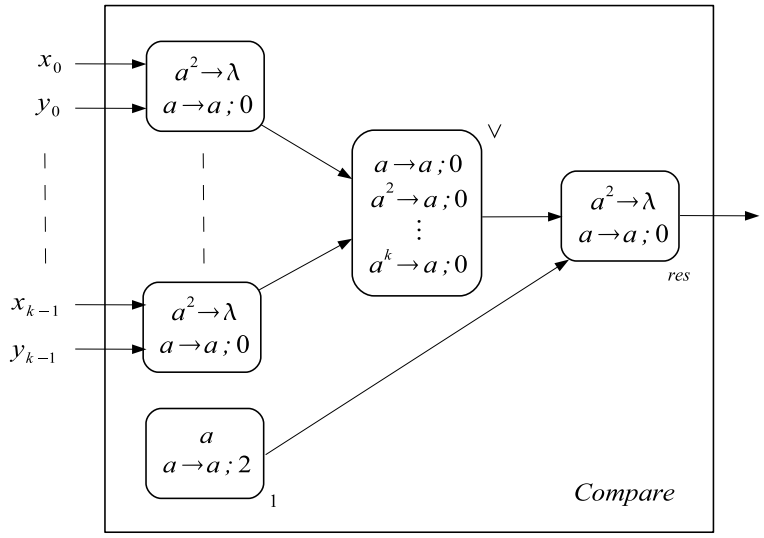


Fig. 4. The subsystem of $\Pi(\langle n, k\rangle)$ that compares two k-bit natural numbers

computes its logical negation (NOT). In order to be able to produce one spike if no spikes come from res, we use one auxiliary neuron that sends to res one spike after two computation steps. Indeed, the delay of the rule contained in neuron 1 (whose contents will be initialized with one spike at the beginning of the computation) should be set in order to make neuron 1 fire exactly when the results computed by the SUM subsystems reach the COMPARE subsystems (plus two steps).

Observe that S is a k-bit number, just like $v_1, v_2, \ldots, v_n$, and thus if we sum a subset of these latter values we could easily end up with a result that needs more than k bits to be expressed in binary form, thus complicating a little bit the comparison with S. However, recall that $k = \log K$ where $K = \max\{v_1, v_2, \ldots, v_n, S\}$, and thus for reasonable values it is very likely that a large portion of the most significant bits of $v_1, v_2, \ldots, v_n$ is equal to zero. Anyway, just to be cautious, since the COMPARE subsystems perform a k-bit comparison, we should avoid the situation in which a SUM subsystem produces an m-bit sequence, with $m > k$, such that its k less significant bits coincide with the bits that compose S. Fortunately it is easy to check whether we are in this situation: we just design each of the SUM subsystems so that it produces an m-bit sequence, where $m = k + \lceil \log_2 n \rceil$ (in facts the maximum integer number that we can represent using k bits is $2^k - 1$, so if we sum n of such numbers we obtain a result which is less than $n2^k$, that requires $k + \lceil \log_2 n \rceil$ bits to be represented in binary form), and we check that the $m - k$ most significant bits of this sequence are all zero. This is easily done by sending these bits (that is, the corresponding spikes) to a neuron whose contents (the presence of at least one spike) signals to the user of the system that the above situation occured.

The core of the system is composed by the SUM subsystems. In a generic SUM subsystem, r values from the set $\{v_1, v_2, \ldots, v_n\}$ have to be summed together, and this sum has to be performed in polynomial time. If $r = 2$ then we can use either a traditional or a carry look-ahead adder [32, p. 6]. Let $x = \sum_{i=0}^{k-1} x_i 2^i$ and $y = \sum_{i=0}^{k-1} y_i 2^i$ be the two k-bit binary numbers to be summed. We denote by $s_0, s_1, \ldots, s_k$ the bits of the sum, and by $c_0, c_1, \ldots, c_k$ the carries generated during the addition. The traditional addition algorithm (which can be trivially implemented using a boolean circuit) puts $s_0 = x_0 \oplus y_0$, $c_0 = 0$, and then defines inductively $c_i = (x_{i-1} \wedge y_{i-1}) \vee (x_{i-1} \wedge c_{i-1}) \vee (y_{i-1} \wedge c_{i-1})$ (that is, $c_i = 1$ if and only if at least two of x_{i-1}, y_{i-1}, c_{i-1} is 1), $s_i = x_i \oplus y_i \oplus c_i$ for $1 \leq i < k$, and $s_k = c_k$. Such an algorithm sums the two k-bit integer numbers in $O(k)$ steps.

A carry look-ahead adder operates by computing the values of the carries c_i in a finite number of steps, indepedent of k, starting from the values of $x_0, x_1, \ldots, x_{k-1}$ and $y_0, y_1, \ldots, y_{k-1}$. The crucial observation is that a carry is generated at position i if and only if both input bits x_i and y_i are 1, and a carry is eliminated at position i if and only if both input bits x_i and y_i are 0. This observation yields to the following definitions: for $0 \leq i < k$, let:

$$g_i = x_i \wedge y_i \qquad \text{(position } i \textit{ generates} \text{ a carry)}$$
$$p_i = x_i \vee y_i \qquad \text{(position } i \textit{ propagates} \text{ a carry)}$$

Now, a carry ripples into position i if and only if there exists a position $j < i$ where a carry is generated, and all positions in between propagate it. Formally:

$$c_i = \bigvee_{j=0}^{i-1} \left(g_j \wedge \bigwedge_{k=j+1}^{i-1} p_k \right) \qquad \text{for } 1 \leq i \leq k \tag{1}$$

Once we have computed the carries, the bits of the sum are computed as before: $s_0 = x_0 \oplus y_0$, $s_i = x_i \oplus y_i \oplus c_i$ for $1 \leq i < k$, and $s_k = c_k$. It is easily seen that the above formulas allow to compute all the c_i in parallel, since they only depend on the input bits $x_0, x_1, \ldots, x_{k-1}$ and $y_0, y_1, \ldots, y_{k-1}$, in constant time: all g_i and p_i are computed in one step, and two more steps are needed to compute the ANDs and the ORs that appear in (1). By using XOR ($\oplus$) gates, all the bits of the sum are computed in one more step.

The boolean circuit that implements a carry look-ahead adder can be easily simulated by an SN P system, simply substituting every logical gate with an appropriate neuron. Figure 5 shows this mapping from AND, OR and XOR gates to neurons. When needed, for example when the output value of a gate has to skip one or more layers and go directly to one of the subsequent layers, for synchronization purposes we can also use delay neurons, that contain the rule $a \to a; d$ for an appropriate value of d. It is clear that the size of the SN P system thus obtained is polynomially related with the size of the simulated boolean circuit, and that if the simulated circuit performs its computations in constant time then also the SN P system performs its computations in constant time.

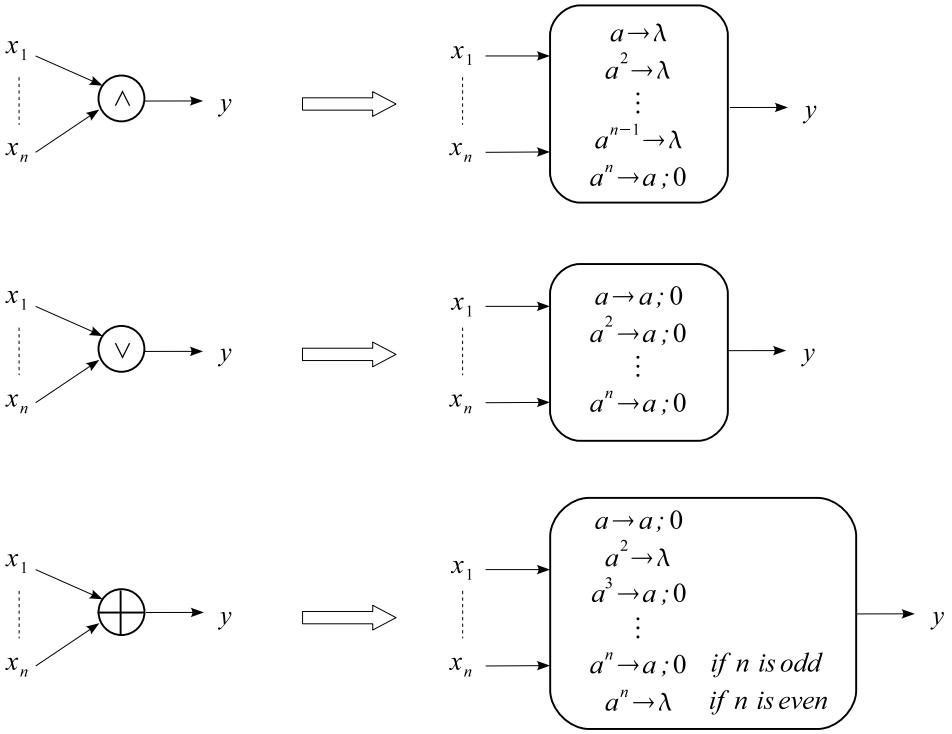


Fig. 5. Simulation of n-input AND, OR and XOR gates by means of single–neuron SN P systems

If we need to compute the sum of $r > 2$ binary numbers of length k, then a slightly more complicated construction is needed. As shown in [32, p. 13], while designing a boolean circuit that computes the ITERATED ADDITION (that is, the sum of n natural numbers, each of n bits), the addition of three k-bit binary numbers $a = \sum_{i=0}^{k-1} a_i 2^i$, $b = \sum_{i=0}^{k-1} b_i 2^i$ and $c = \sum_{i=0}^{k-1} c_i 2^i$ can be reduced to the addition of two $(k+1)$-bit numbers e and d, by defining:

$$
\begin{aligned}
& e_0 = 0 \\
& e_i = (a_{i-1} \wedge b_{i-1}) \vee (a_{i-1} \wedge c_{i-1}) \vee (b_{i-1} \wedge c_{i-1}) \qquad \text{for all } 1 \leq i \leq k \\
& d_i = a_i \oplus b_i \oplus c_i \qquad \text{for } 0 \leq i < k \\
& d_k = 0
\end{aligned}
$$

The rationale behind these formulas is the following. If we look at a single position i, then we have to add a_i, b_i and c_i. The result is given by the two bit number $e_{i+1}d_i$; bit e_{i+1} is 1 if and only if at least two of the bits a_i, b_i and c_i are 1, and $d_i = 1$ if and only if an odd number of a_i, b_i and c_i is 1. We can thus conclude that $a + b + c = d + e$.

If we are given $r > 2$ binary numbers of length k, we can group them into three elements sets (plus one set with only one or two numbers, if r is not a multiple of 3), and then compute for each set as just explained two numbers whose sum is equal to the sum of all the three numbers from the set. In this way we end up with r' numbers of $k + 1$ bits each, where:

$$r' = \begin{cases} \frac{2}{3}\, r & \text{if } r \equiv 0 \mod 3 \\ \frac{2}{3}\,(r-1)+1 & \text{if } r \equiv 1 \mod 3 \\ \frac{2}{3}\,(r-2)+2 & \text{if } r \equiv 2 \mod 3 \end{cases}$$

In any case, if $r > 2$ then $r' \leq \frac{4}{5}\, r$. Thus, given r numbers of k bits each, by iterating this reduction procedure $O(\log r)$ times we end up with two numbers of $k + O(\log r)$ bits each. These two numbers can then be added using a carry look-ahead adder, as explained above. In the worst case, we have to add all the numbers from $\{v_1, v_2, \ldots, v_n\}$. The reduction process can thus be implemented by a $O(\log n)$ depth boolean circuit, since each reduction involves a constant depth (and bounded fan-in) circuit. At the end of the reduction process we have to add two $(k + O(\log n))$-bit numbers, which can be done by a boolean circuit of polynomial (quadratic in $k + O(\log n)$) size and constant depth. The fan-in of such a circuit is unbounded, and thus also the in-degree of the neurons of the SN P system that simulates it is unbounded. However, any unbounded fan-in AND or OR gate can be simulated by a polynomial size logarithmic depth circuit composed by bounded fan-in AND and OR gates, and thus we can conclude that the SUM subsystems can be implemented by polynomial size SN P systems which are composed by a logarithmic number of layers and whose in-degree is bounded (that is, constant). The same argumentation holds for the COMPARE subsystems: they can be implemented as polynomial size logarithmic depth OR/XOR circuits of bouded fan-in, and hence as polynomial size SN P systems composed by a logarithmic number of layers, each composed by constant in-degree neurons. Finally, the large OR that provides the output to the environment has $2^n - 1$ inputs, and thus it can be realized as an exponential size polynomial depth tree of bounded fan-in OR gates.

The system $\Pi(\langle n, k\rangle)$ thus obtained is able to solve all the instances $\mathcal{I} \in SS(n, k)$ of SUBSET SUM which can be expressed as sequences of $n + 1$ natural numbers, each of k bits. The family $\{\Pi(\langle n, k\rangle)\}_{n,k\in\mathbb{N}}$ thus constitutes a uniform solution to the SUBSET SUM problem. The size of $\Pi(\langle n, k\rangle)$ is exponential with respect to the instance size, but the computation time it takes to determine whether the instance $\mathcal{I} \in SS(n, k)$ is positive or not is polynomial with respect to n and k. The fact that $\mathcal{I}$ is a positive instance is signalled by the emission of a spike from neuron *out*; in any case, after computing the solution the system halts. An important observation is that the system $\Pi(\langle n, k\rangle)$ has a very regular structure, and hence also in this case we can assume that it can be built in a polynomial time by a deterministic Turing machine whose computational power has been augmented by adding some controlled duplication instruction. Just like in the case of

the semi–uniform solution illustrated in the previous section, it is an open problem to determine how precisely this controlled duplication instruction should work.

5 Conclusions and Directions for Future Research

We have proposed two families of spiking neural P systems that solve Subset Sum, the well known **NP**-complete decision problem. The peculiarity and importance of Subset Sum, while trying to assess the computational power of a new computational device, is that it is a *numerical* **NP**-complete problem, and the difficulty of solving it depends upon the magnitude of the integer numbers that appear in its instances. To be precise it is not **NP**-complete in the strong sense, and hence the problem becomes easy to solve (through a well known algorithm which is based on the dynamic programming paradigm) when the numbers contained into the instances are small; equivalently, we can say that it becomes easy to solve when its instances are expressed in unary form.

For this reason, after showing in section 3 how for any instance of Subset Sum an SN P system that solves it can be built (thus working in the so called semi–uniform setting), in section 4 we have illustrated a uniform solution. Precisely, we have defined a family $\{\Pi(\langle n,k\rangle)\}_{n,k\in\mathbb{N}}$ of SN P systems such that for all $n,k\in\mathbb{N}$ the system $\Pi(\langle n,k\rangle)$ solves all the instances $\mathcal{I}\in SS(n,k)$ which are composed by $n+1$ positive k-bit integer numbers. The system $\Pi(\langle n,k\rangle)$ performs its computations in a time which is polynomial in n and k, but its size generally grows exponentially with respect to these parameters. However the structure of $\Pi(\langle n,k\rangle)$ is so regular that we can assume that the system may be built in a polynomial time by a deterministic Turing machine whose computational power has been augmented by adding to its set of instructions some form of controlled duplication, that replicates (possibly substituting some pieces of the structure) part of the output it has built up to that moment.

It is important to note that, as proved in [16], an SN P system of polynomial size cannot solve in a deterministic way and in a polynomial time an **NP**-complete problem (unless $\mathbf{P}=\mathbf{NP}$), hence efficient solutions to **NP**-complete problems cannot be obtained without introducing features which enhance the efficiency (pre-computed resources, ways to exponentially grow the workspace during the computation, non-determinism, and so on). A more careful examination of such features – in particular, possible relations with the well known notions of *uniformity* traditionally studied in the theory of circuit complexity – is a research direction of a clear interest.

Acknowledgements

The ideas exposed in this paper emerged during the Sixth Brainstorming Week on Membrane Computing, held in Seville from February 4 to February 8, 2008.

The first author wishes to acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER

funds, and the support of the project of excellence TIC-581 of the Junta de Andalucía.

The work of both authors was partially supported by the project "Azioni Integrate Italia-Spagna - Theory and Practice of Membrane Computing" (Acción Integrada Hispano-Italiana HI 2005-0194).

References

1. A. Alhazov, M.J. Pérez-Jiménez. Uniform solution to QSAT using polarizationless active membranes. In M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero (eds.), *Fourth Brainstorming Week on Membrane Computing*, RGCN Report 02/2006, Sevilla University, Fénix Editora, Vol. I, 29–40.
2. J.L. Balcázar, J. Díaz, J. Gabarró. *Structural Complexity*. Voll. I and II, Springer-Verlag, Berlin, 1988–1990.
3. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez. On string languages generated by spiking neural P systems. In M.A. Gutiérrez–Naranjo, Gh. Păun, A. Riscos–Núñez, F.J. Romero–Campero (eds.), *Fourth Brainstorming Week on Membrane Computing*, RGCN Report 02/2006, Sevilla University, Fénix Editora, Vol. I, 169–194.
4. H. Chen, M. Ionescu, T.-O. Ishdorj. On the efficiency of spiking neural P systems. *Proc. 8th Intern. Conf. on Electronics, Information, and Communication*, Ulanbator, Mongolia, June 2006, 49–52.
5. T.H. Cormen, C.H. Leiserson, R.L. Rivest, *Introduction to Algorithms*. MIT Press, Boston, 1990.
6. M. García-Arnau, D. Peréz, A. Rodríguez-Patón, P. Sosík. Spiking Neural P Systems. Stronger Normal Forms. In M.A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez, A. Riscos-Núñez (eds.), *Fifth Brainstorming Week on Membrane Computing*, RGCN Report 01/2007, Sevilla University, Fénix Editora, 157–178.
7. M.R. Garey, D.S. Johnson. *Computers and Intractability. A Guide to the Theory on NP–Completeness*. W.H. Freeman and Company, 1979.
8. W. Gerstner, W. Kistler. *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.
9. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosík, S. Woodworth. Normal Forms for Spiking Neural P Systems. *Theoretical Computer Science*, 372(2-3):196–217, 2007.
10. M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez. Computing with spiking neural P systems: Traces and small universal systems. In C. Mao, T. Yokomori, B.-T. Zhang (eds.), *DNA Computing,* 12^{th} *International Meeting on DNA Computing (DNA12)*, Revised Selected Papers, LNCS 4287, Springer-Verlag, Berlin, 1–16, 2006.
11. M. Ionescu, Gh. Păun, T. Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2-3):279–308, 2006.
12. T.-O. Ishdorj, A. Leporati. Uniform Solutions to SAT and 3-SAT by Spiking Neural P Systems with Pre-computed Resources. *Natural Computing*, in press. A preliminary version appeared as Turku Centre for Computer Science – TUCS Report No. 876, 2008.
13. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. A fast P system for finding a balanced 2–partition. *Soft Computing*, 9(9):673–678, 2005.

14. S.N. Krishna, R. Rama. A variant of P systems with active membranes: Solving **NP**-complete problems. *Romanian Journal of Information Science and Technology*, 2(4):357–367, 1999.
15. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M.J. Pérez-Jiménez. Uniform Solutions to SAT and Subset Sum by Spiking Neural P Systems. Submitted for publication, 2008.
16. A. Leporati, C. Zandron, C. Ferretti, G. Mauri. On the computational power of spiking neural P systems, *Intern. J. Unconventional Computing*, 2007, in press.
17. A. Leporati, C. Zandron, C. Ferretti, G. Mauri. Solving Numerical NP-complete Problems with Spiking Neural P Systems. In G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa (eds.) *Membrane Computing, International Workshop, WMC8*, Selected and Invited Papers, LNCS 4860, Springer-Verlag, Berlin, 336–352, 2007.
18. A. Leporati, C. Zandron, M.A. Gutiérrez-Naranjo. P systems with input in binary form. *International Journal of Foundations of Computer Science*, 17(1):127–146, 2006.
19. W. Maass. Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8(1):32–36, 2002.
20. W. Maass, C. Bishop (eds.). *Pulsed Neural Networks*, MIT Press, Cambridge (MA), 1999.
21. A. Obtulowicz. Deterministic P systems for solving SAT problem. *Romanian Journal of Information Science and Technology*, 4(1–2):551–558, 2001.
22. C.H. Papadimitriou. *Computational Complexity*, Addison-Wesley, 1994.
23. A. Păun, Gh. Păun. Small universal spiking neural P systems. *BioSystems*, 90(1):48–60, 2007.
24. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 1(61):108–143, 2000. See also Turku Centre for Computer Science – TUCS Report No. 208, 1998.
25. Gh. Păun. Computing with membranes. An introduction. *Bulletin of the EATCS*, 67:139–152, February 1999.
26. Gh. Păun. P systems with active membranes: Attacking **NP**-complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 2001.
27. Gh. Păun. *Membrane Computing. An Introduction.* Springer–Verlag, Berlin, 2002.
28. Gh. Păun, G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287(1):73–100, 2002.
29. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg. Infinite spike trains in spiking neural P systems. Submitted for publication.
30. M.J. Pérez-Jiménez, A. Riscos-Núñez. Solving the SUBSET SUM problem by active membranes. *New Generation Computing*, 23(4):367–384, 2005.
31. M.J. Pérez-Jiménez, A. Riscos-Núñez. A linear–time solution to the KNAPSACK problem using P systems with active membranes. In C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa (eds.), *Membrane Computing, 4th International Workshop, WMC 2003*, Revised Selected and Invited Papers, LNCS 2933, Springer-Verlag, Berlin, 250–268, 2004.
32. H. Vollmer, *Introduction to Circuit Complexity: A Uniform Approach*. Springer–Verlag, Berlin, 1999.
33. C. Zandron, C. Ferretti, G. Mauri. Solving **NP**-complete Problems Using P Systems with Active Membranes. In I. Antoniou, C.S. Calude, M.J. Dinneen (eds.), *Unconventional Models of Computation*, Springer-Verlag, Berlin, 289–301, 2000.
34. The P systems Web page: http://ppage.psystems.eu

A First Model for Hebbian Learning with Spiking Neural P Systems

Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mails: {magutier,marper}@us.es

Summary. *Spiking neural P systems* and *artificial neural networks* are computational devices which share a biological inspiration based on the transmission of information among neurons. In this paper we present a first model for Hebbian learning in the framework of Spiking Neural P systems by using concepts borrowed from neuroscience and artificial neural network theory.

1 Introduction

> *When an axon of cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*
>
> *D. O. Hebb (1949) [13]*

Neuroscience has been a fruitful research area since the pioneering work of Ramón y Cajal in 1909 [22] and after a century full of results on the man and the mind, many interesting questions are today open problems. Two of such problems of current neuroscience are the understanding of neural plasticity and the neural coding.

The first one, the understanding of neural plasticity, is related to the changes in the amplitude of the postsynaptic response to an incoming action potential. Electrophysiological experiments show that the response amplitude is not fixed over time. Since the 1970's a large body of experimental results on synaptic plasticity has been accumulated. Many of these experiments are inspired by Hebb's postulated (see above). In the integrate-and-fire formal spiking neuron model [9] and also in artificial neural networks [12] is usual to consider a factor w as a measure of the *efficacy* of the synapse from neuron to another.

The second one, the neural coding, is related to the way in which one neuron sends information to other ones. It is interested on the information contained in the spatio-temporal pattern of pulses and on the code used by the neurons to transmit information. This research area wonders how other neurons decode the signal or if the code can be read by external observers and understand the message. At present, a definite answer to these questions is not known.

The elementary processing units in the central nervous system are neurons which are connected to each other in an intricate pattern. Cortical neurons and their connections are packed into a dense network with more than 10^4 cell bodies per cubic millimeter. A single neuron in a vertebrate cortex often connects to more than 10^4 postsynaptic neurons.

The neuronal signals consist of short electrical pulses (also called action potentials or *spikes*) and can be observed by placing a fine electrode close to the soma or axon of a neuron. The junction between two neurons is a *synapse* and it is common to refer to the sending neuron as a presynaptic cell and to the receiving neuron as the postsynaptic cell.

Since all spikes of a given neuron look alike, the form of the action potential does not carry any information. Rather, it is the number and the timing of spikes which matter. Traditionally, it has been thought that most, if not all, of the relevant information was contained in the *mean* firing rate of the neuron. The concept of mean firing rates has been successfully applied during the last 80 years (see, e.g., [18] or [14]) from the pioneering work of Adrian [1, 2]. Nonetheless, more and more experimental evidence has been accumulated during recent years which suggests that a straightforward firing rate concept based on temporal averaging may be too simplistic to describe brain activity. One of the main arguments is that reaction times in behavioral experiment are often too short to allow long temporal averages. Humans can recognize and respond to visual scenes in less than 400ms [24]. Recognition and reaction involve several processing steps from the retinal input to the finger movement at the output. If at each processing steps, neurons had to wait and perform a temporal average in order to read the message of the presynaptic neurons, the reaction time would be much longer. Many other studies show the evidence of precise temporal correlations between pulses of different neurons and stimulus-dependent synchronization of the activity in populations of neurons (see, for example, [5, 11, 10, 6, 23]). Most of these data are inconsistent with a concept of coding by mean firing rates where the exact timing of spikes should play no role.

Instead of considering mean firing rates, we consider the realistic situation in which a neuron abruptly receives an input and for each neuron the timing of the first spike after the reference signal contains all the information about the new stimulus.

Spiking neural P systems (SN P systems, for short) were introduced in [15] with the aim of incorporating in membrane computing[1] ideas specific to spike-based

[1] The foundations of membrane computing can be found in [20] and updated bibliography at [25].

neuron models. The intuitive goal was to have a directed graph were the nodes represent the neurons and the edges represent de synaptic connections among the neurons. The flow of information is carried on the action potentials, which are encoded by objects of the same type, the *spikes*, which is placed inside the neurons and can be sent from presynaptic to postsynaptic neurons according to specific rules and making use of the time as a support of information.

This paper is a first answer to the question proposed by Gh. Păun in [21] related to link the study of SN P systems with neural computing and as he suggests, the starting point has been not only neural computing, but also recent discoveries in neurology.

The paper is organized as follows: first we discuss about SN P systems with input and delay and a new computational device called Hebbian SN P system unit is presented. In section 3 we present our model of learning with SN P systems based on Hebb's postulate. An illustrative experiment carried out with the corresponding software is shown in section 4. Finally, some conclusions and further discussion on some topics of the paper are given in the last section.

2 SN P Systems with Input and Decay

An SN P system consists of a set of neurons placed in the nodes of a directed graph and sending signals (called *spikes*) along the arcs of the graph (called *synapses*). The objects evolve according to a set of rules (called *spiking rules*). The idea is that a neuron containing a certain amount of spikes can consume some of them and produce other ones. The produced spikes are sent (maybe with a delay of some steps) to all neurons to which a synapse exists outgoing from the neuron where the rule was applied. A global clock is assumed and in each time unit each neuron which can use a rule should do it, but 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 (a detailed description of SN P systems can be found in [21] and the references therein).

In this section we introduce the *Hebbian SN P system unit* which is an SN P system with $m + 1$ neurons (m presynaptic neurons linked to one postsynaptic neuron) endowed with *input* and *decay*. At the starting point all the neurons are inactive. At rest, the membrane of biological neurons has a negative polarization of about $-65mV$, but we will consider the inactivity by considering the the number of spikes inside the neuron is zero. The dynamics of a Hebbian SN P system unit is quite natural. At the starting point, all neurons are at rest and in a certain moment the presynaptic neurons receive spikes enough to activate some rules. The instant of the arrival of the spikes can be different for each presynaptic neuron. These spikes activate one rule inside the neurons and the presynaptic neurons send spikes to the postsynaptic neuron. In the postsynaptic neuron a new rule can be triggered or not, depending on the arrival of spikes and it may send a spike to the environment.

2.1 The Input

The basic idea in SN P systems taken from biological spiking neuron models is the codification of the information in *time*. The information in a Hebbian SN P system unit is also encoded in the time in which the spikes arrive to the neuron and the time in which the new spikes are emitted. The input will be also encoded in time. The idea behind this codification is that the presynaptic neurons may not be activated at the same moment. If we consider a Hebbian SN P system unit as part of a wide neural network, it is quite natural to think that the spikes will not arrive to the presynaptic neurons (and consequently, their rules are not activated) at the same time. In this way, if we consider a Hebbian SN P system unit with m presynaptic neurons $\{u_1, \ldots, u_m\}$, an input will consist of a vector $\vec{x} = \{x_1, \ldots, x_m\}$ of non-negative integers where x_i represents the time unit of the global clock in which the neuron u_i is activated[2].

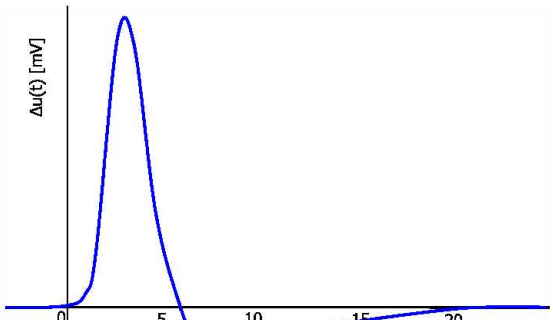
2.2 The Decay

The effect of a spike on the postsynaptic neuron can be recorded with an intracellular electrode which measures the potential difference between the interior of the cell and its surroundings. Without any spike input, the neuron is at rest corresponding to a constant membrane potential. After the arrival of the spike, the potential changes and finally decays back to the resting potential. The spikes, have an amplitude of about 100mV and typically a duration of 1-2 ms. This means that if the total change of the potential due to the arrival of spikes is not enough to activate the postsynaptic neuron, it decays after some milliseconds and the neuron comes back to its resting potential (see Fig. 1).

This biological fact is not implemented in current SN P systems, where the spikes can be inside the neuron for a long time if they are not consumed by any rule. In the Hebbian SN P system unit, we introduce the decay in the action potential of the neurons. When the impulse sent by a presynaptic neuron arrives to the postsynaptic neuron, if it is not consumed for triggering any rule in the postsynaptic neuron it decays and its contribution to the total change of potential in the postsynaptic neuron decreases with time. This decayed potential is still able to contribute to the activation of the postsynaptic rule if other spikes arrive to the neuron and the addition of all the spikes trigger any rule. If this one does not occur, the potential decays and after a short time the neuron reaches the potential at rest. Figure 2 shows a scheme in which two presynaptic neurons send two spikes each of them at different moments to a postsynaptic neuron. Figure 3 shows the changes of potential in the postsynaptic neuron till reaching the threshold for firing a response.

In order to formalize the idea of decay in the framework of SN P systems we introduce a new type of extended rules: the *rules with decay*. They are rules of the form

[2] In Section 5 we discuss about other codings for the input.



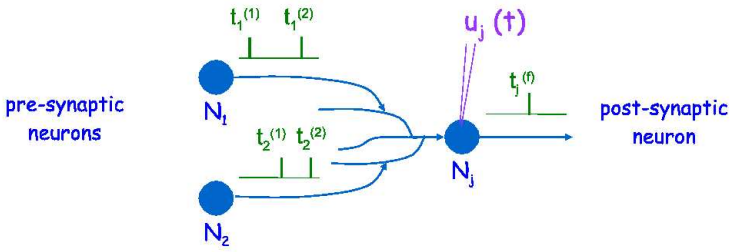


Fig. 2. Two presynaptic and one postsynaptic neuron

$$E/a^k \rightarrow (a^p, S); d$$

where, E is a regular expression over $\{a\}$, k and p are natural numbers with $k \geq p \geq 0$, $d \geq 0$ and $S = (s_1, s_2, \ldots, s_r)$ is a finite non-increasing sequence of natural numbers called the *decaying sequence* where $s_1 = k$ and $s_r = 0$. If $E = a^k$, we will write $a^k \rightarrow (a^p, S); d$ instead of $a^k/a^k \rightarrow (a^p, S); d$.

The intuition behind the *decaying sequence* is the following. When the rule $E/a^k \rightarrow (a^p, S); d$ is triggered at t_0 we look in $S = (s_1, \ldots, s_r)$ for the greatest l such that $p \geq s_l$. Such s_l spikes are sent to the postsynaptic neurons according with the delay d in the usual way. Notice that s_l can be equal to p, so at this point this new type of rule is a generalization of the usual extended rules.

At t_0+d+1, the s_l spikes arrive to the postsynaptic neurons. The decay of such spikes is determined by the decaying sequence. If the spikes are not consumed by the triggering of a rule in the postsynaptic neuron, they decay and at time t_0+d+2 we will consider that $s_l - s_{l+1}$ spikes have disappeared and we only have s_{l+1} spikes in the postsynaptic neuron. If the spikes are not consumed in the following steps

by the triggering of a postsynaptic rule, at $t_0 + d + 1 + r - l$ the number of spikes will be decreased to $s_r = 0$ and the spikes are lost.

This definition of decay[3] can be seen as a generalization of the decaying spikes presented in [7]. In that paper a decaying spike a is written in the form (a, e), where $e \geq 1$ is the period. From the moment a spike (a, e) arrives in a neuron, e is decremented by one in each step of computation. As soon as $e = 0$, the corresponding spike is lost and cannot be used anymore.

In this way, a rule $E/a^k \rightarrow a^p; d$ $(k > p)$ where a^p are p decaying spikes (a, e) can be seen with our notation as $E/a^k \rightarrow (a^p, S); d$ with $S = (s_1, \ldots, s_{e+2})$, $s_1 = k$, $s_2 = \cdots = s_{e+1} = p$ and $s_{e+2} = 0$.

2.3 Hebbian SN P System Units

Hebbian SN P system units are SN P systems with a fixed topology endowed with input and decay. They have the following common features:

- The initial number of the spikes inside the neurons is always zero in all Hebbian SN P system units, so we do not refer to them in the description of the unit.
- All the presynaptic neurons are linked to the postsynaptic neuron and these are all the synapses in the SN P system, so they are not provided in the description.
- The output neuron is the postsynaptic one.

Bearing in mind these features, we describe a Hebbian SN P system unit in the following way.

[3] Further discussion about the decay can be found in Section 5.

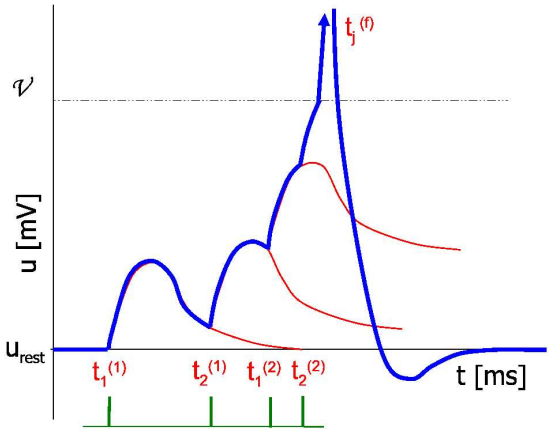


Fig. 3. The potential at the postsynaptic neuron

Definition 1. *A Hebbian SN P system unit of degree m is a construct*

$$H\Pi = (O, u_1, \ldots, u_m, v),$$

where:

- $O = \{a\}$ *is the alphabet (the object a is called spike);*
- $u_1, \ldots, u_m$ *are the presynaptic neurons. Each presynaptic neuron u_i has associated a set of rules $R_i = \{R_{i1}, \ldots, R_{il_i}\}$ where for each $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, l_i\}$, R_{ij} is a decaying rule of the form:*

$$a^k \rightarrow (a^{n_{ij}}, S); d_{ij}$$

  We will call n_{ij} the presynaptic potential *of the rule and d_{ij} is the* delay *of the rule. Note that all rules are triggered by k spikes. The decaying sequence S will be discussed below.*
- *v is the postsynaptic neuron which contains only one postsynaptic rule $E_p^*/a^p \rightarrow a; 0$ where E_p^* is the set[4] of regular expressions $\{n \in \mathbb{N} \mid n \geq p\}$. We will call p the* threshold *of the postsynaptic potential of the Hebbian SN P system unit.*

By considering the decaying sequences we can distinguish among three types of Hebbian SN P system units:

- Hebbian SN P system units with *uniform decay*. In this case the decaying sequence S is the same for all the rules in the m presynaptic neurons.
- Hebbian SN P system units with *locally uniform decay*. In this case the decaying sequence S is the same for all the rules in each presynaptic neuron.
- Hebbian SN P system units with *non-uniform decay*. In this case each rule has associated a decaying sequence.

A Hebbian SN P system unit is an abstract machine where a global clock is assumed (the system is synchronized). It takes an input and can provide an output or not, depending if the potential in the postsynaptic neuron reaches or not its threshold. The concept of input of a Hebbian SN P system unit is defined as follows:

Definition 2. *An* input *for a* Hebbian SN P system unit *of degree m is a vector $\vec{x} = (x_1, \ldots, x_m)$ of m non-negative integers x_i.*

A Hebbian SN P system unit with input *is a pair $(H\Pi, \vec{x})$ where $H\Pi$ is Hebbian SN P system unit and $\vec{x}$ is an input for it.*

The intuitive idea behind the input is encoding the information in time. Each x_i represent the moment, according to the global clock, in which one spike is provided to each presynaptic neuron.

[4] This rule is an adaptation of the concept of a rule from an extended spiking neural P system with thresholds taken from [7].

2.4 How it works

In this subsection we provide a description of the semantics of a Hebbian SN P system unit. As we saw before, each x_i in the input $\vec{x} = (x_1, \ldots, x_m)$ represents the time in which k spikes are provided to the neuron u_i. At the moment x_i in which the spike arrives to the neuron u_i one rule $(a^k \to (a^{n_{ij}}, S); d_{ij})$ is chosen in a non-deterministic way among all the rules of the neuron.

Applying it means that k spikes are consumed and we look in $S = (s_1, \ldots, s_r)$ for the greatest l such that $n_{ij} \geq s_l$. Such s_l spikes are sent to the postsynaptic neurons according to the delay d_{ij} in the usual way, i.e., s_l spike arrive to the postsynaptic neuron at the moment $x_i + d_{ij} + 1$. The decay of such spikes is determined by the decaying sequence. As we saw above, if the spikes are not consumed by the triggering of a rule in the postsynaptic neuron, they decay and at time $x_i + d_{ij} + 2$ we will consider that $s_l - s_{l+1}$ spikes have disappeared and we only have s_{l+1} spikes in the postsynaptic neuron. If the spikes are not consumed in the following steps by the triggering of a postsynaptic rule, at $x_0 + d_{ij} + 1 + r - l$ the number of spikes will be decreased to $s_r = 0$ and the spikes are lost.

The potential on the postsynaptic neuron depends on the contributions of the chosen rules in the presynaptic neurons. Such rules send spikes that arrive to the postsynaptic neuron at different moments which depend on the input (the moment in which the presynaptic neuron is activated) and the delay of the chosen rule. The contribution of each rule to the postsynaptic neuron also changes along the time due to the decay.

Formally, the potential of the postsynaptic neuron is a natural number calculated as a function R^* which depends on the time t, on the input $\vec{x}$ and on the rules chosen in each neuron $R^*(R_{1i_1}, \ldots, R_{mi_m}, \vec{x}, t) \in \mathbb{N}$. Such a natural number represents the number of the spikes at the moment t in the postsynaptic neurons and it is the result of adding the contributions of the rules $R_{1i_1}, \ldots, R_{mi_m}$.

The Hebbian SN P system unit produces an output if the rule of the postsynaptic neuron v, $E_p^*/a^p \to a$ is triggered, i.e., if at any moment t the amount of spikes in the postsynaptic neuron is greater than or equal to the threshold p, then the rule is activated and triggered. If there does not exist such t, then the Hebbian SN P system unit does not send any spike to the environment.

Bearing in mind the decay of the spikes in the postsynaptic neuron, if any spike has been sent out by the postsynaptic neuron after an appropriate number of steps, any spike will be sent to the environment. From a practical point of view we have a bound for the number of steps in which the spike can be expelled, so we have a decision method to determine if the input $\vec{x}$ provided to the Hebbian SN P system unit produces or not an output.

Example 1. Let us consider the following Hebbian SN P system unit

$$H\Pi = (O, u_1, u_2, v)$$

with non-uniform decay, where:

- $O = \{a\}$ is the alphabet;
- u_1, u_2 are the presynaptic neurons. The presynaptic neurons u_1, u_2 have associated the sets of rules $R_1 = \{R_{11}, R_{12}, R_{13}\}$ and $R_2 = \{R_{21}, R_{22}\}$, respectively, with

$$R_{11} \equiv a^3 \to (a^2, (3,2,0)); 0 \qquad R_{21} \equiv a^3 \to (a^2, (3,2,0)); 1$$
$$R_{12} \equiv a^3 \to (a, (3,1,0)); 1 \qquad R_{22} \equiv a^3 \to (a, (3,1,0)); 0$$
$$R_{13} \equiv a^3 \to (a^3, (3,0)); 0$$

- v is the postsynaptic neuron which contains only one postsynaptic rule $E_2^*/a^2 \to a; 0$.

Notice that in this example, the rules send all the presynaptic potential to the postsynaptic neuron but it only lasts one time unit before being lost. If they are not consumed immediately, they disappear.

Case 1: Let us consider the input $\vec{x} = (0,0)$, i.e., at $t = 0$ three spikes are placed in each presynaptic neuron. We represent the contribution of each rule for $\vec{x} = (0,0)$ in the following table. Notice that for $t \geq 3$ the contribution is zero for all the rules.

	R_{11}	R_{12}	R_{13}	R_{21}	R_{22}
$t=1$	2	0	3	0	1
$t=2$	0	1	0	2	0

Considering the different contributions of the rules and bearing in mind that in each neuron only one rule is non-deterministically chosen, the changes in the postsynaptic potential for $\vec{x} = (0,0)$ are described in the following table.

	$R_{11}\,R_{21}$	$R_{12}\,R_{21}$	$R_{13}\,R_{21}$	$R_{11}\,R_{22}$	$R_{12}\,R_{22}$	$R_{13}\,R_{22}$
$t=1$	**2**	0	**3**	**3**	1	**4**
$t=2$	2	**3**	2	0	1	0

Notice that with the input $\vec{x} = (0,0)$, the postsynaptic neuron activates the rule at $t = 1$ if the chosen rules are $R_{11}\,R_{21}$, $R_{13}\,R_{21}$, $R_{11}\,R_{22}$ or $R_{13}\,R_{22}$. If the chosen rules are $R_{12}\,R_{21}$, then the rule is activated at $t = 2$ and if the chosen rules are $R_{12}\,R_{22}$ then the postsynaptic rule is not activated.

Case 2: Let us consider now the input $\vec{x} = (1,0)$, i.e., at $t = 0$ three spikes are placed in the presynaptic neuron u_2 and in $t = 1$ other three spikes are placed in u_1. As above, we represent the contribution for $\vec{x} = (1,0)$ in the following table.

	R_{11}	R_{12}	R_{13}	R_{21}	R_{22}
$t=1$	0	0	0	0	1
$t=2$	2	0	3	2	0
$t=3$	0	1	0	0	0

The changes of the potential R^* in the postsynaptic potential for $\vec{x} = (1,0)$ are described in the following table.

	$R_{11}\,R_{21}$	$R_{12}\,R_{21}$	$R_{13}\,R_{21}$	$R_{11}\,R_{22}$	$R_{12}\,R_{22}$	$R_{13}\,R_{22}$
$t=1$	0	0	0	1	1	1
$t=2$	**4**	**2**	**5**	**2**	0	**3**
$t=3$	0	1	0	0	1	0

In this case, with the input $\vec{x} = (1, 0)$, the postsynaptic neuron triggers its rule at $t = 2$ but if the chosen rules are $R_{12}\,R_{22}$.

3 Learning

If we look at the Hebbian SN P system units as computational devices where the target is the transmission of information, we can consider that the device *successes* if a spike is sent to the environment and it *fails* if the spike is not sent. In this way, the lack of determinism in the choice of rules is a crucial point in the success of the devices because as we have seen above, if we provide several times the same input, the system can succeed or not.

In order to improve the design of these computational devices and in a narrow analogy with the Hebbian principle, we introduce the concept of *efficacy* in the Hebbian SN P system units. Such efficacy is quantified by endowing each rule with a *weight* that changes along the time, by depending on the contribution of the rule to the success of the device.

According to [8], in Hebbian learning, a synaptic weight is changed by a small amount if presynaptic spike arrival and postsynaptic firing *coincides*. This simultaneity constraint is implemented by considering a parameter s_{ij} which is the difference between the arrival of the contribution of the rule R_{ij} and the postsynaptic firing. Thus, the efficacy of the synapses such that its contributions arrive repeatedly shortly before a postsynaptic spike occurs is increased (see [13] and [3]). The weights of synapses such that their contributions arrive to the postsynaptic neuron *after* the postsynaptic spike is expelled are decreased (see [4] and [16]). This is basically the learning mechanism suggested in [17].

3.1 The Model

In order to implement a learning algorithm in our Hebbian SN P system units, we need to extend it with a set of weights that measure the efficacy of the synapses. The meaning of the weights is quite natural and it fits into the the theory of artificial neural networks [12]: The amount of spikes that arrives to the postsynaptic neuron due to the rule R_{ij} depends on the *contribution* of each rule and also on the *efficacy* of the synapse w_{ij}. As usual in artificial neural networks, the final contribution will be the contribution sent by the rule multiplied by the efficacy w_{ij}.

We fix these concepts in the following definition.

Definition 3. *An extended Hebbian SN P system unit of degree m is a construct*

$$EH\Pi = (H\Pi, w_{11}, \ldots, w_{ml_m}),$$

where:

- *$H\Pi$ is a Hebbian SN P system unit of degree m and the rules of the presynaptic neuron u_i are $R_i = \{R_{i1}, \ldots, R_{il_i}\}$ with $i \in \{1, \ldots, m\}$.*
- *For each rule R_{ij} with $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, l_i\}$, w_{ij} is a real number which denotes the* initial weight *of the rule R_{ij}.*

Associating a weight to each rule means to consider an individual synapse for each rule instead of a synapse associated to the whole neuron. The idea of considering several synapses between two neurons is not new in computational neuron models. For example, in [19] the authors present a model for spatial and temporal pattern analysis via spiking neurons where several synapses are considered. The same idea had previously appeared in [8]. Considering several rules in a neuron and one synapse associated to each rule allows us to design an algorithm for changing the weight (the efficacy) of the synapse according to the result of the different inputs.

The concept of input of a extended Hebbian SN P system unit is similar to the previous one. The information is encoded in time and the input of each neuron denotes the moment in which the neuron is excited.

Definition 4. *An* input *for an* extended Hebbian SN P system unit *of degree m is a vector $\vec{x} = (x_1, \ldots, x_m)$ of m non-negative integers x_i.*

An extended Hebbian SN P system unit with input *is a pair $(EH\Pi, \vec{x})$ where $H\Pi$ is an extended Hebbian SN P system unit and $\vec{x}$ is an input for it.*

The semantics

As we saw before, each x_i in the input $\vec{x} = (x_1, \ldots, x_m)$ represents the time in which the presynaptic neuron u_i is activated. The formalization of the activation of the neuron in this case differs from the Hebbian SN P system units. The idea behind the formalization is still the same: the postsynaptic neuron receives a little amount of electrical impulse according to the excitation time of the presynaptic neuron and the efficacy of the synapsis. The main difference is that we consider that there exist several synapses between one presynaptic neuron and the postsynaptic one (one synapse for each rule in the neuron) and the potential is transmitted along *all* these synapses according to their efficacy.

Extending the Hebbian SN P system units with efficacy in the synapses and considering that there are electrical flow along all of them can be seen as a generalisation of the Hebbian SN P system units. In Hebbian SN P system units only one rule R_{ij} is chosen in the presynaptic neuron u_i and the contribution emitted by R_{ij} arrives to the postsynaptic neuron according to the decaying sequence. Since the weight w_{ij} multiplies the contribution in order to compute the potential that

arrives to the postsynaptic neuron, we can consider the Hebbian SN P system unit as an extended Hebbian SN P system unit with the weight of the chosen rule R_{ij} equals to one and the weight of the remaining rules equals to zero.

At the moment x_i in the presynaptic neuron u_i we will consider that *all* rules $(a^k \to (a^{n_{ij}}, S); d_{ij})$ are activated. The potential on the postsynaptic neuron depends on the contributions of the rules in the presynaptic neurons and the efficacy of the respective synapses. Let us consider that at time x_i the rule $(a^k \to (a^{n_{ij}}, S); d_{ij})$ is activated and the efficacy of its synapse is represented by the weight w_{ij}. When the rule $(a^k \to (a^{n_{ij}}, S); d_{ij})$ is triggered at t_0 we look in $S = (s_1, \ldots, s_r)$ for the greatest l such that $p \times w_{ij} \geq s_l$. Then s_l spikes are sent to the postsynaptic neurons according with the delay d in the usual way.

At t_0+d+1, the s_l spikes arrive to the postsynaptic neurons. The decay of such spikes is determined by the decaying sequence. If the spikes are not consumed by the triggering of a rule in the postsynaptic neuron, they decay and at time t_0+d+2 we will consider that $s_l - s_{l+1}$ spikes have disappeared and we only have s_{l+1} spikes in the postsynaptic neuron. If the spikes are not consumed in the following steps by the triggering of a postsynaptic rule, at step $t_0 + d + 1 + r - l$ the number of spikes will be decreased to $s_r = 0$ and the spikes are lost. The extended Hebbian SN P system unit produces an output if the rule of the postsynaptic neuron v, $E_p^*/a^p \to a$ is triggered.

Bearing in mind the decay of the spikes in the postsynaptic neuron, if the output has not been produced after an appropriate number of steps, no spike will be sent to the environment. From a practical point of view we have a bound for the number of steps in which the spike can be expelled, so we have a decision method to determine if the input $\vec{x}$ provided to the extended Hebbian SN P system unit produces or not an output.

Example 2. Let us consider the extended Hebbian SN P system unit of degree m with uniform decay:

$$H\Pi = (O, u_1, u_2, v, w_{11}, w_{12}, w_{13}, w_{21}, w_{22}),$$

where:

- $O = \{a\}$ is the alphabet;
- u_1, u_2 are the presynaptic neurons. The presynaptic neurons u_1, u_2 have associated sets of rules $R_1 = \{R_{11}, R_{12}, R_{13}\}$ and $R_2 = \{R_{21}, R_{22}\}$, respectively, with

$$\begin{array}{ll} R_{11} \equiv a^{100} \to (a^{40}, S); 0 & R_{21} \equiv a^{100} \to (a^{80}, S); 1 \\ R_{12} \equiv a^{100} \to (a^{70}, S); 1 & R_{22} \equiv a^{100} \to (a^{40}, S); 0 \\ R_{13} \equiv a^{100} \to (a^{30}, S); 0 & \end{array}$$

  The decaying sequence is the same for all the rules, $S = (100, 80, 70, 30, 15, 0)$
- v is the postsynaptic neuron, and it contains only one postsynaptic rule $E_{70}^*/a^{70} \to a; 0$.

- The initial weights are $w_{11} = 0.9$, $w_{12} = 1.2$, $w_{13} = 0.5$, $w_{21} = 0$ and $w_{22} = 1$

In order to compute the function of the postsynaptic potential we need an input. Let us consider $\vec{x} = (1, 0)$. Let us focus on the first rule $R_{11} \equiv a^{100} \rightarrow (a^{40}, S); 0$. At $t = 1$ the rule is activated. According to its efficacy, 30 spikes will be placed in the postsynaptic neuron at $t = 2$, since $70 > 40 \times 0.9 = 36 \geq 30$. At $t = 3$ the contribution of this rule is 15 due to the decay and for $t \geq 4$ the contribution is zero. The second rule $R_{12} \equiv a^{100} \rightarrow (a^{70}, S); 1$ is also activated at $t = 1$. Due to the delay $d_{12} = 1$, the spikes sent by this rule will be placed at the postsynaptic neuron at $t = 3$. The number of emitted spikes will be 80 since $100 > 70 \times 1.2 = 84 \geq 80$. These spikes will decay in the following steps. We summarize the contributions in the following table. The last column represents the final contribution in the postsynaptic neuron by adding the partial contribution of all the rules.

	R_{11}	R_{12}	R_{13}	R_{21}	R_{22}	$\sum$
$t = 1$	0	0	0	0	30	30
$t = 2$	30	0	15	0	15	60
$t = 3$	15	80	0	0	0	**95**
$t = 4$	0	70	0	0	0	70
$t = 5$	0	30	0	0	0	30
$t = 6$	0	15	0	0	0	15

At time $t = 3$ the postsynaptic potential reaches the value 95 and it is the fist time that it is grater than the threshold, so the postsynaptic rule $E^*_{70}/a^{70} \rightarrow a; 0$ is activated and in the next step one spike is sent to the environment.

3.2 The Learning Problem

Let us come back to the Hebbian SN P system units. In such units, provided an input $\vec{x}$, success can be reached or not (i.e., the postsynaptic rule is triggered or not) depending on the non-deterministically rules chosen. In this way, the choice of some rules is *better* than the choice of other ones, by considering that a rule is *better* than another if the choice of the former leads us to the success with a higher probability than the choice of the latter. Our target is to learn which are the best rules according to this criterion.

Formally, a *learning problem* is a 4-uple $(EH\Pi, X, L, \epsilon)$, where:

- $EH\Pi$ is an extended Hebbian SN P system unit
- $X = \{\vec{x_1}, \ldots \vec{x_n}\}$ is a finite set of *inputs* of $EH\Pi$.
- $L : \mathbb{Z} \rightarrow \mathbb{Z}$ is a function from the set of integer numbers onto the set of integer numbers. It is called the *learning function*.
- ϵ is a positive constant called the *rate of learning*.

The *output* of a learning problem is an extended Hebbian SN P system unit.

Informal description of the algorithm

Let us consider an extended Hebbian SN P system $EH\Pi$, a learning function $L : \mathbb{Z} \to \mathbb{Z}$ and a rate of learning ϵ.. Let us consider an input $\vec{x}$ and we will denote by $t_{\vec{x}}$ the moment when the postsynaptic neuron reaches the potential for the trigger of the postsynaptic neuron. If such potential is not reached (and the postsynaptic neuron is not triggered) then $t_{\vec{x}} = \infty$.

On the other hand, for each rule $R_{ij} \equiv a^k \to (a^{n_{ij}}, S); d_{ij}$ of a presynaptic neuron we can compute the moment $t_{ij}^{\vec{x}}$ in which its contribution to the postsynaptic potential arrives to the postsynaptic neuron. It depends on the input $\vec{x}$ and the delay d_{ij} of the rule

$$t_{ij}^{\vec{x}} = \vec{x}_i + d_{ij} + 1$$

where $\vec{x}_i$ is the i-th component of $\vec{x}$.

We are interested in the influence of the rule R_{ij} on the triggering of the postsynaptic neuron. For that we need to know the difference between the arrival of the contribution $t_{ij}^{\vec{x}}$ and the moment $t_{\vec{x}}$ in which the postsynaptic neuron is activated.

For each rule R_{ij} and each input $\vec{x}$, such a difference is

$$s_{ij}^{\vec{x}} = t_{\vec{x}} - t_{ij}^{\vec{x}}$$

- If $s_{ij}^{\vec{x}} = 0$, then the postsynaptic neuron reaches the activation exactly in the instant in which the contribution of the rule R_{ij} arrives to the postsynaptic neuron. This fact leads us to consider that the contribution of R_{ij} to the postsynaptic potential has had a big influence on the activation of the postsynaptic neuron.
- If $s_{ij}^{\vec{x}} > 0$ and it is *small*, then the postsynaptic neuron reaches the activation a bit later than the arrival of the contribution of the rule R_{ij} to the postsynaptic neuron. This fact leads us to consider that the contribution of R_{ij} to the postsynaptic potential has influenced on the activation of the postsynaptic neuron due to the decay, but it is not so important as in the previous case.
- If $s_{ij}^{\vec{x}} < 0$ or $s_{ij}^{\vec{x}} > 0$ and it is not *small*, then the contribution of R_{ij} has no influence on the activation of the postsynaptic neuron.

The different interpretations of *small* or *big influence* are determined by the different *learning functions* $L : \mathbb{Z} \to \mathbb{Z}$. For each rule R_{ij} and each input $\vec{x}$, $L(s_{ij}^{\vec{x}}) \in \mathbb{Z}$ measures de degree of influence of the contribution of R_{ij} to the activation of the postsynaptic neuron produced by the input $\vec{x}$.

According to the principle of Hebbian learning, the efficacy of the synapses such that their contributions influence on the activation of the postsynaptic neuron must be increased. The weights of synapses such that their contributions have no influence on the activation of the postsynaptic neuron are decreased.

Formally, given an extended Hebbian SN P system $H\Pi$, a learning function $L : \mathbb{Z} \to \mathbb{Z}$, a rate of learning ϵ and an input $\vec{x}$ of $H\Pi$, the *learning algorithm*

outputs a new extended Hebbian SN P system $H\Pi'$ which is equal to $H\Pi$, but the weights: each w_{ij} has been replaced by a new w'_{ij}

$$w'_{ij} = w_{ij} + \epsilon L(s_{ij}^{\vec{x}})$$

Depending on the sign of $L(s_{ij}^{\vec{x}})$, the rule R_{ij} will increase or decrease its efficacy. Note that $L(s_{ij}^{\vec{x}})$ is multiplied by the *rate of learning* ϵ. This rate of learning is usual in learning process in artificial neural networks. It is usually a small number which guarantees that the changes on the efficacy are not abrupt.

Finally, given a *learning problem* $(H\Pi, X, L, \epsilon)$, the learning algorithm takes $\vec{x} \in X$ and outputs $H\Pi'$. In the second step, the learning problem $(H\Pi', X - \{\vec{x}\}, L)$ is considered and we get a new $H\Pi'$. The process finishes when all the inputs has been consumed and the algorithm outputs the last extended SN P system unit.

Example 3. Let us consider the extended Hebbian SN P system unit of degree m with uniform decay:

$$H\Pi = (O, u_1, u_2, v, w_{11}, w_{12}, w_{13}, w_{21}, w_{22}),$$

where:

- $O = \{a\}$ is the alphabet;
- u_1, u_2 are the presynaptic neurons. The presynaptic neurons u_1, u_2 have associated the sets of rules $R_1 = \{R_{11}, R_{12}, R_{13}\}$ and $R_2 = \{R_{21}, R_{22}\}$, respectively, with

$$\begin{array}{ll} R_{11} \equiv a^{100} \to (a^{40}, S); 0 & R_{21} \equiv a^{100} \to (a^{80}, S); 1 \\ R_{12} \equiv a^{100} \to (a^{70}, S); 1 & R_{22} \equiv a^{100} \to (a^{40}, S); 0 \\ R_{13} \equiv a^{100} \to (a^{30}, S); 0 & \end{array}$$

  The decaying sequence is the same for all the rules, $S = (100, 80, 70, 30, 15, 0)$
- v is the postsynaptic neuron which contains only one postsynaptic rule $E^*_{70}/a^{70} \to a; 0$.
- The initial weights are $w_{11} = 1.0$, $w_{12} = 1.0$, $w_{13} = 1.0$, $w_{21} = 1.0$ and $w_{22} = 1.0$

Let us consider the learning problem $(EH\Pi, X, L, \epsilon)$, where

- $EH\Pi$ is the extended Hebbian SN P system unit described above.
- $X = \{\vec{x_1}, \vec{x_2}\}$ with $\vec{x_1} = (0, 2)$ and $\vec{x_2} = (0, 0)$.
- L is the learning function $L : \mathbb{Z} \to \mathbb{Z}$

$$L(s) = \begin{cases} 4 & \text{if } s = 0 \\ 2 & \text{if } s = 1 \\ 1 & \text{if } s = 2 \\ -1 & \text{otherwise} \end{cases}$$

- The rate of learning $\epsilon = 0.1$

Step 1: Let us consider the input $\vec{x} = (0, 2)$. The contribution can be summarised in the following table:

	R_{11}	R_{12}	R_{13}	R_{21}	R_{22}	$\sum$
$t = 1$	30	0	30	0	0	60
$t = 2$	15	70	15	0	0	100
$t = 3$	0	30	0	0	30	60
$t = 4$	0	15	0	80	15	110
$t = 5$	0	0	0	70	0	70
$t = 6$	0	0	0	30	0	30
$t = 7$	0	0	0	15	0	15

Therefore, at time $t = 2$ the potential of the postsynaptic neuron reaches a value greater than the threshold 70, then $t_{(0,2)} = 2$. We can compute now the values $t_{ij}^{(0,2)} = x_i + d_{ij} + 1$, $s_{ij}^{(0,2)} = t_{(0,2)} - t_{ij}^{(0,2)}$ and $L(s_{ij}^{(0,2)})$ for every rule R_{ij}.

After computing the values $L(s_{ij}^{(0,2)})$ for every rule R_{ij}, the new weights are calculated as

$$w'_{ij} = w_{ij} + \epsilon\, L(s_{ij}^{(0,2)})$$

These values are summarised in the following table

	$t_{ij}^{(0,2)}$	$s_{ij}^{(0,2)}$	$L(s_{ij}^{(0,2)})$	w_{ij}	w'_{ij}
R_{11}	1	1	2	1	1.2
R_{12}	2	0	4	1	1.4
R_{13}	1	1	2	1	1.2
R_{21}	4	-2	-1	1	0.9
R_{22}	3	-1	-1	1	0.9

Therefore, after this fist step the new weights are $w'_{11} = 1.2$, $w'_{12} = 1.4$, $w'_{13} = 1.2$, $w'_{21} = 0.9$ and $w'_{22} = 0.9$.

Step 2: Let us consider the new extended Hebbian SN P system unit built by replacing the initial weights by the new w'_{ij} and let us consider the second input $\vec{x_2} = (0, 0)$. The contribution can be summarized in the following table.

	R_{11}	R_{12}	R_{13}	R_{21}	R_{22}	$\sum$
$t = 1$	30	0	30	0	30	90
$t = 2$	15	80	15	70	15	195
$t = 3$	0	70	0	30	0	100
$t = 4$	0	30	0	15	0	45
$t = 5$	0	15	0	0	0	15

Therefore, at time $t = 1$ the potential of the postsynaptic neuron reaches a value greater than the threshold 70, then $t_{(0,0)} = 1$. We can compute now the values $t_{ij}^{(0,0)} = x_i + d_{ij} + 1$, $s_{ij}^{(0,0)} = t_{(0,0)} - t_{ij}^{(0,0)}$ and $L(s_{ij}^{(0,0)})$ for every rule R_{ij}.

After computing the values $L(s_{ij}^{(0,0)})$ for every rule R_{ij}, the new weights are calculated as

$$w''_{ij} = w'_{ij} + \epsilon\, L(s_{ij}^{(0,0)})$$

These values are summarized in the following table

	$t_{ij}^{(0,2)}$	$s_{ij}^{(0,2)}$	$L(s_{ij}^{(0,2)})$	w'_{ij}	w''_{ij}
R_{11}	1	0	4	1.2	1.6
R_{12}	2	-1	-1	1.4	1.3
R_{13}	1	0	4	1.2	1.6
R_{21}	2	-1	-1	0.9	0.8
R_{22}	1	0	4	0.9	1.3

Therefore, after this fist step the new weights are $w'_{11} = 1.6$, $w'_{12} = 1.3$, $w'_{13} = 1.6$, $w'_{21} = 0.8$ and $w'_{22} = 1.3$.

The use of weights needs more discussion. The weights are defined as real numbers and membrane computing devices are discrete. If we want to deal with discrete computation in all the steps of the learning process we have to choose the parameters carefully. The following result gives a sufficient constraint for having an integer number of spikes at any moment.

Theorem 1. *Let a be the greatest non-negative integer such that for all presynaptic potential n_{ij} there exists an integer z_{ij} such that $n_{ij} = x_{ij} \times 10^a$.*

Let b be the smallest non-negative integer such that for all initial weight w_{ij} and for the rate of learning ϵ there exist the integers k_{ij} and k such that $w_{ij} = k_{ij} \times 10^b$ and $\epsilon = k \times 10^b$.

If $a - b \geq 0$, then for all presynaptic potential n_{ij} and all the weights w obtained along the learning process, $n_{ij} \times w$ is an integer number.

In other words, if there exists a and b such that all the presynaptic potentials n_{ij} can be expressed as $n_{ij} = x_{ij} \times 10^a$ for an appropriate integer x_{ij} and the initial weights w_{ij} and rate of learning ϵ can be expressed as $w_{ij} = k_{ij} \times 10^b$ and $\epsilon = k \times 10^b$ for appropriate integer numbers k_{ij}, k and $a - b \geq 0$ then for all presynaptic potential n_{ij} and all the weights w obtained along the learning process, $n_{ij} \times w$ is an integer number.

Proof. It suffices to consider the recursive generation of new weights $w_{n+1} = w_n + \epsilon L(s_n)$ and therefore

$$w_{n+1} = w_0 + \epsilon(L(s_0) + \cdots + L(s_n)).$$

If we develop $n_{ij} \times w_{n+1}$ according to the statement of the theorem, we have

$$\begin{aligned} n_{ij} \times w_{n+1} &= x_{ij} \times 10^a \times [k_0 \times 10^{-b} + (k \times 10^{-b}(L(s_0) + \cdots + L(s_n)))] \\ &= 10^{a-b} \times x_{ij} \times [k_0 + k(L(s_0) + \cdots + L(s_n))] \end{aligned}$$

Since $x_{ij} \times [k_0 + k(L(s_0) + \cdots + L(s_n))]$ is an integer number, if $a - b \geq 0$ then $n_{ij} \times w_{n+1}$ is an integer number.

4 An Experiment

Let us consider the Hebbian SN P system

$$H\Pi = (O, u_1, u_2, v)$$

with uniform decay, where:

- $O = \{a\}$ is the alphabet;
- u_1, u_2 are the presynaptic neurons. The presynaptic neurons u_1, u_2 have associated the sets of rules $R_1 = \{R_{11}, R_{12}, R_{13}\}$ and $R_2 = \{R_{21}, R_{22}\}$, respectively, with

$$\begin{array}{ll} R_{11} \equiv a^{3000} \rightarrow (a^{3000}, S); 0 & R_{21} \equiv a^{3000} \rightarrow a^{1000}; 0 \\ R_{12} \equiv a^{3000} \rightarrow (a^{2000}, S); 1 & R_{22} \equiv a^{3000} \rightarrow a^{3000}; 3 \\ R_{13} \equiv a^{3000} \rightarrow (a^{2000}, S); 7 & \end{array}$$

- The *decaying sequence* is $S = (3000, 2800, 1000, 500, 0)$.
- v is the postsynaptic neuron which contains only one postsynaptic rule $E^*_{1200}/a^{1200} \rightarrow a; 0$.

Let $EH\Pi$ be the Hebbian SN P system unit $H\Pi$ extended with the initial weights $w_{11} = 0.5$, $w_{12} = 0.5$, $w_{13} = 0.5$, $w_{21} = 0.5$ and $w_{22} = 0.5$.

Let us consider the learning problem $(EH\Pi, X, L, \epsilon)$ where

- $EH\Pi$ is the extended Hebbian SN P system unit described above,
- X is a set of 200 random inputs (x_i^1, x_i^2) with $1 \leq 1 \leq 200$ and $x_i^1, x_i^2 \in \{0, 1, \ldots, 5\}$
- L is the learning function $L : \mathbb{Z} \rightarrow \mathbb{Z}$

$$L(s) = \begin{cases} 3 & \text{if } s = 0 \\ 1 & \text{if } s = 1 \\ -1 & \text{otherwise} \end{cases}$$

- The rate of learning is $\epsilon = 0.001$

We have programmed an appropriate software for dealing with this learning problems. After applying the learning algorithm, we obtain a new extended Hebbian SN P system unit similar to $EH\Pi$ but with the weights

$$w_{11} = 0.754, \quad w_{12} = 0.992, \quad w_{13} = 0.3, \quad w_{21} = 0.454, \quad w_{22} = 0.460$$

Fig 4 shows the evolution of the weights of the synapses.

The learning process shows clearly the differences among the rules.

- The *worst* rule is R_{13}. In a debugging process of the design of an SN P System network that rule should be removed. The value of the weight has decreased along all the learning process. This fact means that the rule has never contributed to the success of the unit and then it can be removed. The reason is

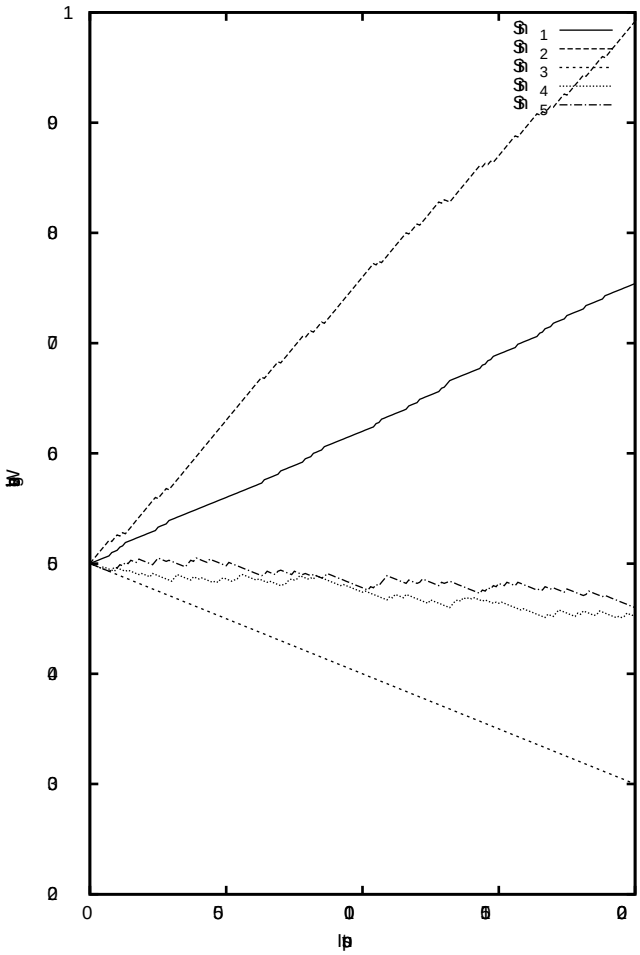


Fig. 4. The evolution of the weights

clear. The rule emits four spikes and the postsynaptic rule is activated with two spikes. Even with the decay, the potential provided by the rule is too much for triggering the rule.

- On the other extreme, the *best* rules are R_{11} and R_{21}. In most of the cases, (not all) these rules have been involved in the success of the unit.
- The other two rules R_{21} and R_{22} have eventually contributed to the success of the unit but not so clearly as R_{11} and R_{21}. We can also guess the reasons. For R_{11}, the presynaptic potential, 1000, has little influence in the postsynaptic potential and for R_{22}, the presynaptic potential is larger than the threshold, but it has a large delay, so the arrival of its potential to the postsynaptic neuron is often later than the activation of the postsynaptic rule.

5 Conclusions and Future Work

The integration in an unique model of concepts from neuroscience, artificial neural networks and spiking neural P systems is not an easy task. Each of the three fields have its own concepts, languages and features. The work of integration consists on choosing ingredients from each field and trying to compose a computational device with the different parts. This means that some of the ingredients used in the devices presented in this paper are not usual in the SN P systems framework. Although the authors have tried to be as close to the SN P system spirit as possible some remarks should be considered.

In the paper, the input of the device is provided as a vector $(t_1, \ldots, t_m)$ of non-negative integers, where t_i represents the moment in which one rule (non-deterministically chosen) of the neuron u_i is activated. Obviously, this is not the usual way to provide the input to an SN P system. Nonetheless, the information encoded in the vector $(t_1, \ldots, t_m)$ can be provided to the input neurons by m spike trains were all the elements are 0's and there is only one 1 in the position t_i. In this way, the input is encoded by m spike trains, which is closer to the standard inputs in SN P systems.

The idea of providing the input with a spike train of 0's and only one 1 in the position t_i carries out new problems. In the literature of SN P systems, in the instant t_i only one spike is supplied to the neuron u_i. In our device we want that a rule of type $a^r \rightarrow a^p; d$ is activated with $r > 1$. At this point we can consider several choices. The first one is to consider that at time t_i the spike train provides r spikes, but this choice lead us far from the SN P system theory. A second option is to consider that the spike trains have r consecutive 1's and each of them provide one spike. The remaining elements in the train are zeros. In this way the moment t_i will be the instant in which the r spikes have been provided to the neuron. A drawback for this proposal can be that r can be a big number and this increase the number of steps of the device. A third choice is to consider amplifier modules as in Figure 5. The leftmost neuron receives a spike train where all the elements are 0's but the $t_i - th$ which is 1. At the moment t_i only one spike is supplied to the neuron. At $t_i + 1$, one spike arrives to the r postsynaptic neurons, and each of them sends one spike to the rightmost neuron, so at $t_i + 2$ exactly r spikes arrive simultaneously to the last neuron.

These three solutions can be an alternative to the use of the vector $(t_1, \ldots, t_m)$ and deserve to be considered for further research in this topic.

Another main concept in this paper is the delay. It has strong biological intuition, but it is difficult to insert into the SN P systems theory. The main reason is that if we consider the spike as the information unit it does not make sense to talk about a half of a spike or a third of a spike. In that sense, the approach to decay from [7] is full of sense since one spike exists or it is lost, but its potential it is not decreasing in time.

The key point for the decay in this paper is taken from the definition of *extended* SN P systems. In such devices, a neuron can send a different amount of spikes depending on the chosen rule. So, in such devices the information is not only

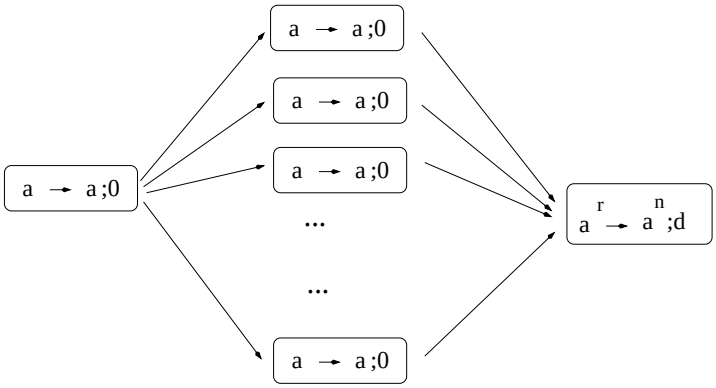


Fig. 5. Amplifier module

encoded in the *time* between two consecutive spikes, but on the *number* of spikes. This lead us to define the decay as a decrement in the number of spikes. In this way, we can consider that a pulse between two neurons is composed by a certain number of spikes which can be partially lost depending on the time.

In this paper, such a decay has been implemented by extending the rules with a finite decreasing sequence which can be uniform, locally-uniform or non uniform for the set of rules. Other implementations are also possible. Probably, the decay can also be implemented with an extra neuron as in Figure 6 which sends to the final neuron a decaying sequence of spikes.

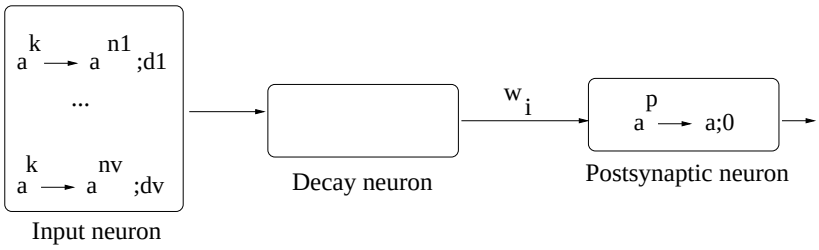


Fig. 6. Including a decay neuron

The use of weights also deserves to be discussed. In Theorem 1 we provide sufficient conditions for handling at every moment an integer number of spikes. In this way, the presented devices keep the principle of discrete computation of SN P systems. Nonetheless, further questions should be considered. For example, the use of negative weights or weights greater than one. Should we consider negative weights and/or a *negative* contribution to the postsynaptic potential? On the other hand, the use weights greater than one leads us to consider that the contribution of

one rule to the postsynaptic potential is *greater than* its own presynaptic potential. Can the efficiency of the synapses amplify the potential beyond the number of emitted spikes?

More technical questions are related to the rate of learning and to the algorithm of learning. Both concepts have been directly borrowed from artificial neural networks and need deeper study in order to adapt them to the specific features of SN P systems.

As a final remark, we consider that this paper opens a promising line research bridging SN P systems and artificial neural networks without forgetting the biological inspiration and also opens a door to applications of SN P systems.

Acknowledgements

The authors acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the project of excellence TIC-581 of the Junta de Andalucía.

References

1. E.D. Adrian. The impulses produced by sensory nerve endings. J. Physiol. (Lond.), 61, 49-72, 1926.
2. E.D. Adrian. The basis of Sensation. W.W. Norton. New York, 1926.
3. T.V.P. Bliss and G.L. Collingridge. Nature, 361, 31–99, 1993.
4. D. Debanne, B.H. Gähwiler and S.M. Thompson. Proc. Natl. Acsd. Sci. USA, 91, 1148-1152, 1994.
5. R. Eckhorn, R. Bauer, W. Jordan, M. Brosch, W. Kruse, M. Munk and H.J. Reitboeck. Coherent oscillations: A mechanism of feature linking in the visual cortex? Biol. Cybern. 60, 121-130, 1988.
6. A.K. Engel, P. König, A.K. Kreiter, and W. Singer. Interhemispheric sycrhonization of oscillatory neural responses in cat visual cortex. Science, 252, 1177-1179, 1991.
7. R. Freund, M. Ionescu and M. Oswald. Extended spiking neural P systems with decaying spikes and/or total spiking. Proceedings of the International Workshop Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, 64-75, 2007.
8. W. Gerstner, R. Kempter, L. van Hemmen and H. Wagner. A neuronal learning rule for sub-millisecond temporal coding. Nature, 383, 76–78, 1996.
9. W. Gerstner and W.Kistler. Spiking Neuron Models. Single Neurons, Populations, Plasticity. Cambridge University Press, 2002.
10. C.M. Gray, P. König, A.K. Engel and W. Singer. Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties. Nature, 338, 334-337, 1989.
11. C.M. Gray and W. Singer. Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. Proc. Natl. Acad. Sci. USA, 86, 1698-1702, 1989.
12. S. Haykin. Neural Networks. A Comprehensive Foundation. Macmillan College Publishin Company, Inc. 1994.
13. D.O. Hebb. The Organization of Behavior, Wiley, New York, 1949.

14. D.H. Hubel and T.N. Wiesel. Receptive Fields of Single Neurons in the Cat's Striate Cortex. J. Phisiol. (Lond.), 148, 574-591, 1959.
15. M. Ionescu, Gh. Păun and T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3, 279-308, 2006.
16. H. Markram and B. Sakmann. Soc. Neurosci. Abstr., 21, 2007, 1995.
17. H. Markram and M. Tsodyks. Redistribution of synaptic efficacy between neocortical pyramidal neurons. Nature, 382, 807–810, 1996.
18. V.B. Mountcastle. Modality anf topographic properties of single neurons of cat's somatosensory cortex. J. Neurophysiol., 20, 408-434, 1957.
19. T. Natschläger and B. Ruf. Spatial and temporal pattern analysis via spiking neurons. Network: Comp. Neural Syst., 9(3), 319-338, 1998.
20. Gh. Păun: *Membrane Computing. An Introduction.* Springer–Verlag, Berlin, 2002.
21. Gh. Păun: Twenty six research topics about spiking neural P systems. In M.A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez and A. Riscos-Núñez, editors. Fifth Brainstorming Week on Membrane Computing, Fénix Editora, Sevilla, 263–280, 2007.
22. S. Ramón y Cajal. Histologie du Systeme Nerveux de l'Homme et des Vertébres. A. Maloine, Paris, 1909.
23. W. Singer. The role of synchrony in neocortical processing and synaptic plasticity. In E. Domany, J.L. van Hemmel and K. Schulten, editors, Models of Neural Networks II, chapter 4, Springer Verlag, Berlin, 1994.
24. S. Thorpe, S, Fize and C. Marlot. Speed of processing in the human visual system. Nature, 381, 520-522, 1996.
25. P systems web page `http://ppage.psystems.eu/`

Ordinary Membrane Machines versus Other Mathematical Models of Systems Realizing Massively Parallel Computations

Adam Obtułowicz

Institute of Mathematics of the Polish Academy of Sciences
E-mail: adamo@impan.gov.pl

Summary. A comparison of ordinary membrane machines, understood as certain recursive families of deterministic P systems, with some other mathematical models of systems realizing massively parallel computations is discussed. These mathematical models are those which respect recursiveness of computational tasks of systems, i.e., the functions to be computed are recursive functions and the decision problems correspond to recursive sets. The comparison together with open problems is summarized in the enclosed tables, where open problems are indicated by question mark "?".

1 Introduction

We present and discuss a comparison of ordinary membrane machines, understood as certain recursive families of deterministic P systems (for P systems see [23]), with some other mathematical models of systems realizing massively parallel computations. These mathematical models are those which respect recursiveness of computational tasks of systems, i.e., the functions to be computed are recursive functions and the decision problems correspond to recursive sets.

The comparison is discussed with regard to those (comparative) features of the mathematical models which one can treat as advantageous features from the logical or complexity theoretical point of view, or by means that they provide natural extensions of the (classes of) models for application of other approaches to computing than the discrete time approach or deterministic approach. The mathematical models are chosen in such a way that for every comparative feature there is provided at least one representative or typical example of a model of this feature.

2 Compared Models and Their Features

We discuss the following mathematical models of systems realizing massively parallel computations.

1: *ordinary membrane machines* defined to be recursive families $\Pi = (\Pi^i : i \in \mathrm{INP})$ of deterministic P systems Π^i for recursive sets INP of input data, where P systems Π^i are constructs understood as in [23] and their determinism is understood as in e.g. [16]. For more explanations see Remark 1 below;
2: Parallel Random Access Machines (PRAMs), cf. [9], [14], [20];
3: neural net models due to H. T. Siegelmann and E. D. Sontag, cf. [27];
4: R. Gandy's machines, cf. [10] and [26], the parallelism of their computations was pointed out in [7];
5: parallel Abstract State Machines and intra-step interacting Abstract State Machines due to Y. Gurevich, cf. [2];
6: Connection Machines due to W. D. Hillis, cf. [12].

The above models are such that they respect recursiveness of computational tasks understood as in Introduction.

The ordinary membrane machines require more explanations which are given in the following remark.

Remark 1 A representative example of an ordinary membrane machine is discussed in [18], where input data in INP are propositional formulas Φ in conjunctive normal form and the deterministic P systems Π^Φ—the elements of the family are used to solve SAT problem in a polynomial time, like in [16]. More precisely, the P system Π^Φ associated to a formula Φ generates that unique evolution process of membrane systems which provides a decision in a polynomial time (with respect to the number of clauses and the number of variables occurring in Φ) whether Φ holds for some valuation of variables occurring in Φ. The above recursive family of P systems was introduced in [18] to describe in a program-like uniform way the P systems solving SAT problem in [16].

We use "membrane machines" to name the families in 1 because evolving membrane systems are basic mechanisms of computations realized by P systems, see [23]. The adjective "ordinary" is applied to distinguish the families in 1 from other possible families of P systems, e.g. families of stochastic P systems or quantum P systems.

Remark 2 An evolving membrane system or simply a *membrane system* $\mathcal{S}$ is understood in the paper to be given by its *underlying tree* $\mathcal{T}_\mathcal{S}$, i.e., finite non-empty graph which is a tree, whose vertices, called *membranes*, are labeled by multisets over the *sets* $\mathbb{O}_\mathcal{S}$ *of objects* of $\mathcal{S}$. More precisely, there is given *labeling function* $\mathcal{M}_\mathcal{S} : V(\mathcal{T}_\mathcal{S}) \to N^{\mathbb{O}_\mathcal{S}}$ of $\mathcal{S}$ defined on the set $V(\mathcal{T}_\mathcal{S})$ of vertices of $\mathcal{T}_\mathcal{S}$ such that the values $\mathcal{M}_\mathcal{S}(v)$ are functions $f : \mathbb{O}_\mathcal{S} \to N$ valued in the set N of natural numbers with 0. For an equivalence of the above treatment membrane systems with the treatment of membrane systems understood as in [23] see Remark 2 in [19].

The comparison of the above models is discussed with regard to the following features of them.

A: basic definitions of systems and computations realized by them are free from concepts involving recursiveness, e.g., recursive families of programs, etc., and

the number of defining axioms and principles is finite and minimal in the sense that leaving one of them does not suffice to prove recursiveness of computational tasks understood as in Introduction;

B: definition of computational complexity measure of consumed space during computation is explicit, natural, and simple;

C: the modeled computations comprise a wide scope of possibilities of parallelism from computations realized by distributed systems, with every processor equipped with an (access) independent memory unit from other processors, to systems with processors sharing an access to common memory unit like in the case of PRAMs in 2;

D: solutions of NP complete problems in a polynomial time with an exponential space expense are provided;

E: immediate extensions to randomized or quantum counterparts are provided, like in [15], [17];

F: immediate extensions to continuous time computations are provided like in [27];

G: explicit treatment of communication (interaction) with environment during computation, understood as in Y. Gurevich's papers ([2], [8]), is provided;

H: the models have an immediate realization by really existing devices (computers) in silicon or biochemical one.

We complete the above listed features A–H by the following comments and remarks containing explanatory, representative, or typical examples.

Ad A. The class of Gandy's machines in 4 is a representative example of a class having the feature A. These machines are defined in an abstract mathematical way in [10] by four principles and [10] contains the result that whatever is computable by the devices satisfying these principles is also computable by Turing machines. The principles are minimal in the sense that no three of them suffice to prove the mentioned result.

Ad B. The class of ordinary membrane machines in 1 is an example of a class having the feature B. Let for an ordinary membrane machine $\Pi = (\Pi^i \,|\, i \in \mathrm{INP})$ a unique evolution process generated by Π^i be presented by the following sequence of length n_i

$$\mathcal{S}_0^i \Rightarrow \mathcal{S}_1^i \Rightarrow \ldots \Rightarrow \mathcal{S}_{n_i}^i,$$

where $\mathcal{S}_0^i, \mathcal{S}_1^i, \ldots, \mathcal{S}_{n_i}^i$ are membrane systems such that $\mathcal{S}_0^i$ is the initial membrane system of the process, $\mathcal{S}_j^i$ evolves into $\mathcal{S}_{j+1}^i$ for all j with $0 \leq j < n_i$, and $\mathcal{S}_{n_i}^i$ is the final membrane system of the process. Then one defines the claimed in B space complexity measure $\mathrm{SPACE}(i)$ by

$$\mathrm{SPACE}(i) = \max_{0 \leq j \leq n_i} \sum_{v \in V(\mathcal{T}_{\mathcal{S}_j^i})} \left(1 + \sum_{a \in \mathbb{O}_{\mathcal{S}_j^i}} \mathcal{M}_{\mathcal{S}_j^i}(v)(a)\right),$$

where $V(\mathcal{T}_{\mathcal{S}_j^i})$, $\mathbb{O}_{\mathcal{S}_j^i}$, $\mathcal{M}_{\mathcal{S}_j^i}$ are the set of vertices of the underlying tree $\mathcal{T}_{\mathcal{S}_j^i}$, the set of objects, and labeling function of $\mathcal{S}_j^i$, respectively, see Remark 2.

Ad C. Some ordinary membrane machines together with evolution processes generated by them can be treated as distributed systems such that every their processor has an independent memory unit, see representative example in Remark 1. In this example a membrane v of an evolving membrane system $\mathcal{S}$ can be treated as a processor equipped with a memory unit containing $\mathcal{M}_{\mathcal{S}}(v)$, see [16] and [18] for more details.

Ad D. The ordinary membrane machine shown in Remark 1 provides a solution of NP complete problem SAT in a polynomial time with an exponential space expense. The construction of this machine was inspired by the pioneering paper [22]. For other related constructions see e.g. [24].

Ad E. Randomized P systems in [17] and quantum P systems in [15] give rise to the extensions of ordinary membrane machines as claimed in E. We point out that one may claim the extensions to randomized counterparts which could provide probabilistic approach to P/NP conjecture proposed in [1].

Ad F. The class of models in 3 is an example of a class of models having feature F because inductive next state formula for neural nets can be simply converted to a system of ordinary differential equations. We point out here that one may claim the extensions to continuous, analog models of computations which could include continuous approach to P/NP conjecture in [5], [6].

Ad G. The attempts to describe the interconnections of membrane systems with environment through their skins are shown among others in [21].

Ad H. The connection machines in 6 have a physical realization. Namely, the Connection Machine Models CM-1, CM-2, and CM-5 were made by Thinking Machines Corporation in Boston, between 1986 and 1996.

3 Concluding Discussion and Open Problems

The comparison of the ordinary membrane machines and the mathematical models with regard to the features described in Section 2 is summarized in the following tables, where new open problems are indicated by question mark "?".

Table of answers to the question:
does X hold for i?

	A	B	C	D	E	F	G	H	?
1	?	Yes	Yes	Yes	Yes	?	Yes?	?	
2	?	Yes	No?	Yes?	Yes	?	No	?	
3	?	?	Yes?	Yes?	Yes	Yes	?	?	
4	Yes	?	Yes?	?	?	?	?	?	
5	?	?	Yes?	Yes?	?	?	Yes	?	
6	?	?	?	?	?	?	?	Yes	
?									

For $X \in \{\mathrm{A}, \mathrm{B}, \ldots, \mathrm{H}\}$, $1 \leq i \leq 6$, and open problems indicated by '?', including conjectures indicated by 'Yes?', 'No?', meant 'rather Yes' and 'rather No', respectively, where the properties $\mathrm{A}, \mathrm{B}, \ldots, \mathrm{H}$ correspond to the comparative features and the numbers $1, 2, \ldots, 6$ correspond to the models as in the lists given in Section 2, respectively.

Matrix-like table of answers to the question:
does i simulate j in polynomial slow-down?
(*i*—row, *j*—column)

	1	2	3	4	5	6
1	Yes	Yes	?	?	?	Yes
2	No?	Yes	No?	No?	No?	Yes
3	?	Yes	Yes	?	?	Yes
4	?	Yes	Yes	Yes	Yes?	Yes
5	?	Yes	?	?	Yes	Yes
6	No	No	No	No	No	Yes

Open problems and conjectures are indicated by '?', 'Yes?', 'No?' as in the first table.

From the first table and its first row we conclude that despite the treatment of natural computing as less important than e.g. multicore computing, cf. [25], or not worth to mention, cf. [3], the bio-inspired membrane computing, a vital part of natural computing, contains ordinary membrane machines which are computation models of advantageous features from complexity theoretical point of view (see features B, C, D) and open for extensions to new approaches to computing outlined among others in [4].

References

1. Arora, S., and Safra, Sh., *Probabilistic checking of proofs: a new characterization of NP*, in: Proc. 33rd IEEE Symp. on Foundation of Computer Science, 1992, 2–12.
2. Blass, A., and Gurevich, Yu., *Algorithms: a quest for absolute definitions*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 81 (2003), 195–225.
3. Buss, S. R., Kechris, A. S., Pillay, A., Shore, S. A., *The prospects for mathematical logic in the twenty-first century*, the independent presentations included in panel discussion at The Annual Meeting of the Association for Symbolic Logic held in Urbana–Champaign, June 2000.
4. Calude, C. S., and Păun, Gh., *Bio-steps beyond Turing*, BioSystems 77 (2004), 175–194.
5. Costa, J. F., and Mycka, J., *An analytic condition for $P \subset NP$*, March 2006.
6. Costa, J. F., and Mycka, J., *The conjecture $P \neq NP$ presented by means of some class of real functions*, 2007.
7. Dahlhaus, E., and Makowsky J. A., *Gandy's principles for mechanisms as a model for parallel computation*, in: The Universal Turing Machine: a Half-Century Survey, ed. R. Herken, second ed., Springer, Wien-New York 1995, pp. 283–288.

8. Dershowitz, N., and Gurevich, Yu., *A natural axiomatization of Church's thesis*, July 2007.
9. Fich, F. E., *The Complexity of Computation on the Parallel Random Access Machine*, Chapter 21.
10. Gandy, R., *Church's thesis and principles for mechanisms*, in: The Kleene Symposium, eds. J. Barwise et al., North-Holland, Amsterdam 1980, pp. 123–148.
11. Grover, L., and Rudolph, T., *How significant are the known collision and element distinctness quantum algorithms?*, `arXiv: quant-ph/0309123v1, 16 Sep 2003`.
12. Hillis, W. D., *The Connection Machines*, Cambridge, Mass. 1985.
13. Hirvensalo, M., *Quantum Computing*, second edition, Springer, Berlin 2004.
14. Karp, R. M., and Ramachandran, V., *Parallel algorithms for shared-memory machines*, in: Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity, MIT Press, Cambridge 1990, 869–941.
15. Leporati, A., Pescini, D., and Zandron, C., *Quantum energy-based P systems*, in: Proc. Brainstorming Workshop on Uncertainty in Membrane Computing, Palma de Mallorca, November 2004.
16. Obtułowicz, A., *Deterministic P systems for solving SAT problem*, Romanian Journal of Information Science and Technology 4 (2001), 195–201.
17. Obtułowicz, A., *Probabilistic P systems*, in: Membrane Computing, Lecture Notes in Comput. Sci. 2597, Springer, Berlin 2003, 377–387.
18. Obtułowicz, A., *Note on some recursive family of P systems with active membranes*, P systems Web page, 2003.
19. Obtułowicz, A., *Gandy's principles for mechanisms and membrane computing*, in: Proc. Cellular Computing (Complexity Aspects), ESF PESC Exploratory Workshop, January 31–February 2, 2005, ed. M. A. Gutiérrez-Naranjo et al., Sevilla 2005, pp. 267–276.
20. Papadimitriou, Ch. H., *Computational Complexity*, Addison–Wesley, Reading 1994.
21. Păun, A., and Păun, Gh., *The power of communication*: *P systems with symport/antiport*, New Generation Computing 20 (2002), 295–306.
22. Păun, Gh., *P systems with active membranes: Attacking NP complete problems*, Journal of Automata, Languages and Combinatorics 6 (2000), 75–90.
23. Păun, Gh., *Membrane Computing. An Introduction*, Springer-Verlag, Berlin 2002.
24. Perez Jimenez, M. J., Romero Jimenez, A., and Caparrini, F. S., *Decision P systems and $P \neq NP$ conjecture*, in: Membrane Computing, Lecture Notes in Comput. Sci. 2597, Springer, Berlin 2003, 388–399.
25. Scott, D. S., *Looking to the Future*, talk given at CiE (Computability in Europe) Conference, Siena, July 2007, `http://www.mat.unisi.it/~sorbi/sito/CiETalks/ScottCiE07.pdf`.
26. Sieg, W., *Computability Theory*, Seminar Lectures, University of Bologna, November 2004, `http://www.phil.cmu.edu/summerschool/2006/Sieg/computability_theory.pdf`
27. Siegelmann, H. T., and Sontag, E. D., *On the computational power of neural nets*, J. Comput. System Sci. 50 (1995), 132–150.

Graphics and P Systems: Experiments with JPLANT

Elena Rivero-Gil, Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mails: elen.rg@gmail.com, magutier@us.es, marper@us.es

Summary. The hand-made graphical representation of the configuration of a P system becomes a hard task when the number of membranes and objects increases. In this paper we present a new software tool, called JPLANT, for computing and representing the evolution of a P system model with membrane creation. We also present some experiments performed with JPLANT and point out new lines for the research in computer graphics with membrane systems.

1 Introduction

Since A.R. Smith [12] proposed the Lindenmayer systems (L-systems) [5] as a tool for synthesizing realistic images of plants, many efforts have been done for bridging the theory of formal languages and computer graphics.

In [2, 3], a first membrane-based device for computer graphics was presented. It was a hybrid model between L-systems and membrane computing and it used concepts very close to the L-systems model. Later, in [10], a new approach was presented for representing the development of higher plants with P systems. It was based on a type of P systems with membrane creation and it was entirely developed with membrane computing techniques. The basic idea was to consider the growing of the structure the membranes in a P system with membrane creation.

By definition, the structure of membranes in a cell-like P system is a tree. In P systems with membrane creation, new membranes can be created inside the existing membranes and this produces the expansion of the structure of membranes by increasing the depth of the branches. With an appropriate interpretation of the objects inside the membranes, the membrane structure can be represented as a tree which evolves in time and the length and width of the branches can grow in a similar way to real plants. In [11], the study started at [10] was completed by adding stochastic rules to the P system. In this case, the non-deterministic choice

of different rules produces different configurations of the P systems and hence, different graphical representations.

The hand-made graphical representation of the configuration of a P system becomes a hard task when the number of membranes and objects increases. For going on with the study of the relation between P systems and computer graphics it was necessary to develop a software able to deal with complex P systems and represent graphically its evolution in time.

In this paper we present such a software, JPLANT, which computes the first configurations of a computation and draws the corresponding graphical representation. This software is a very useful tool for the experimental research of the graphical representation of P systems. We show several experiments and open new research lines for exploring the possibilities of P systems.

The paper is organized as follows: Section 2 recall the restricted model of P systems with membrane creation used for the graphical design. Section 3 gives a brief presentation of the software JPLANT and the next section shows several experiments. The paper finishes with some conclusions and lines for future research.

2 P Systems with Membrane Creation

Membrane computing is a branch of natural computing which abstracts from the structure and the functioning of the living cell. In the basic model, membrane systems (also frequently called P systems) are distributed parallel computing devices, processing multisets of symbol-objects, synchronously, in the compartments defined by a cell-like membrane structure[1].

In this paper we will consider P systems which make use of membrane creation rules, which was first introduced in [4, 6]. However, our needs are far simpler than what the models found in the literature provide. This is the reason why we introduce the new variant of *restricted P systems with membrane creation.*

A restricted P system with membrane creation is a tuple $\Pi = (O, \mu, w_1, \ldots, w_m, R)$ where:

1. O is the alphabet of *objects*. There exist two distinguished objects, F and W that always belong to the alphabet of any P system considered below.
2. μ is the initial *membrane structure*, consisting of a hierarchical structure of m membranes (all of them with the same label; for the sake of simplicity we omit the label).
3. $w_1, \ldots, w_m$ are the multisets of objects initially placed in the m regions delimited by the membranes of μ.
4. R is a finite set of *evolution rules* associated with every membrane, which can be of the two following kinds:
   a) $a \to v$, where $a \in O$ and v is a multiset over O. This rule replaces an object a present in a membrane of μ by the multiset of objects v.

[1] A detailed description of P systems can be found in [9] and updated information in [13].

b) $a \to [v]$, where $a \in O$ and v is a multiset over O. This rule replaces an object a present in a membrane of μ by a new membrane with the same label and containing the multiset of objects v.

A membrane structure together with the objects contained in the regions defined by its membranes constitute a configuration of the system. A transition step is performed applying to a configuration the evolution rules of the system in the usual way within the framework of membrane computing, that is, in a non-deterministic maximally parallel way; a rule in a region is applied if and only if the object occurring in its left–hand side is available in that region; this object is then consumed and the objects indicated in the right–hand side of the rule are created inside the membrane. The rules are applied in all the membranes simultaneously, and all the objects in them that can trigger a rule must do it. When there are several possibilities to choose the evolution rules to apply, non-determinism takes place.

2.1 Graphical Representation

In this section we show how to use, through a suitable graphical representation, restricted P systems with membrane creation to model branching structures. The key point of the representation relies on the fact that a membrane structure is a *rooted tree of membranes*, whose root is the skin membrane and whose leaves are the elementary membranes. Thus, this seems a suitable frame to encode the branching structure.

Let us suppose that the alphabet O of objects contains the objects F and W, and let us fix the lengths l and w.

A simple model to graphically represent a membrane structure is to make a depth-first search of it, drawing, for each membrane containing the object F, a segment of length $m \times l$, where m is the multiplicity of F. If the number of copies of F in a membrane increases along the computation, the graphical interpretation is that the corresponding segment is lengthening. Analogously, the multiplicity of the symbol W specify the width of the segments to be drawn as follows: if the number of objects W present in a membrane is n, then the segment corresponding to this membrane must be drawn with width $n \times w$.

Each segment is drawn rotated with respect to the segment corresponding to its parent membrane. In order to determine the rotation angle we need to fix a third parameter δ. This angle δ together with the length l and the width w will determine the picture of the P system.

In order to compute the rotation angle of a segment with respect to its parent membrane we consider two new objects that can appear in the alphabet: $+$ and $-$. The rotation angle will be $n \times \delta$, where n is the multiplicity of objects "+" minus the multiplicity of objects "−" in the membrane. That is, each object "+" means that the rotation angle is increased by δ whereas each object "−" means that it is decreased by δ.

Inside the membranes other objects can appear that do not have geometrical interpretation. They are related to the development of the graph in time.

For a better understanding let us consider the following example: let Π_1 be the restricted P system with membrane creation such that

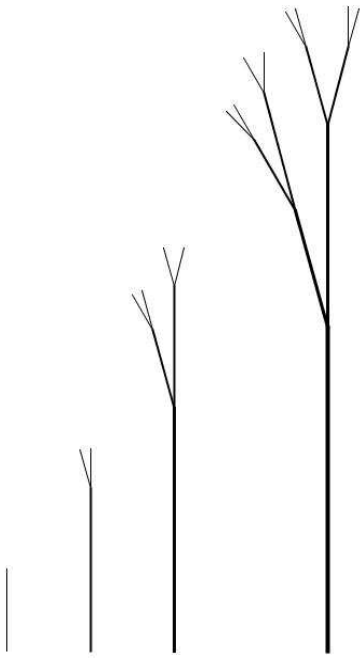


Fig. 1. First four configurations

- The alphabet of objects is $O = \{F, W, B_l, B_s, B_r, L, L_1, E, +, -\}$.
- The initial membrane structure together with the initial multiset of objects is $[F^2\, W\, B_l\, B_s\, L_1\, E]$.
- The rules are:

$$\begin{array}{ll} B_l \rightarrow [+\, F\, W\, B_l\, B_s\, L\, E] & L \rightarrow L\, F \\ B_s \rightarrow [F\, W\, B_l\, B_r\, L_1\, E] & L_1 \rightarrow L_1\, F^2 \\ B_r \rightarrow [-\, F\, W\, B_l\, B_s\, L\, E] & E \rightarrow E\, W \end{array}$$

In this system, the object B_s represents the straight branches to be created, whereas the objects B_l and B_r represent branches to be created rotated to the left and to the right, respectively. The objects F and W will determine the length and the width of the corresponding branch. The objects L, L_1 and E do not have a graphical interpretation; they can be considered as seeds for growing the branch in length and width.

The initial configuration consists of one membrane which contains two copies of F and one copy of W. If we consider the parameters l, w and δ, then the graphical representation of this initial configuration is a single segment of length $2 \times l$ and width w. In the first step, the objects B_l and B_s create new membranes, so the picture of this configuration consists on three segments. The new membrane created by B_s does not contains objects $+$ or $-$ and then the corresponding segment is not rotated with respect to the segment that represents the skin. On the other hand, the membrane created by B_l contains one object $+$, so its segment will be rotated an angle δ with respect to its parent membrane.

Notice also that the evolution of the objects L_1 and E has modified the number of objects F and W in the skin, so in this new picture, the segment corresponding to the skin has length $4 \times l$ and width $2 \times w$.

Figure 1 shows the graphical representation of the first four configurations where we fix a bottom-up orientation and an angle δ of 15 degrees.

2.2 Stochastic Versus Non-deterministic P Systems

The non-determinism is one of the main features of P systems and the possibility of reaching different configurations leads us to consider different graphical representations in the evolution of a P system.

One possible way to formalize the probability of obtaining one or other configuration is via stochastic P systems. Several alternatives to incorporate randomness into membrane systems can be found in the literature (see [1, 7, 8] and the references therein). One of them is to associate each rule of the P system with a probability. Thus, to pass from a configuration of the system to the next one we apply to every object present in the configuration a rule chosen at random, according to those probabilities, among all the rules whose left–hand side coincides with the object[2].

For example, let us consider Π_2 the following restricted P system with membrane creation:

- The alphabet of objects is $O = \{F, W, B_l, B_s, B_r, L, L_1, E\}$.
- The initial membrane structure together with the initial multiset of objects is $[F^2\, W\, B_l\, B_s\, L_1\, E]$.
- The rules are:

$$
\begin{array}{ll}
B_l \xrightarrow{1/2} [+\, F\, W\, B_l\, B_s\, L\, E] & L \to L\, F \\
B_l \xrightarrow{1/2} [-\, F\, W\, B_l\, B_s\, L\, E] & L_1 \to L_1\, F^2 \\
B_r \xrightarrow{1/2} [+\, F\, W\, B_l\, B_s\, L\, E] & E \to E\, W \\
B_r \xrightarrow{1/2} [-\, F\, W\, B_l\, B_s\, L\, E] & B_s \to [F\, W\, B_l\, B_r\, L_1\, E]
\end{array}
$$

[2] This idea was presented in [11].

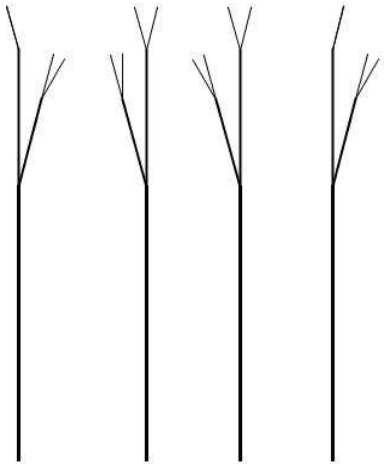


Fig. 2. Four configurations after the second step

There exist two rules for the evolution of the object B_l and two possibilities for the evolution of the object B_r. The probability for each choice is 1/2. Notice that we do not make explicit the probability of the rule when this is one.

Figure 2 shows four different configurations after the second step of this P system with the angle $\delta = 15$.

3 JPLANT

In order to avoid the heavy hand-made computation for the graphical representation a new software tool has been designed. In this paper we present JPLANT [3], which computes the first configurations of a computation of a restricted P system with membrane creation and draws the corresponding graphical representation of the configurations of such computation.

JPLANT has been written in Java and it has a nice intuitive user-friendly graphical interface. The initial configuration and the set of rules are provided in plain text mode. The right syntax of the initial configuration and the rules are checked before starting the computation. The generation of a new configuration is driven by the user which can choose between jumping to a configuration N or generating (and drawing) at each time the next configuration.

The software tool is thought as a drawing tool so the computed new configurations are not showed to the user in the text mode. The output is a picture with a set of connected segments drawn according with the rules described in Section

[3] The software is available from [13].

2. For each new configuration, a new picture is drawn, so the output of this tool is a sequence of pictures which can be saved in several computer graphic formats.

The graphical representation of one configuration is not unique. It depends on the parameters l, w and δ which determine the length and width of the segments as well as the rotation angle with respect to the segment corresponding to the parent membrane. Such parameters are the input of the tool and they must be also provided by the user with the initial configuration and the rules.

The current version of JPLANT includes the ability of load and save files with the input data and save the generated pictures and also provide color to the pictures.

The color is one of the basic tools in the graphical design. In the current version, the color of the segment associated with each membrane is not associated with any object inside the membrane. In this way, we cannot change the color of a membrane by the analysis of the membrane structure of a computation. Nonetheless, JPLANT provides the ability of giving color to the generated picture. It is an ability which is not associated with the P system which generates the picture, but it is a powerful tool in order to get realistic representations.

4 Applications

Next we illustrate the possibilities of JPLANT with some examples.

4.1 Polygons and Spirals

Polygons and spirals can be considered a very special case of branching structures. They consists of a connected set of segments where a vertex only connect two segments. From a membrane computing point of view, this means that each membrane in a configuration only contains one membrane.

Polygons

A first example of figures built with P systems are regular polygons. In such polygons the length of the side is constant and the angle of deviation from the previous side is also constant. A simple calculus shows us that a deviation of $\delta = 360/n$ degrees allows us to built a regular polygon of n sides.

Figure 3 shows regular polygons of $n = 10$ and $n = 12$ sides obtained with $\delta = 36$ and $\delta = 30$ degrees. Obviously the number of steps are 10 and 12 respectively. The P system is the following

Initial configuration: $[F\,W\,H]$

Rule: $H \rightarrow [-F\,W\,H]$

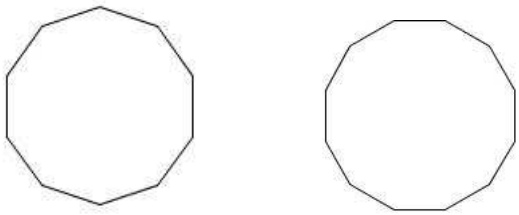


Fig. 3. 10-polygon and 12-polygon

Spirals

In mathematics, a spiral is a curve which emanates from a central point, getting progressively farther away as it revolves around the point. The concise mathematical definition is *the locus of a point moving at constant speed whose distance from a fixed point increases at a specific rate.*

An Archimedean spiral (a spiral named after the 3rd-century-BC Greek mathematician Archimedes) is the locus of points corresponding to the locations over time of a point moving away from a fixed point with a constant speed along a line which rotates with constant angular velocity. Equivalently, in polar coordinates (ρ, ω) it can be described by the equation $\rho = a + b\omega$ with real numbers a and b. Archimedes described such a spiral in his book *On Spirals*. It can be represented with the following P system:

$$\begin{array}{l} \texttt{Initial configuration:} \quad [F^n W H L] \\ \qquad\qquad\qquad\quad \texttt{Rules:} \quad H \rightarrow [-F^n W L H] \\ \qquad\qquad\qquad\qquad\quad\;\; L \rightarrow LF \end{array}$$

Figure 4 shows the representation of such Archimedes spiral for $n = 5$, length of $F = 0.01$, width $W = 1.0$, angle $\delta = 15$ and step 120.

The logarithmic spiral is a special kind of spiral curve which often appears in nature. It was first described by Descartes and extensively investigated by Jakob Bernoulli, who called it *Spira mirabilis*, "the marvelous spiral". Its equation in polar coordinates is $\rho = c^{\omega}$. It can be approximated by the P system

$$\begin{array}{l} \texttt{Initial configuration:} \quad [F^n W H L] \\ \qquad\qquad\qquad\quad \texttt{Rules:} \quad H \rightarrow [-F^n W L H] \\ \qquad\qquad\qquad\qquad\quad\;\; L \rightarrow LM_1F \\ \qquad\qquad\qquad\qquad\quad\;\; M_1 \rightarrow M2 \\ \qquad\qquad\qquad\qquad\quad\;\; \ldots \\ \qquad\qquad\qquad\qquad\quad\;\; M_{i-1} \rightarrow M_i \\ \qquad\qquad\qquad\qquad\quad\;\; M_i \rightarrow L \end{array}$$

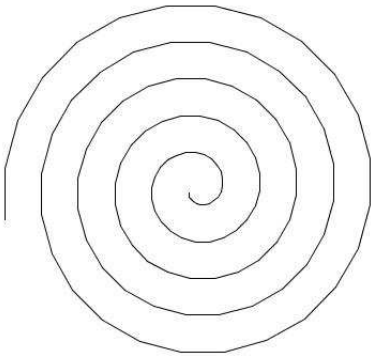


Fig. 4. Archimedes' spiral

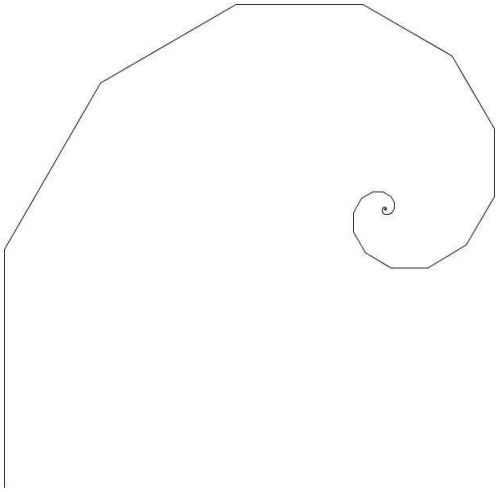


Fig. 5. Logarithmic spiral

Figure 5 shows the representation of such logarithmic spiral for $n = 10$, $i = 7$, length of $F = 0.001$, width $W = 1.0$ angle $\delta = 30$ and step 40.

4.2 Friezes

Another application of JPLANT for the graphical representation of restricted P systems with membrane creation is the design of friezes.

With the appropriate interpretation of the symbols, the following P system can be represented as a frieze based on right angles which has a flavor of Greek friezes. It can be extended horizontally in a non bounded way.

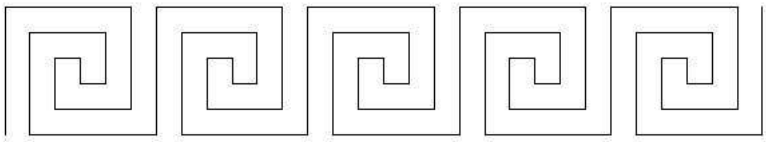


Fig. 6. The first frieze

`Initial configuration:`	$[F^5\, W\, H_1]$	
`Rules:`	$H_1 \rightarrow [-\, F^5\, W\, H_2]$	$H_7 \rightarrow [+\, F\, W\, H_8]$
	$H_2 \rightarrow [-\, F^4\, W\, H_3]$	$H_8 \rightarrow [+\, F^2\, W\, H_9]$
	$H_3 \rightarrow [-\, F^3\, W\, H_4]$	$H_9 \rightarrow [+\, F^3\, W\, H_{10}]$
	$H_4 \rightarrow [-\, F^2\, W\, H_5]$	$H_{10} \rightarrow [+\, F^4\, W\, H_{11}]$
	$H_5 \rightarrow [-\, F\, W\, H_6]$	$H_{11} \rightarrow [+\, F^5\, W\, H_{12}]$
	$H_6 \rightarrow [-\, F\, W\, H_7]$	$H_{12} \rightarrow [+\, F^5\, W\, H_1]$

Figure 6 shows the representation of such frieze for length of $F = 0.5$, width $W = 1$ angle $\delta = 90$ and step 60.

Figure 7 shows a horizontally bounded frieze based on the Archimedes spiral.

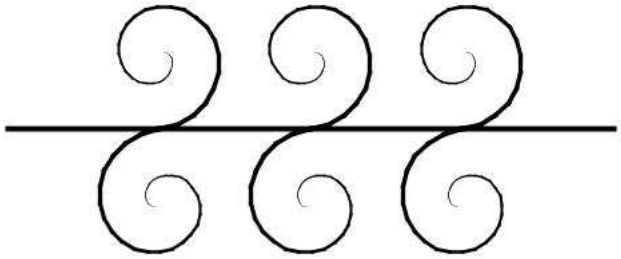


Fig. 7. The second frieze

`Initial configuration:`	$[F^{300}\, W^{40}\, H_1\, I_1\, D_1]$	
`Rules:`	$H_1 \rightarrow [F^{300}\, W^{40}\, H_2\, I_2\, D_2]$	$I_1 \rightarrow I_2$
	$H_2 \rightarrow [F^{300}\, W^{40}\, H_3\, I_3\, D_3]$	$D_1 \rightarrow D_2$
	$H_3 \rightarrow [F^{300}\, W^{40}]$	$I_2 \rightarrow I_3$
	$L \rightarrow L\, F$	$D_2 \rightarrow D_3$
	$K \rightarrow K\, W$	$I_3 \rightarrow I_4$
	$I_4 \rightarrow [-^{11}\, F\, W\, L\, K\, D_4]$	$D_3 \rightarrow D_4$
	$D_4 \rightarrow [+\, F\, W\, L\, K\, D_4]$	

Figure 7 shows the representation of such frieze for length of $F = 0.01$, width $W = 0.1$ angle $\delta = 15$ and step 40.

4.3 Plants

Figure 8 shows the corresponding graphical representation of the ninth configuration of the P system presented in Section 2.1, where we fix a bottom-up orientation with a length $F = 1$, width $W = 2$ and an angle δ of 15 degrees.

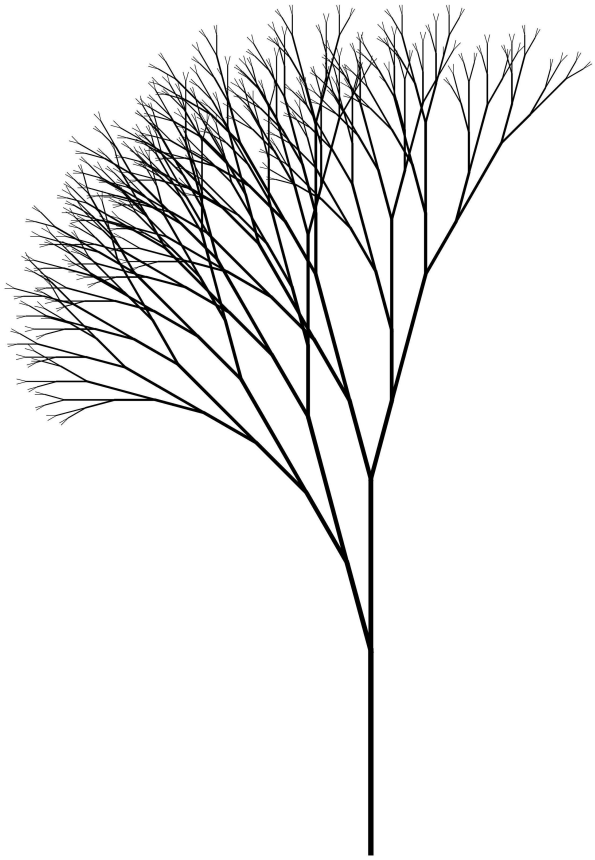


Fig. 8. Tree

Figure 9 represents four different trees obtained with JPLANT from the P system in Section 2.2.

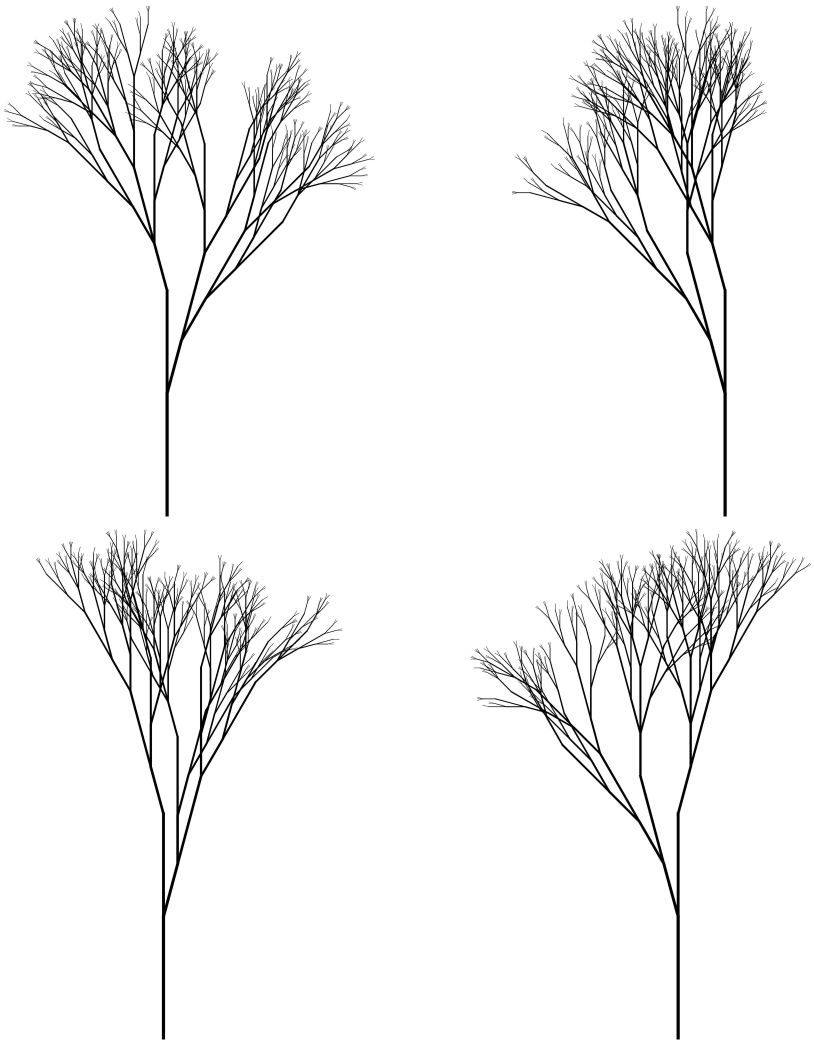


Fig. 9. Four configurations

5 Conclusions and Future Work

In this paper we have shown the suitability of P systems for modeling the growth of branching structures. It is our opinion that using membrane computing for this task could be an alternative to L-systems, the model most widely studied nowadays, for several reasons: the process of growing is closer to reality, since for example a plant does not grow by "rewriting" its branches, but by lengthening, widening and ramifying them; the membrane structure of P systems supports better and clearer the differentiation of the system into small units, easier to

understand and possibly with different behaviors; the computational power of membrane systems can provide tools to easily simulate more complex models of growing, for example taking into account the flow of nutrients or hormones.

Nevertheless, it is still necessary a deeper study of several features of our proposed framework as compared with that of Lindenmayer systems. Two aspects that have to be investigated are the complexity of the models that can be constructed, and the computational efficiency in order to generate their graphical representation. On one hand, the use of the ingredients of membrane computing can lead to more intuitive models; on the other hand, we lose the linear sequence of graphical commands that characterize the parsing algorithm of L-systems.

From a theoretical point of view, one of the main drawbacks of the model is that it is extremely simple. Although the orientation of the paper belongs to the framework of membrane computing, the exclusive use of rules of type $a \to v$ and $a \to [v]$ miss the potential richness of expressiveness and computation of P systems. The following steps on this line should be devoted to the study of the graphical possibilities of P systems with more features, such as labels for the membranes (they can help to distinguish between different parts of a plant), the use of communication rules, allowing objects to cross the membranes of the system, division and/or dissolution rules, rules with cooperation, etc.

Acknowledgement

The authors acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the project of excellence TIC-581 of the Junta de Andalucía.

References

1. Ardelean, I.I., Cavaliere, M.: Modelling Biological Processes by Using a Probabilistic P System Software. Natural Computing, 2(2), (2003), 173–197.
2. Georgiou, A., Gheorghe, M.: Generative Devices Used in Graphics. In Alhazov, A., Martín–Vide, C., Păun, Gh. (eds.): Preproceedings of the Workshop on Membrane Computing. Technical Report, Vol. 28/03. Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona, (2003), 266–272.
3. Georgiou, A., Gheorghe, M., Bernardini, F.: Membrane–Based Devices Used in Computer Graphics. In Ciobanu, G. Păun, Gh., Pérez–Jiménez, M.J. (eds.): Applications of Membrane Computing. Springer–Verlag, Berlin Heidelberg New York, (2006), 253–282.
4. Ito, M., Martín–Vide, C., Păun, Gh.: A Characterization of Parikh Sets of ET0L Languages in Terms of P Systems. In Ito, M., Păun, Gh., Yu, S. (eds.): Words, Semigroups, and Transducers. World Scientific, (2001), 239–254.
5. Lindenmayer, A.: Mathematica Models for Cellular Interaction in Development, Parts I and II. Journal of Theoretical Biology, 18, (1968), 280–315.

6. Madhu, M., Krithivasan, K.: P Systems with Membrane Creation: Universality and Efficiency. In Margenstern, M., Rogozhin, Y. (eds.): Proceedings of the Third International Conference on Universal Machines and Computations. Lecture Notes in Computer Science, 2055, (2001), 276–287.
7. Obtulowicz, A., Păun, Gh.: (In Search of) Probabilistic P Systems. Biosystems 70(2), (2003), 107–121.
8. Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical Probabilistic P Systems. International Journal of Foundations of Computer Science 17(1), (2006), 183–204.
9. Păun, Gh.: Membrane Computing – An Introduction. Springer-Verlag, Berlin Heidelberg New York (2002).
10. Romero-Jiménez, A., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M. J.: The Growth of Branching Structures with P Systems. In Graciani–Díaz, C., Păun, Gh., Romero–Jiménez, A., Sancho–Caparrini, F. (eds.): Proceedings of the Fourth Brainstorming Week on Membrane Computing, Vol. II. Fénix Editora, Sevilla, (2006), 253–265.
11. Romero-Jiménez, A., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Graphical Modelling of Higher Plants Using P Systems In H.J. Hoogeboom, Gh. Păun, G. Rozenberg, A. Salomaa (eds.) Membrane Computing, Seventh International Workshop. WMC 2006, Lecture Notes in Computer Science, 4361, (2006), 496-506.
12. Smith, A.R.: Plants, Fractals and Formal Languages. Computer Graphics, 18(3), (1984), 1–10.
13. The P Systems webpage `http://ppage.psystems.eu/`

Computing by Carving with P Systems. A First Approach*

José M. Sempere

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Valencia, Spain
E-mail: jsempere@dsic.upv.es

Summary. In this work, we propose a P system which carries out computing by carving. Computing by carving was proposed by Gh. Păun as a technique to generate formal languages which can even be non recursively enumerable. Hence, it can be considered a hypercomputational technique. Here, we propose a first scheme based on P systems in order to perform computing by carving any formal language. So, the paper shows indirectly that these systems, under certain assumptions, can be considered a model for hypercomputation.

1 Introduction

Computing by carving is a computational strategy to generate formal languages proposed by Gh. Păun in 1999 [4]. This technique has been proved to generate even non recursively enumerable languages, hence it can be considered a hypercomputational technique. Furthermore, in the same work, Păun proved that it can be a used as a solution to language approximation problems. Here, we will propose a membrane system architecture to perform computing by carving. Hence, we indirectly prove that membrane systems can be considered hypercomputational models.

Hypercomputational models have been proposed along the time that solve some problems proved to be unsolvable by classical Turing machines. Most of these models need some kind of infinite resources (infinite tape alphabets, sets of states, etc.) as pointed out in [10]. Here, we will introduce infinite membrane regions and objects as a source for hypercomputing with P systems.

The structure of this work is as follows: In the following section, we will introduce basic concepts about formal language theory and membrane computing. Then, we will introduce the basic aspects and results about computing by carving. In section 4, we will propose a P system to perform computing by carving.

* Work supported by the Spanish Ministerio de Educación y Ciencia under project TIN2007-60769

We will discuss different approximations to make so by using membrane creation and other ways to obtain potentially infinite resources. Finally, we will draw some conclusions and we will point out new problems for future research.

2 Basic concepts on formal language theory and membrane computing

We will introduce some basic concepts from formal language theory according to [2, 7] and from membrane computing according to [5].

An alphabet Σ is a finite nonempty set of elements named symbols. A string defined over Σ is a finite ordered sequence of symbols from Σ. The infinite set of all the strings defined over Σ will be denoted by Σ^*. Given a string $x \in \Sigma^*$ we will denote its length by $|x|$. The empty string will be denoted by λ and Σ^+ will denote $\Sigma^* - \{\lambda\}$. A language L defined over Σ is a set of strings from Σ. The difference between two languages L_1 and L_2 is defined by $L_1 - L_2 = \{x \in L_1 : x \notin L_2\}$.

A *generalized sequential machine* can be defined by the tuple $T = (\Sigma, \Delta, Q, R, q_0, F)$, where Σ and Δ are aphabets, Q is a finite states set, $R \subseteq Q \times \Sigma^* \times \Delta^* \times Q$ is a finite transition relation, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is a set of final (or acceptance) states. The machine takes an initial string $x \in \Sigma^*$ and, after applying the transitions defined by T, it obtains an output string $y \in \Delta^*$ whenever it finishes in a final state. Then, we will say that T performs a function g such that $g(x) = y$.

A general P system of degree m is a construct

$$\Pi = (V, T, C, \mu, w_1, \cdots, w_m, (R_1, \rho_1), \cdots, (R_m, \rho_m), i_0),$$

where:

- V is an alphabet (the *objects*)
- $T \subseteq V$ (the *output alphabet*)
- $C \subseteq V$, $C \cap T = \emptyset$ (the *catalysts*)
- μ is a membrane structure consisting of m membranes
- w_i, $1 \leq i \leq m$, is a string representing a multiset over V associated with the region i
- R_i, $1 \leq i \leq m$, is a finite set of *evolution rules* over V associated with the ith region and ρ_i is a partial order relation over R_i specifying a *priority*.
  An evolution rule is a pair (u, v) (or $u \rightarrow v$) where u is a string over V and $v = v'$ or $v = v'\delta$ where v' is a string over
  $$\{a_{here}, a_{out}, a_{in_j} \mid a \in V, 1 \leq j \leq m\}$$
  and δ is an special symbol not in V (it defines the *membrane dissolving action*). From now on, we will denote the set *tar* by $\{here, out, in_k \mid 1 \leq k \leq m\}$.
- i_0 is a number between 1 and m and it specifies the *output* membrane of Π (in the case that it equals ∞ the output is read outside the system).

The language generated by Π in external mode ($i_0 = \infty$) is denoted by $L(\Pi)$ and it is defined as the set of strings that can be defined by collecting the objects that leave the system arranged in the leaving order (if several objects leave the system at the same time, then all permutations are allowed). The set of numbers that represent the objects in the output membrane i_0 will be denoted by $N(\Pi)$. Obviously, both sets $L(\Pi)$ and $N(\Pi)$ are defined only for *halting computations*.

One of the multiple variations of P systems is related to the modification of membrane structures. There have been several works in which these variants have been proposed (see, for example, [1, 3, 6]).

In the following, we enumerate some kinds of rules which are able to modify the membrane structure:

1. 2-division: $[_h a]_h \rightarrow [_{h'} b]_{h'} [_{h''} c]_{h''}$
2. Creation: $a \rightarrow [_h b]_h$
3. Dissolving: $[_h a]_h \rightarrow b$

The power of P systems with the previous operations and other ones (e.g., *exocytosis, endocytosis*, etc.) has been widely studied in the previously mentioned works and other papers.

3 Computing by carving

Păun introduced in [4] *computing by carving* as a technique to compute formal languages inspired by the search of solutions by filtering conditions in DNA computing. The main ingredients of computing by carving are the following:

1. A target language L
2. An initial couple of languages L_0 and L_1
3. An initial language M
4. A generalized sequential machine (gsm) g

The way of obtaining the target language L can be explained as follows: First, select a broader (regular) language M and an initial couple of (regular) languages L_0 and L_1 and calculate $L_{i+1} = g(L_i)$ for $i \geq 1$. The ith iteration of g over L_1 can be denoted by $g^i(L_1)$ and $g^*(L_1)$ will denote the language $\bigcup_{i \geq 0} g^i(L_1)$. Then L can be calculated as $L = M - (L_0 \cup g^*(L_1)$. If the latter condition holds, then L is said C-REG computable. Then, the triple (L_0, L_1, g) identifies the sequence that allows the calculation of L. Observe that M can be assumed to be Σ^*.

The following results can be found in [4]

Theorem 1. *Every recursively enumerable language $L \subseteq T^*$ can be written in the form $L = g^*(\{a_0\}) \cap T^*)$ where g is a gsm, depending on L, and a_0 is a fixed symbol not in T.*

Theorem 2. *There are C-REG computable languages which are not recursively enumerable.*

4 A P system for computing by carving

In this section, we propose a first approach to implement the components M, L_0, L_1 and g in order to compute a language L by carving. In general, we will work with sets of integers instead of languages of strings given that the connection of languages to sets of integers are very common in P systems.

The general scheme that we will initially follow in the proposed P system is shown in Fig. 1. First, the projections of the languages M, L_0 and L_1 can be generated by using P systems Π_M, Π_{L_0} and Π_{L_1}. Observe that any recursively enumerable language can be generated by a P system so they can be generated too. Moreover, we can take L_1 to be finite, even a *singleton* as shown in [4], so we can define the set of initial objects in Π_{L_1} to be exactly L_1. With respect to L_0 we can fix it to be Σ^* or the empty set. In both cases, it can be trivially generated even in a lexicographic order.

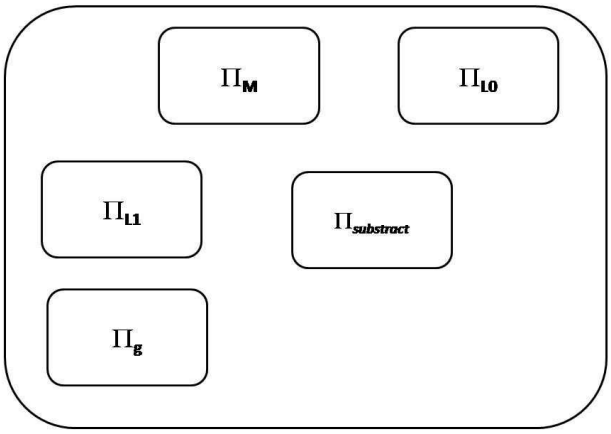


Fig. 1. Initial scheme for the P system

Once, we have generated a string in Π_M, Π_{L_0} or Π_{L_1} it is sent to $\Pi_{substract}$. Observe that, if we generate L_0 and L_1 in lexicographic order then we can make the difference in $\Pi_{substract}$ in lexicographic order too. The generalized sequential machine g can be applied by using a P system Π_g that receives objects from $\Pi_{substract}$. Once the generalized sequential machine has been applied in Π_g two different regions are created by using membrane division and membrane creation: Π_{g_2} and Π_{L_2}. Now, the new regions are used to calculate the second g iteration

over L_1 in Π_{g_2} and the language L_2 in Π_{L_2}. The second substraction over the set M is performed again in $\Pi_{substract}$.

This scheme can be generalized to obtain L_j. Observe that the P system structure in this case is shown in Figure 2.

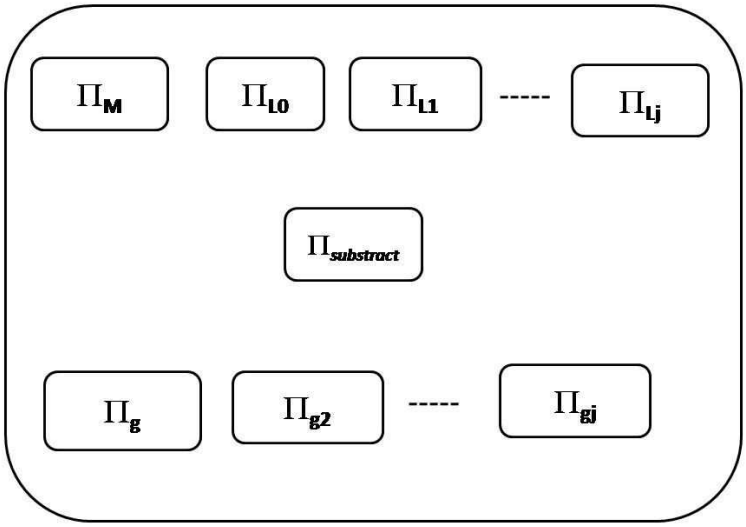


Fig. 2. Calculating L_j through jth iteration for the generalized sequential machine g

Some remarks about the proposed architecture:

1. The region corresponding to $\Pi_{substract}$ always contains the updated approximation to L. That is, when the computation is in progress, some strings will be sent from Π_M to $\Pi_{substract}$ and they will be deleted if the same string appears in any Π_{L_i}.
2. The whole process can be considered as an infinite time one, given that the generation of every L_i is infinite and so is for the updated approximation to L.
3. Here, the resources needed to compute L are infinite. Think about the number of regions and objects in every region. They will be unboundedly increased over the time.

5 Conclusions and future work

In this paper we have made a first approach to computing by carving with P systems. Here, we have only sketched the general ideas behind the full definition. So, we need to define the P systems carrying out the following tasks:

- The generation of any infinite (regular) language in lexicographic order.
- The application of the generalized sequential machine over strings and languages.
- The substraction between languages.

Every task referred before will be investigated in future works.

References

1. A. Alhazov, T.O. Ishdorj. *Membrane operations in P systems with active membranes.* In Proc. Second Brainstorming Week on Membrane Computing. TR 01/04 of RGNC. Sevilla University. pp 37-44. 2004.
2. J. Hopcroft, J. Ullman. Introduction to Automata Theory, Languages and Computation. Addison Wesley Publishing Co.,1979.
3. A. Păun. *On P systems with active membranes.* In Proc. of the First Conference on Unconventionals Models of Computation (UMC2K). pp 187-201. 2000.
4. Gh. Păun. (DNA) computing by carving. *Soft Computing* 3, pp 30-36. 1999
5. G. Păun. *Membrane Computing. An Introduction.* Springer. 2002.
6. G. Păun, Y. Suzuki, H. Tanaka, T. Yokomori. *On the power of membrane division on P systems.* Proc. Conf. on Words, Languages and Combinatorics (Kyoto). 2000.
7. G. Rozenberg, A. Salomaa (Eds.). Handbook of Formal Languages Vol. 1. Springer. 1997.
8. J.M. Sempere. P systems with external input and learning strategies. In *Preproceedings of the Workshop on Membrane Computing* (July 2003). A. Alhazov, C. Martín-Vide and Gh. Păun (editors), pp 445-448. 2003
9. J.M. Sempere. Covering reductions and degrees in P systems. In *Pre-proceedings of the Fifth Workshop on Membrane Computing* (June, 2004), pp 384-391. 2004
10. M. Stannet. Hypercomputational Models. In *Alan Turing: Life and Legacy of a Great Thinker.* C. Teuscher (*editor*). Springer. 2004

On the Computational Efficiency of Polarizationless Recognizer P Systems with Strong Division and Dissolution

Claudio Zandron[1], Alberto Leporati[1], Claudio Ferretti[1], Giancarlo Mauri[1], Mario J. Pérez-Jiménez[2]

[1] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
E-mails: {zandron,leporati,ferretti,mauri}@disco.unimib.it

[2] 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: marper@us.es

Summary. Recognizer P systems with active membranes have proven to be very powerful computing devices, being able to solve **NP**-complete decision problems in a polynomial time. However such solutions usually exploit many powerful features, such as electrical charges (polarizations) associated to membranes, evolution rules, communication rules, and strong or weak forms of division rules. In this paper we contribute to the study of the computational power of polarizationless recognizer P systems with active membranes. Precisely, we show that such systems are able to solve in polynomial time the **NP**-complete decision problem 3-SAT by using only dissolution rules and a form of strong division for non–elementary membranes, working in the maximal parallel way.

1 Introduction

Membrane systems (also known as *P systems*) have been introduced in [11] as a parallel, nondeterministic, synchronous and distributed model of computation inspired by the structure and functioning of living cells. The basic model consists of a hierarchical structure composed by several membranes, embedded into a main membrane called the *skin*. Membranes divide the Euclidean space into *regions*, that contain some *objects* (represented by symbols of an alphabet) and *evolution rules*. Using these rules, the objects may evolve and/or move from a region to a neighboring one. Usually, the rules are applied in a nondeterministic and maximally parallel way; moreover, all the objects that may evolve are forced to evolve. A *computation* starts from an initial configuration of the system and terminates

when no evolution rule can be applied. The result of a computation is the multiset of objects contained into an *output membrane*, or emitted from the skin of the system. An interesting subclass of membrane system is constituted by *recognizer* P systems, in which: (1) all computations halt, (2) only two possible outputs exist (usually named `yes` and `no`), and (3) the result produced by the system depends only upon its input, and is not influenced by the particular sequence of computation steps taken to produce it. For a systematic introduction on P systems we refer the reader to [14], whereas the latest information can be found in [24].

Since the introduction of membrane systems, many investigations have been performed on their computational properties: in particular, many variants have been proposed in order to study the contribution of various ingredients (associated with the membranes and/or with the rules of the system) to the achievement of the computational power of these systems. In this respect, it is known [19, 23, 5] that the class of all decision problems which can be solved in polynomial time by a family of recognizer P systems that use only basic rules, that is, evolution, communication and rules involving membrane dissolution, coincides with the standard complexity class **P**. Hence, in order to efficiently solve **NP**-complete problems by means of P systems it seems necessary to be able to construct an exponential workspace, expressed by the number of membranes, in polynomial time. In particular, two features have proven to be of paramount importance in establishing whether a membrane system is able to solve **NP**-complete decision problems in polynomial time: membrane division and dissolution. The former is inspired from the biological process called *mitosis*: using *division rules* we can duplicate a given membrane that contains one specified symbol, possibly rewriting this symbol in a different way in each of the cells produced by the process. All the other symbols, as well as the rules, which are contained in the original cell are copied unaltered into each of the resulting cells. As for the membranes eventually contained in the original cell, we can make the following distinctions. If no membrane occurs, the we say that the division is *elementary*; if at least one membrane occurs, then the division is non elementary, and we have to specify how the membranes are distributed to the resulting membranes. If all the membranes are copied to each of the resulting membranes, then we have a *weak* (non-elementary) division; if, on the other hand, we can choose what membranes are copied into each of the resulting membranes, then we have *strong* (non-elementary) division. Membrane dissolution is performed by rules that simply dissolve the surrounding membrane when a specified symbol occurs.

Recognizer P systems with active membranes (using division rules and, eventually, polarizations associated to membranes) have thus been successfully used to efficiently solve **NP**-complete problems. The first solutions were given in the so called *semi-uniform* setting [13, 23, 7, 9], which means that we assume the existence of a deterministic Turing machine that, for every instance of the problem, produces in polynomial time a description of the P system that solves such an instance. The solution is computed in a *confluent* manner, meaning that the instance given in input is positive if and only if every computation of the P system

associated with it is an accepting computation. Another way to solve **NP**-complete problems by means of P systems is by considering the *uniform* setting, in which all the instances of the problem are given in input — encoded in an appropriate way — to the same P system and then solved by it. Sometimes, a uniform solution to a decision problem Q is provided by defining a family $\{\Pi_Q(n)\}_{n\in\mathbb{N}}$ of P systems such that for every $n \in \mathbb{N}$ the system $\Pi_Q(n)$ reads in input an encoding of any possible instance of size n, and solves it. P systems with active membranes have thus been successfully used to design uniform polynomial-time solutions to some well-known **NP**-complete problems, such as SAT [20], SUBSET SUM [17], KNAPSACK [18], PARTITION [6] and the COMMON ALGORITHMIC PROBLEM [21].

All the papers mentioned above deal with P systems with three polarizations that use only division rules for elementary membranes (in [22] also division for non–elementary membranes is permitted, and in this way a semi–uniform solution to the **PSPACE**-complete problem QSAT is provided), and working in the *maximal parallel* way. As shown in [1], the number of polarizations can be decreased to two without loss of efficiency.

Since by using all these features (membrane division, dissolution and polarizations) we can solve **NP**-complete problems, we have a model of computation which is considered too powerful from the point of view of traditional complexity theory. Hence a research direction of a clear interest is to selectively remove one or more of these features and see whether the computation power changes, that is, investigating for what combinations of features we are still able to obtain polynomial time solutions to computationally hard problems and what features, once removed, only allow to obtain polynomial time solutions to tractable problems, in the classical sense. In this direction, in [15] a conjecture was formulated by Gh. Păun about the computational power of polarizationless P systems with active membranes and working in the maximally parallel mode, stating that such systems can only solve decision problems that are in **P** (by using only elementary division), and some partial answers were given in [8]. Also, in [4] the computational power of recognizer P systems with active membranes but without electrical charges and dissolution rules was investigated, establishing that they characterize the complexity class **P**.

In this paper we continue this research line, showing that polarizationless P systems with active membranes that use strong division for non–elementary membranes and dissolution rules, working in the maximal parallel way, are able to solve in polynomial time the **NP**-complete problem 3-SAT. This result provides further partial answers to Păun's conjecture, establishing that neither evolution nor communication rules, and no electrical charges are needed to solve **NP**-complete problems, provided that we can use strong division rules for non–elementary membranes (as well as dissolution rules, otherwise we would fall in the case considered in [4]).

The paper is organized as follows. In Sections 2 and 3 we recall the definition of polarizationless recognizer P systems with active membranes, thus establishing our model of computation, and we recall the definition of the **NP**-complete decision problem 3-SAT. In Section 4 we show how the systems we are considering are able

to solve the 3-SAT problem. Finally, Section 5 contains the conclusions and some directions for further research.

2 Polarizationless recognizer P systems with active membranes

Usually, P systems with active membranes are defined in the literature with three electrical charges (also called *polarizations*) associated with membranes (even thought two charges suffice, as proved in [1]) to control the application of the rules, which can be of the following types: *evolution rules*, by which single objects evolve to a multiset of objects, *communication rules*, by which an object is introduced in or expelled from a membrane, and possibly changed to another object while performing this operation, *dissolution rules*, by which a membrane is dissolved under the influence of an object, that can also be modified during this operation, and *membrane division rules* (both for elementary and non–elementary membranes, or only for elementary membranes). However, in this paper we will consider *polarizationless* P systems with active membranes, that is, P systems in which no electrical charge is associated with any membrane.

Formally, a P system with polarizationless active membranes of the initial degree $n \geq 1$ is a tuple of the form $\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_n, R, h_0)$, where:

1. Γ is the alphabet of objects;
2. H is a finite set of labels for membranes;
3. μ is a membrane structure, consisting of n membranes being labelled with elements of H;
4. $\mathcal{M}_1, \ldots, \mathcal{M}_n$ are strings over Γ, describing the multisets of objects placed in the n initial regions of μ;
5. R is a finite set of developmental rules, of the following forms:
   (a) $[\, a \to v \,]_h$, for $h \in H$, $a \in \Gamma$, $v \in \Gamma^*$ (object evolution rules);
   (b) $a[\ \ ]_h \to [\, b\,]_h$, for $h \in H$, $a, b \in \Gamma$ (in communication rules);
   (c) $[\, a\,]_h \to b[\ \ ]_h$, for $h \in H$, $a, b \in \Gamma$ (out communication rules);
   (d) $[\, a\,]_h \to b$, for $h \in H$, $a, b \in \Gamma$ (dissolution rules);
   (e) $[\, a\,]_h \to [\, b\,]_h [\, c\,]_h$, for $h \in H$, $a, b, c \in \Gamma$ (weak division rules for elementary or non–elementary membranes);
   (f) $h_0 \in H$ or $h_0 = env$ indicates the output region (in the latter case, usually h_0 does not appear in the description of the system).

We can also consider rules of the form $[\,[\ \ ]_{h_1}[\ \ ]_{h_2}\,]_{h_3} \to [\,[\ \ ]_{h_1}\,]_{h_3}[\,[\ \ ]_{h_2}\,]_{h_3}$, where h_1, h_2, h_3 are labels from H: if the membrane with label h_3 contains other membranes than those with labels h_1, h_2, these membranes and their contents are duplicated and placed in both new copies of the membrane h_3; all membranes and objects placed inside membranes h_1, h_2, as well as the objects from membrane h_3 placed outside membranes h_1 and h_2, are reproduced in the new copies of membrane h_3. These rules are called *strong division rules for non–elementary membranes*.

As usual, a computation starts in the *initial configuration*, which is given by the membrane structure μ and the strings (multisets) $\mathcal{M}_1, \ldots, \mathcal{M}_n$ of objects initially present in the n regions of μ. Using the *maximally parallel manner*, at each computation step (a global clock is assumed) in each region of the system we apply the rules in such a way that no further rule can be applied to the remaining objects or membranes. In each step, each object and each membrane can be involved in only one rule. The application of a maximal set of rules during a computation step produces a new configuration of the system. A *computation* is a sequence $C_0, C_1, \ldots$ of configurations such that C_0 is the initial configuration described above, and for all $i \geq 1$ the configuration C_i is obtained from C_{i-1} by applying a maximal set of rules as described above. Note that a computation may be finite or infinite; in the former case we require that the last element of the sequence is an *halting* configuration, that is, a configuration in which no rule can be applied anywhere in the system. A halting computation provides a result encoded by the objects present in region h_0 at the end of the computation; this is a region of the system if $h_0 \in H$ (and in this case, for a computation to be successful, exactly one membrane with label h_0 should be present in the halting configuration), or it is the environment if $h_0 = env$. An infinite computation produces no result.

A *recognizer* P system with active membranes is obtained from the definition given above by assuming that the system halts on every computation and produces one of two possible outputs, that are usually denoted by `yes` and `no`. A further requirement is that the system is *confluent*, that is, for any given input configuration, all the computations that can start with such a configuration end by producing the same output. In this way, we can say that a recognizer P system with active membranes recognizes the language which is composed by the strings that encode the initial configurations that produce `yes` as a result. By considering the trivial bijection existing between these languages and decision problems, we can also say that a recognizer P system solves the decision problem whose positive instances are associated with initial configurations of the system that produce the output `yes` in h_0.

We denote by $\mathcal{AM}^0$ the class of polarizationless recognizer P systems with active membranes, and we denote by $\mathcal{AM}^0(\alpha, \beta, \gamma, \delta)$, where $\alpha \in \{-d, +d\}$, $\beta \in \{-ne, +new, +nes\}$, $\gamma \in \{-ev, +ev\}$, and $\delta \in \{-comm, +comm\}$ the class of all recognizer P systems with polarizationless active membranes such that: (a) if $\alpha = +d$ (resp., $\alpha = -d$) then dissolution rules are permitted (resp., forbidden); (b) if $\beta \in \{+new, +nes\}$ (resp., $\beta = -ne$) then division rules for elementary and non–elementary membranes, weak or strong (resp., only division rules for elementary membranes) are permitted; (c) if $\gamma = +ev$ (resp., $\gamma = -ev$) then evolution rules are permitted (resp., forbidden); (d) if $\delta = +comm$ (resp., $\delta = -comm$) then communication rules are permitted (resp., forbidden).

The class of all decision problems which can be solved in uniform (resp., semi–uniform) way, and in polynomial time by a family $\mathcal{R}$ of recognizer membrane systems is denoted by $\mathbf{PMC}_{\mathcal{R}}$ (resp., $\mathbf{PMC}^*_{\mathcal{R}}$). The following inclusions directly follow from these definitions.

Proposition 1. *For all* $\alpha \in \{-d, +d\}$, $\beta \in \{-ne, +new, +nes\}$, $\gamma \in \{-ev, +ev\}$, $\delta \in \{-comm, +comm\}$ *and* $\varepsilon \in \{*, \lambda\}$*:*

1. $\mathbf{PMC}_{\mathcal{AM}^0(\alpha,\beta,\gamma,\delta)} \subseteq \mathbf{PMC}^{*}_{\mathcal{AM}^0(\alpha,\beta,\gamma,\delta)}$
2. $\mathbf{PMC}^{\varepsilon}_{\mathcal{AM}^0(-d,\beta,\gamma,\delta)} \subseteq \mathbf{PMC}^{\varepsilon}_{\mathcal{AM}^0(+d,\beta,\gamma,\delta)}$
3. $\mathbf{PMC}^{\varepsilon}_{\mathcal{AM}^0(\alpha,-ne,\gamma,\delta)} \subseteq \mathbf{PMC}^{\varepsilon}_{\mathcal{AM}^0(\alpha,+new,\gamma,\delta)}$
4. $\mathbf{PMC}^{\varepsilon}_{\mathcal{AM}^0(\alpha,-ne,\gamma,\delta)} \subseteq \mathbf{PMC}^{\varepsilon}_{\mathcal{AM}^0(\alpha,+nes,\gamma,\delta)}$
5. $\mathbf{PMC}^{\varepsilon}_{\mathcal{AM}^0(\alpha,\beta,-ev,\delta)} \subseteq \mathbf{PMC}^{\varepsilon}_{\mathcal{AM}^0(\alpha,\beta,+ev,\delta)}$
6. $\mathbf{PMC}^{\varepsilon}_{\mathcal{AM}^0(\alpha,\beta,\gamma,-comm)} \subseteq \mathbf{PMC}^{\varepsilon}_{\mathcal{AM}^0(\alpha,\beta,\gamma,+comm)}$

where $\varepsilon = *$ *(resp.,* $\varepsilon = \lambda$*, the empty string) means that the complexity classes are associated with semi–uniform (resp., uniform) solutions.*

Also, using this notation, Păun's conjecture (problem **F** in [15]) can be restated as follows:

$$\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0(+d,-ne,+ev,+comm)} = \mathbf{PMC}^{*}_{\mathcal{AM}^0(+d,-ne,+ev,+comm)}$$

As stated in the Introduction, results in [4] and [8] proved the following theorem, considering a reachability problem (is the state in which the symbol `yes` is expelled to the environment reachable?) defined on the so called *dependency graph*. We refer the reader to [4] and [8] for further details on the proofs.

Theorem 1. *For all* $\beta \in \{-ne, +new, +nes\}$*,*

$$\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0(-d,\beta,+ev,+comm)} = \mathbf{PMC}^{*}_{\mathcal{AM}^0(-d,\beta,+ev,+comm)}$$

This result holds for systems working in the maximal parallel manner; in [8] also systems working with *minimal parallelism* were considered, but in this paper we will not address them.

3 The 3-SAT problem

Let us now consider the **NP**-complete decision problem 3-SAT [3, p. 46]. The instances of 3-SAT depend upon two parameters: the number n of variables, and the number m of 3-clauses. We recall that a *clause* is a disjunction of literals, occurrences of x_i or $\neg x_i$, built on a given set $X = \{x_1, x_2, \ldots, x_n\}$ of boolean variables. A *3-clause* is a clause that contains exactly three literals. In what follows we will require that no repetitions of the same literal may occur in any clause. Without loss of generality we can also avoid the clauses in which both the literals x_i and $\neg x_i$, for any $1 \leq i \leq n$, occur. An *assignment* of the variables $x_1, x_2, \ldots, x_n$ is a mapping $a : X \to \{0, 1\}$ that associates to each variable a truth value. The number of all possible assignments to the variables of X is 2^n. We say that an assignment *satisfies* the clause C if, assigned the truth values to all the variables which occur in C, the evaluation of C (considered as a boolean formula) gives 1 (*true*) as a result.

We can now formally state the 3-SAT problem as follows.

Problem 1. NAME: 3-SAT.

- INSTANCE: a set $C = \{C_1, C_2, \ldots, C_m\}$ of 3-clauses, built on a finite set $\{x_1, x_2, \ldots, x_n\}$ of boolean variables.
- QUESTION: is there an assignment of the variables $x_1, x_2, \ldots, x_n$ that satisfies all the clauses in C?

In what follows we will sometimes equivalently say that an instance of 3-SAT is a propositional formula $\gamma_{n,m} = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, expressed in the conjunctive normal form as a conjunction of m clauses, where each clause is a disjunction of three literals built using the boolean variables $x_1, x_2, \ldots, x_n$. With a little abuse of notation, from now on we will denote by 3-SAT(n, m) the set of instances of 3-SAT which have n variables and m clauses.

The reason for which we are here interested into 3-SAT (rather that with the more generic problem SAT, see [3, p. 39], where we put no upper bound on the number of literals that may appear in each clause) is that the number of possible 3-clauses which can be built using n boolean variables is $2n \cdot (2n-2) \cdot (2n-4) \in \Theta(n^3)$, a polynomial quantity with respect to n. This quantity is obtained by looking at a 3-clause as a triple, and observing that each component of the triple may contain one of the $2n$ possible literals, with the constraints that we do not allow neither the repetition of literals in the clauses, nor the use of the same variable two or three times in a clause. On the other hand, an instance of SAT may have a number of clauses which is exponential in n, since for every $i \in \{1, 2, \ldots, n\}$ either variable x_i or its negation (or none of them) can appear in a clause, yielding to 3^n possible combinations.

4 Solving 3-SAT with strong division and dissolution rules

In this section we propose a semi–uniform family $\{\Pi_{3SAT}(\gamma_{n,m})\}_{\gamma_{n,m} \in 3SAT(n,m)}$ of polarizationless recognizer P systems with active membranes that solves the **NP**-complete decision problem 3-SAT by using only membrane dissolution rules and a form of strong division rules for non–elementary membranes. Precisely, for every instance $\gamma_{n,m}$ of 3-SAT(n, m) we show how to build the system $\Pi_{3SAT}(\gamma_{n,m})$ that solves such an instance. Our result can be summarized by the statement of the following theorem.

Theorem 2. *3-SAT* $\in \mathbf{PMC}^{*}_{\mathcal{AM}^0(+d,+nes,-ev,-comm)}$.

Proof. Let $\gamma_{n,m} = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be an instance of 3-SAT(n, m), built using the boolean variables $x_1, x_2, \ldots, x_n$, and let $\Pi_{3SAT}(\gamma_{n,m})$ be the recognizer P system associated (in the semi–uniform framework) to $\gamma_{n,m}$, whose initial configuration is illustrated in Figure 1. The system is composed by m outer membranes (not counting the skin membrane) which are associated with the clauses of $\gamma_{n,m}$. Precisely, the membrane immediately contained in the skin is associated with clause C_m and contains membrane C_{m-1}, which is associated with the namesake clause;

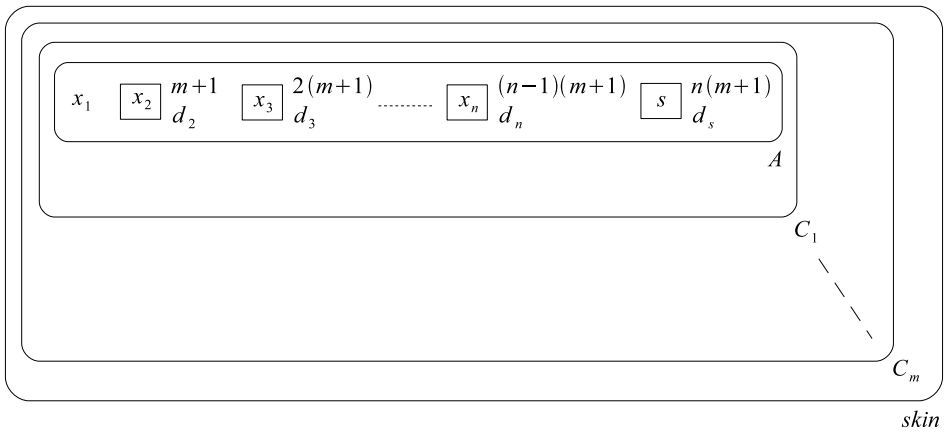


Fig. 1. Initial configuration of the system $\Pi_{3SAT}(\gamma_{n,m})$ that solves the instance $\gamma_{n,m}$ of 3-SAT(n, m)

on its turn, membrane C_{m-1} contains a membrane labelled with C_{m-2}, and so on, until we reach membrane C_1 that contains a membrane labelled with A, that will be used to generate all the possible assignments to $x_1, x_2, \ldots, x_n$. Membrane A contains the object x_1 (that represents the namesake variable) as well as n hierarchies of nested membranes. As depicted in Figure 2, the notation $\boxed{x_i}^{k}_{d_i}$ that we

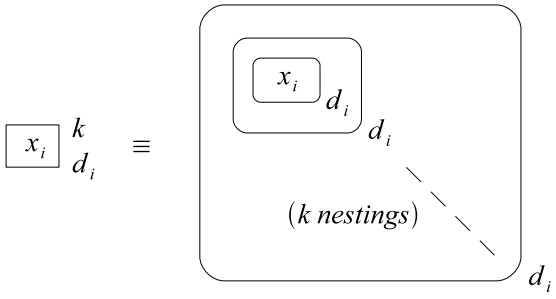


Fig. 2. The hierarchies of nested membranes used in the system depicted in Figure 1 to perform the correct sequence of membrane divisions

have adopted in Figure 1 indicates that symbol x_i is surrounded by k membranes, nested one into the other, all labelled by d_i. In this way, we can operate on membrane A through a rule which is activated by x_1 and, in the meanwhile, dissolve one membrane in each of the subsystems contained in A. After $m + 1$ steps x_2 emerges and activates another rule of A, and so on, until symbol s emerges and starts another phase of computation.

The computation of the system is composed by two phases: the *generation stage* and the *verification stage*. During the generation stage, 2^n copies of the subsystem contained into the skin of the initial configuration depicted in Figure 1 are produced, where in each copy membrane A contains an encoding of one of the possible assignments to $x_1, x_2, \ldots, x_n$. Such a phase is performed by the following rules:

1. $[\,[\,]_A[\,]_A\,]_{C_1} \rightarrow [\,[\,]_A\,]_{C_1}\,[\,[\,]_A\,]_{C_1}$
2. $[\,[\,]_{C_{i-1}}[\,]_{C_{i-1}}\,]_{C_i} \rightarrow [\,[\,]_{C_{i-1}}\,]_{C_i}\,[\,[\,]_{C_{i-1}}\,]_{C_i}$ for all $i = 2, 3, \ldots, m$
3. $[\,x_j\,]_A \rightarrow [\,t_j\,]_A\,[\,f_j\,]_A$ for all $j = 1, 2, \ldots, n$
4. $[\,x_j\,]_{d_j} \rightarrow x_j$ for all $j = 1, 2, \ldots, n$
5. $[\,s\,]_{d_s} \rightarrow s$
6. $[\,s\,]_A \rightarrow \texttt{yes}$
7. $[\,t_j\,]_{C_i} \rightarrow t_j$ if $x_j \in C_i$, $[\,f_j\,]_{C_i} \rightarrow f_j$ if $\neg x_j \in C_i$, where $1 \leq i \leq m$

Rules 1 and 2 are strong division rules for non–elementary membranes: whenever a membrane C_i contains two membranes at their immediately inner level, it divides and each of the resulting copies contains one of the previous inner membranes. Rule 3 is used to generate the assignments: when the symbol x_j, for $j \in \{1, 2, \ldots, n\}$, occurs in membrane A then A divides; in one of the resulting copies the symbol x_j is rewritten to t_j, indicating the fact that we are assigning the value TRUE to the boolean variable x_j. Similarly, in the other copy of A the symbol x_j is rewritten to f_j, indicating that the boolean value FALSE is assigned to x_j. In order to control the order of application of division rules during the generation phase, only one symbol x_j occurs in membrane A every $m + 1$ computation steps. In this way we first divide membrane A, assigning the two boolean values TRUE and FALSE to x_j as described above; then, rule 1 can be applied, thus duplicating membrane C_1.

In the subsequent $m - 1$ computation steps, membranes $C_2, C_3, \ldots, C_m$ are duplicated exactly in this order thanks to rules 2. Figure 3 depicts the first steps of this process for an instance containing $n = 2$ variables and $m = 2$ clauses (note that this example is conceived only for illustrative purposes, since at least three boolean variables are needed to build valid 3-clauses).

The rules are applied in the maximal parallel manner. In particular, at every computation step one membrane labelled with d_j, for each $j \in \{1, 2, \ldots, n\}$ such that membrane d_j still occurs in the system, is dissolved. In this way, a symbol x_j emerges in membrane A just after the assignment to x_{j-1} and all the subsequent duplications of membranes $C_1, C_2, \ldots, C_m$ have been performed. By using the same mechanism, symbol s emerges in membrane A after $n(m + 1)$ steps, that is, after all the assignments to $x_1, x_2, \ldots, x_n$ and all the duplications of membranes $C_1, C_2, \ldots, C_m$ have been performed. In practice, the construct composed by $n(m+1)$ nested membranes, all labelled with d_s, together with the symbol s into the innermost membrane and the dissolution rule $[\,s\,]_{d_s} \rightarrow s$, implement a counter whose initial value is nm and which is decremented each time the dissolution rule is applied.

When the symbol s appears in A then $n(m + 1)$ computation steps have been performed, that is, the generation stage has ended and the verification stage can

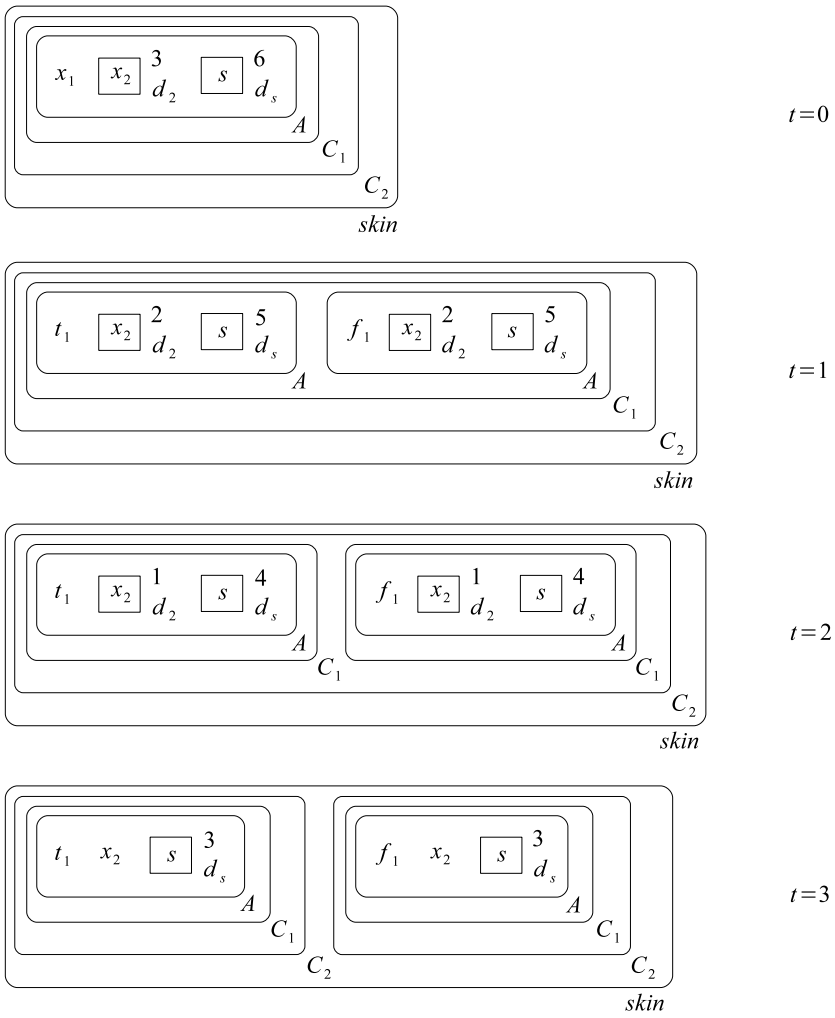


Fig. 3. First steps of the generation stage of a system designed to work on two clauses, built using two boolean variables. For reasons of space, also in this figure we have used the abbreviation depicted in Figure 2

start. All the copies of membrane A are dissolved by executing rule 6 (which also changes s to `yes`), so that all the objects t_j and f_j that represent the truth values of $x_1, x_2, \ldots, x_n$ can reach the corresponding membrane C_1 and activate its rules. These rules, of type 7, depend upon the instance $\gamma_{n,m}$ of 3-SAT(n, m) we are solving. For example, assume that the first clause of $\gamma_{n,m}$ is $C_1 = x_1 \vee \neg x_3 \vee x_4$. Then, membranes C_1 will contain the following dissolution rules:

$$[\, t_1 \,]_{C_1} \to t_1$$
$$[\, f_3 \,]_{C_1} \to f_3$$
$$[\, t_4 \,]_{C_1} \to t_4$$

In this way, a membrane labelled with C_1 is dissolved if and only if at least one of the objects t_j and f_j that encode the assignment satisfy the clause. If no object satisfy the clause then the computation in that subsystem halts; on the contrary, if the assignment under consideration satisfies C_1 then by dissolving membrane C_1 the objects t_j and f_j that encode the assignments are released to membrane C_2. Then, the rules that correspond to clause C_2 are executed; if the assignment satisfies also C_2 then the corresponding membrane is dissolved and the computation continues in membrane C_3, otherwise membrane C_2 is not dissolved and the computation halts in that subsystem. If an assignment satisfies all the clauses of $\gamma_{n,m}$ then it will dissolve all the membranes $C_1, C_2, \ldots, C_m$, and the objects that represent the assignment will reach the skin membrane, that we consider as the output membrane. Hence, the instance $\gamma_{n.m}$ of 3-SAT(n, m) solved by the system is positive if and only if in the halting configuration (in which no rule can be applied) at least one symbol occurs in the region enclosed by the skin membrane (equivalently, if at least one copy of symbol `yes` occurs in such a region).

As stated above, we have focused our attention on the 3-SAT problem because the number m of clauses is $O(n^3)$. It is apparent that the number of computation steps of the system $\Pi_{3SAT}(\gamma_{n,m})$ we have just described is $\Theta(n(m+1)+m) \subseteq O(n^4)$. The number of membranes in the initial configuration of the system is:

$$\sum_{i=1}^{n} i(m+1) + m + 2 = (m+1)\,\frac{n(n+1)}{2} + m + 2$$
$$\in \Theta(n^2 m) \subseteq O(n^5)$$

a polynomial quantity in n. The total number of rules is $2n + m + 3 \in O(n^3)$, and the initial number of objects is $n + 1 \in \Theta(n)$.

For the sake of completeness, please note that we could enlarge this system so to always produce exactly one output, being it either `yes` or `no`, just by adding an object b, initially put in the leaf membrane of a series of $n(m+1)+1$ membranes, a second outermost membrane (we can call it "external skin") enclosing our system and this new series of nested membranes, and finally adding a set of rules which first let b move toward the external skin and eventually change it to `no`. Notice that, in case of a positive answer, the object `yes` arrives to skin membrane one step before the object b does.

5 Conclusions and directions for future research

For every possible instance $\gamma_{n,m}$ of 3-SAT(n, m), having m clauses built on the boolean variables $x_1, x_2, \ldots, x_n$, we have shown how to build a polarizationless

recognizer P system $\Pi_{3SAT}(\gamma_{n,m})$ with active membranes that determines whether $\gamma_{n,m}$ is positive, that is, whether there exists an assignment to the variables $x_1, x_2, \ldots, x_n$ that satisfies all the clauses $C_1, C_2, \ldots, C_m$ of $\gamma_{n,m}$. The system works in the maximal parallel manner and, besides using no electrical charges associated with the membranes, it does not use neither evolution nor communication rules. However, it uses a form of strong division rules for non–elementary membranes, that allow to divide the content of the membrane which is being duplicated among the two resulting copies of the membrane. We can summarize the result exposed in this paper by saying that 3-SAT $\in \mathbf{PMC}^{*}_{\mathcal{AM}^0(+d,+nes,-ev,-comm)}$.

A first related question that comes to our mind is the following: is the class $\mathbf{PMC}^{*}_{\mathcal{AM}^0(+d,+nes,-ev,-comm)}$ closed under polynomial reductions? If so, any problem in **NP** could (at least, in principle) be transformed to 3-SAT by a polarizationless P system with active membranes that performs its computations without leaving this class. As a result, we could conclude that $\mathbf{NP} \subseteq \mathbf{PMC}^{*}_{\mathcal{AM}^0(+d,+nes,-ev,-comm)}$. Another question of clear interest is: can we use only *weak* division rules? Stated otherwise: is 3-SAT (or some other **NP**-complete problem) in $\mathbf{PMC}^{*}_{\mathcal{AM}^0(+d,+new,-ev,-comm)}$?

Acknowledgements

The ideas exposed in this paper emerged during the Sixth Brainstorming Week on Membrane Computing, held in Seville from February 4 to February 8, 2008.

The work of the authors was partially supported by the project "Azioni Integrate Italia-Spagna - Theory and Practice of Membrane Computing" (Acción Integrada Hispano-Italiana HI 2005-0194).

References

1. A. Alhazov, R. Freund. On efficiency of P systems with active membranes and two polarizations. In G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (eds.), *Membrane Computing, 5th International Workshop, WMC 2004*, Revised Selected and Invited Papers, LNCS 3365, Springer-Verlag, Berlin, 2005, pp. 81–94.
2. A. Alhazov, M.J. Pérez-Jiménez. Uniform solution to QSAT using polarizationless active membranes. In M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero (eds.), *Proceedings of the Fourth Brainstorming Week on Membrane Computing*, Volume I, Fénix Editora, Sevilla, 2006, pp. 29–40.
3. M.R. Garey, D.S. Johnson. *Computers and Intractability. A Guide to the Theory on NP–Completeness*. W.H. Freeman and Company, 1979.
4. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero. On the power of dissolution in P systems with active membranes. In R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa (eds.) *Membrane Computing, 6th International Workshop, WMC 2005*, Revised Selected and Invited Papers, LNCS 3850, Springer-Verlag, Berlin, 2006, pp. 224–240.

5. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero, A. Romero-Jiménez. Characterizing tractability by cell-like membrane systems. In K.G. Subramanian, K. Rangarajan, M. Mukund (eds.), *Formal models, languages and applications*, World Scientific, Series in Machine Perception and Artificial Intelligence, Vol. 66, 2006, pp. 137–154.
6. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. A fast P system for finding a balanced 2–partition. *Soft Computing*, 9(9):673–678, 2005.
7. S.N. Krishna, R. Rama. A variant of P systems with active membranes: Solving **NP**-complete problems. *Romanian Journal of Information Science and Technology*, 2(4):357–367, 1999.
8. G. Mauri, M.J. Pérez-Jiménez, C. Zandron. On a Păun conjecture in membrane systems. In J. Mira and J.R. Álvarez (eds.), *Second International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2007*, Part I, LNCS 4527, Springer-Verlag, Berlin, 2007, pp. 180–192.
9. A. Obtulowicz. Deterministic P systems for solving SAT problem. *Romanian Journal of Information Science and Technology*, 4(1–2):551–558, 2001.
10. C.H. Papadimitriou. *Computational Complexity*, Addison-Wesley, 1994.
11. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 1(61):108–143, 2000. See also Turku Centre for Computer Science – TUCS Report No. 208, 1998. Available at: `http://www.tucs.fi/Publications/techreports/TR208.php`
12. Gh. Păun. Computing with membranes. An introduction. *Bulletin of the EATCS*, 67:139–152, February 1999.
13. Gh. Păun. P systems with active membranes: Attacking **NP**-complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 2001.
14. Gh. Păun. *Membrane Computing. An Introduction.* Springer–Verlag, Berlin, 2002.
15. Gh Păun. Further twenty six open problems in membrane computing. In M.A. Gutiérrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan (eds.), *Proceedings of the Third Brainstorming Week on Membrane Computing*, Fénix Editora, Sevilla, 2005, pp. 249–262.
16. G. Păun, G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287(1), 2002, pp. 73–100.
17. M.J. Pérez-Jiménez, A. Riscos-Núñez. Solving the SUBSET SUM problem by active membranes. *New Generation Computing*, 23(4):367–384, 2005.
18. M.J. Pérez-Jiménez, A. Riscos-Núñez. A linear–time solution to the KNAPSACK problem using P systems with active membranes. In C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa (eds.), *Membrane Computing, 4th International Workshop, WMC 2003*, Revised Selected and Invited Papers, LNCS 2933, Springer-Verlag, Berlin, 2004, pp. 250–268.
19. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini. The **P** versus **NP** problem through cellular computing with membranes. In N. Jonoska, Gh. Păun, G. Rozenberg (eds.), *Aspects of Molecular Computing*, LNCS 2950, Springer-Verlag, Berlin, 2004, pp. 338–352.
20. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini. A polynomial complexity class in P systems using membrane division. In E. Csuhaj-Varjú, C. Kintala, D. Wotschke, G. Vaszil (eds.), *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003*, Computer and Automation Research Institute of the Hungarian Academy of Sciences, Budapest, 2003, pp. 284–294.

21. M.J. Pérez-Jiménez, F.J. Romero-Campero. Attacking the Common Algorithmic Problem by recognizer P systems. In M. Margenstern (ed.), *Machines, Computations and Universality*, LNCS 3354, Springer-Verlag, Berlin, 2005, pp. 304–315.
22. P. Sosik. The computational power of cell division. *Natural Computing*, 2(3):287–298, 2003.
23. C. Zandron, C. Ferretti, G. Mauri. Solving **NP**-complete problems using P systems with active membranes. In I. Antoniou, C.S. Calude, M.J. Dinneen (eds.), *Unconventional Models of Computation*, Springer-Verlag, Berlin, 2000, pp. 289–301.
24. The P systems Web page: `http://ppage.psystems.eu`

A Quantum-Inspired Evolutionary Algorithm Based on P systems for a Class of Combinatorial Optimization

Gexiang Zhang[1,2], Marian Gheorghe[2], Chaozhong Wu[3]

[1] School of Electrical Engineering, Southwest Jiaotong University, Chengdu, Sichuan, 610031, P.R. China
E-mail: zhgxdylan@126.com
[2] Department of Computer Science, The University of Sheffield, Regent Court, Portobello Street, Sheffield S1 4DP, UK
E-mail: M.Gheorghe@dcs.shef.ac.uk
[3] Engineering Research Center of Transportation Safety, Ministry of Education, Wuhan University of Technology, Wuhan, Hubei, 430063, P.R. China
E-mail: chaozhongwu@gmail.com

Summary. This paper introduces an evolutionary algorithm which uses the concepts and principles of the quantum-inspired evolutionary approach and the hierarchical arrangement of the compartments of a P system. The P system framework is also used to formally specify this evolutionary algorithm. Extensive experiments are conducted on a well-known combinatorial optimization problem, the knapsack problem, to test the effectiveness of the approach. These experimental results show that this evolutionary algorithm performs better than quantum-inspired evolutionary algorithms, for certain arrangements of the compartments of the P system structure utilized.

1 Introduction

Evolutionary algorithms (EAs) are practical and robust optimization and search methods inspired by evolutionary processes occurring in natural selection and molecular genetics. The main features of EAs are the representation and evaluation of individuals, population dynamics, evolutionary operators such as selection, crossover and mutation [18, 4]. As compared to conventional optimization methods, EAs are more suitable for solving complex optimization problems as they exhibit an intrinsic parallelism derived from dealing with multiple individuals, show remarkable adaptability and flexibility to application problems, good search capability and robust results [1].

As a new distributed-parallel computing model, membrane computing, also known as P system, has been introduced in [13] as an unconventional,

nature-inspired computational paradigm employing various features specific to the structure and functionality of the living cell. A P system consists of membranes delimiting compartments organized in a hierarchical structure. Objects are scattered across this structure and organized as multisets; specific rules are applied in parallel to these objects in every compartment [13, 15]. The hierarchical structure of membranes, type of rules (transformation, communication etc), intrinsic parallelism, are all very effective from a computational point of view and attractive for modelling various problems.

P systems and EAs are different with respect to the objects and rules used, computational strategies employed, but both are nature-inspired models and applied to solve complex problems, e.g., NP complete problems [9, 14]. P systems represent a suitable formal framework for parallel-distributed computation [2, 20] and EAs are very effective for implementing different algorithms to solve numerous problems [1]. Thus, the possible interaction between P systems and EAs, also mentioned by the list of twenty-six open problems in membrane computing [16], represents a fertile research field. In [9, 10, 11] it is proposed a membrane-based evolutionary algorithm combining a membrane structure where each membrane, but the deepest one, contains one membrane and a local search method. This membrane-based algorithm was also employed to solve the min storage problem [7]. In [5, 6] a hybrid algorithm combining P systems and genetic algorithms was presented to solve single-objective and multi-objective numerical optimization problems. In [20], the similarities between distributed EAs and P systems were analyzed and new variants of distributed EAs are suggested and applied for some continuous optimization problems.

This paper proposes a novel EA, called quantum-inspired evolutionary algorithm based on P systems (QEPS), which uses the concepts and principles of quantum-inspired evolutionary algorithms (QIEAs) within a P system framework. A quantum-inspired bit (Q-bit) representation and quantum-inspired gate (Q-gate) evolutionary rules together with a hierarchical membrane structure and transformation/communication-like rules are employed. To demonstrate the effectiveness and applicability scope of this approach, a large number of experiments are carried out for the knapsack problem, a well-known combinatorial optimization problem. The results obtained show that QEPS with a specific membrane structure performs better than known QIEAs.

The knapsack problem has been frequently analysed by both P systems and evolutionary algorithms communities as test-benches for various approaches. Recognizer P systems with active membranes were constructed to solve one-dimensional [17] and multi-dimensional [12] knapsack problems in linear time. Also, extensively convincing experiments show that QIEA is far better than conventional genetic algorithms [3, 4] in solving the same problem.

This paper is organized as follows. Section 2 introduces briefly QIEA and P systems, and then describes QEPS in detail. Section 3 presents an application example comparing QEPS and QIEAs for knapsack problems and summarizes

the experimental results. Section 4 discusses the parameter setting in QEPS and different membrane structures. Concluding remarks follow in Section 5.

2 QEPS – Basic Concepts

This section starts with brief introductions about QIEA and P systems goes then into presenting a P systems-like framework of QIEA, called QEPS.

2.1 QIEA

Inspired by concepts of quantum computing such as quantum bits and quantum gate, QIEA is a new evolutionary algorithm for a classical computer instead of a quantum algorithm. QIEA was first introduced by Narayanan and Moore in [8] and its practical form was proposed by Han and Kim in [3]. Various variants of QIEA can be categorized into two groups: real observation QIEA for numerical optimization [22] and binary observation QIEA (bQIEA) for combinatorial optimization. The latter, referred in this paper, can be sub-classified into four groups: original bQIEA (bQIEAo) [3], bQIEA with migration operator (bQIEAm) [4], bQIEA with a combination of crossover, mutation and selection operators (bQIEAcms) [19], and bQIEA with a novel update method for Q-gates (bQIEAn) [21]. QIEA approach is characterized by a Q-bit representation for individuals and a Q-gate as a variation operator to obtain better fitted individuals. In QIEA, a Q-bit is defined by a pair of numbers (α, β) as

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad , \tag{1}$$

where $|\alpha|^2$ and $|\beta|^2$ are the probabilities that the observation of a Q-bit will render a '0' or '1' state [4]. Normalization requires that $|\alpha|^2 + |\beta|^2 = 1$. Note that QIEA just needs real numbers for probability amplitudes. Besides '0' and '1' states, a Q-bit can also be in a superposition of the two states. A Q-bit individual is represented as a string of l Q-bits

$$\begin{bmatrix} \alpha_1 | \alpha_2 | \cdots | \alpha_l \\ \beta_1 | \beta_2 | \cdots | \beta_l \end{bmatrix} \quad , \tag{2}$$

where $|\alpha_i|^2 + |\beta_i|^2 = 1$ $(i = 1, 2, \cdots, l)$. A Q-gate in QIEA is defined as a variation operator for updating the Q-bit individuals such as to guarantee that they also satisfy the normalization condition $|\alpha|^2 + |\beta|^2 = 1$ [4]. To date, QIEA principally adopts quantum rotation gate as a Q-gate, as it is shown in Eq. (4).

The basic pseudocode algorithm for bQIEA is shown in Fig. 1. Each step in Fig. 1 is described as follows.

```
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 bQIEA [17]

(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 $\boldsymbol{q}_i^t$ is represented as

$$\boldsymbol{q}_i^t = \begin{bmatrix} \alpha_{i,1}^t | \alpha_{i,2}^t | \cdots | \alpha_{i,l}^t \\ \beta_{i,1}^t | \beta_{i,2}^t | \cdots | \beta_{i,l}^t \end{bmatrix} \quad , \tag{3}$$

where l is the number of Q-bits, i.e., the string length of the Q-bit individual. $\alpha_{i,j}^t = \beta_{i,j}^t = 1/\sqrt{2}$ $(j = 1, 2, \cdots, l)$. This means that all possible states are superposed with the same probability at the beginning.

(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 generated at step t. According to the current probability, either $|\alpha_i^t|^2$ or $|\beta_i^t|^2$ of $\boldsymbol{q}_i^t$ $(i = 1, 2, \cdots, l)$, a binary bit 0 or 1 is generated. Thus, a binary solution $\boldsymbol{x}_j^t$ $(j = 1, 2, \cdots, n)$ consists of l binary bits.

(iii) The fitness value for each binary solution $\boldsymbol{x}_j^t$ $(j = 1, 2, \cdots, n)$ is calculated by using an evaluation function.

(iv) In this step, the Q-bit individuals in $Q(t)$ are updated by applying the current Q-gate. In bQIEA, the quantum rotation gate is used as a Q-gate; this is given by

$$\boldsymbol{G(t)} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad , \tag{4}$$

where θ is the Q-gate rotation angle.

(v) The best solutions among $P(t)$ are selected and stored.

Compared to local search methods [9, 10, 11, 7] and conventional genetic algorithms [20, 5, 6], QIEA has several special characteristics. Firstly, the Q-bit encoding can represent probabilistically a linear superposition of states in the search space, which makes QIEA rather good with respect to population

diversity. Secondly, with a small number of individuals, even with one individual, QIEA can exploit the search space for a global solution within a short span of time. Thirdly, the evolutionary rules are very simple, instead of selection, crossover and mutation operators, QIEA uses only a Q-gate operation, which is related to the current best Q-bit individual in the population.

2.2 P Systems

The membrane structure of a P system, shown in Fig. 2, is a hierarchical arrangement of membranes, embedded in the skin membrane, the one which separates the system from its environment [15]. A membrane without any membrane inside is called an elementary one. Each membrane defines a region. Each region constitutes a different compartment of the membrane structure and contains a multiset of objects and a set of transformation and communication rules.

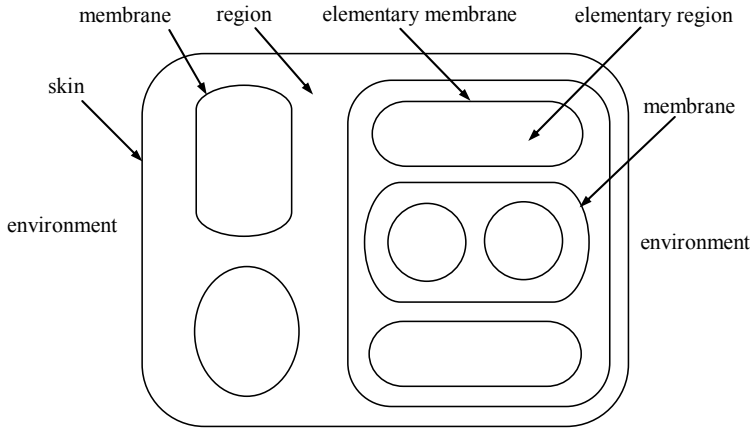


Fig. 2. A membrane structure [2]

The multisets associated to regions form a configuration of a P system. The system will go from one configuration to a new one by applying the rules associated to regions in a non-deterministic and maximally parallel manner, i.e., all the objects that may be transformed or communicated must be processed. The system will halt when no more rules are available to be applied. A computation is a sequence of configurations obtained as it is described above, where the initial configuration consists of the initial multisets associated to regions and the final one is generated when the system halts. The result of a computation is obtained in the region defined by the output membrane.

In what follows a basic P system with an output set of objects and using transformation and communication rules is formally defined. Let us consider a construct [13, 15]

$$\Pi = (V, T, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

where

(i) V is an alphabet; its elements are called objects;
(ii) $T \subseteq V$ (the output alphabet);
(iii) μ is a membrane structure consisting of m membranes, with the membranes and the regions labelled in a one-to-one manner with elements of a given set Λ – usually the set $\{1, \ldots, m\}$; m is called the degree of Π;
(iv) w_i, $1 \leq i \leq m$, are strings which represents multisets over V associated with the regions $1, 2, \cdots, m$ of μ;
(v) R_i, $1 \leq i \leq m$, are sets of rules associated to the regions $1, 2, \cdots, m$ of μ;
(vi) i_0 is a number between 1 and m which specifies the output membrane of Π.

The rules of R_i, $1 \leq i \leq m$, have the form $a \to v$, where $a \in V$ and $v \in (V \times \{here, out, in\})^*$. The multiset v consists of pairs (b, t), $b \in V$ and $t \in \{here, out, in\}$, where *here* means that b will stay in the region where the rules are applied; *out* is used to show that b exits the region and *in* means that b will be communicated to one of the membranes contained in the current region which is chosen in a non-deterministic way.

A P system provides a suitable framework for distributed parallel computation that develops in steps. Indeed, any computation starts by processing the initial multisets, w_i, $1 \leq i \leq m$, and then in each step the rules associated to each region are applied in a non-deterministic and maximally parallel manner. The computation, a multiset of simple objects, is obtained in region i_0. For more details about P systems definition see [15]. We notice that the rules presented above combine both transformation and communication, but these operations may be separated and then the transformation rules are responsible for evolving the objects and the communication rules will transfer objects among regions according to some targets. The initial multisets of simple objects may be replaced by strings or multisets of strings, the multiset rewriting by string rewriting and in the output region obtain a set or multiset of strings.

2.3 QEPS

This section will introduce a P systems-like framework that will help presenting the QEPS algorithm. This framework will use some of the elements of a P system, but others will be used in a rather metaphoric way. A specific membrane structure will be initially introduced, but this will be later on changed. The objects employed will be organized in multisets of special strings built either over the set of Q-bits or $\{0, 1\}$. The rules will be responsible to evolve the system and select the best fit Q-bit individuals.

More precisely the P system-like framework will consist of:

(i) a membrane structure $[_1[_2]_2, [_3]_3, \cdots, [_{m+1}]_{m+1}]_1$ with m regions contained in the skin membrane, denoted by 1;
(ii) a vocabulary that consists of all the Q-bits and the set $\{0, 1\}$;
(iii) a set of terminal symbols, T, consisting of all the Q-bits;
(iv) initial multisets $w_1 = \lambda$,
$w_2 = q_1 q_2 \cdots q_{n_1}, n_1 \leq n$,
$w_3 = q_{n_1+1} q_{n_1+2} \cdots q_{n_2}, n_1 + n_2 \leq n$,
$\cdots \cdots$
$w_{m+1} = q_{n_{(m-1)}+1} q_{n_{(m-1)}+2} \cdots q_{n_m}$, $n_1 + n_2 + \cdots n_m \leq n$, where
q_i, $1 \leq i \leq n$, is a Q-bit individual;
(v) rules which are classified as
(a) evolution rules in each of the compartments 2 to $m + 1$; these are transformation-like rules which update a Q-bit individual according to the current Q-gate (see (iv) of the QIEA presentation);
(b) mapping rules which make binary solutions from Q-bit individuals (see (ii) of the QIEA presentation and algorithm in Fig. 3);
(c) communication rules which send the best fit individual binary representation from each of the m regions into the skin membrane and then the overall best binary representation from the skin back to each region.

In QEPS the initial population of Q-bit individuals is scattered across the membrane structure. The initial population will consist of the multisets $w_2, \cdots, w_{m+1}$. In any step the current generation is assessed compartment by compartment to select the best fit individual (applying rules of type (b)). The best solution is used to adjust the Q-gate which is then employed to produce the next generation by applying evolution rules. Every $g_i (1 \leq i \leq m)$ generations for each compartment, the communication rule is performed once. The process will stop when the best fit solution will remain unchanged for several generations.

```
Begin
   j ← 1
   While (j < l) do
      If random[0,1) < |β_ij^t|^2
      Then x_j ← 1
      Else x_j ← 0
   End
End
```

Fig. 3. Pseudocode algorithm for the mapping rule [4]

3 Application Example

In this section, the QEPS algorithm for the knapsack problem is presented in detail. The knapsack problem is applied to show the effectiveness of QEPS to a combinatorial optimization problem. To make a comparison, three types of QEPS and four types of QIEA are considered. The knapsack problem can be described as selecting from among various items those that are the most profitable, given that the knapsack has a limited capacity [4]. The knapsack problem requires to select a subset of a given set of items so as to maximize a profit function

$$f(x) = \sum_{i=1}^{k} p_i x_i \tag{5}$$

Subject to

$$\sum_{i=1}^{k} r_i x_i \leq C_a \tag{6}$$

where k is the number of items; p_i is the profit of the ith item; r_i is the weight of the ith item; C_a is the capacity of the given knapsack; and x_i is 0 or 1.

Table 1. Lookup table of θ, where $f(.)$ is the fitness, $s(\alpha, \beta)$ is the sign of θ, and b and x are certain bits of the current best solution $\boldsymbol{b}$ and the binary solution $\boldsymbol{x}$, respectively [3]

				$s(\alpha, \beta)$			
x	b	$f(\boldsymbol{x} \geq f(\boldsymbol{b})$	$\Delta\theta$	$\alpha\beta > 0$	$\alpha\beta < 0$	$\alpha = 0$	$\beta = 0$
0	0	False	0	0	0	0	0
0	0	True	0	0	0	0	0
0	1	False	0	0	0	0	0
0	1	True	0.05π	-1	+1	± 1	0
1	0	False	0.01π	-1	+1	± 1	0
1	0	True	0.025π	+1	-1	0	± 1
1	1	False	π	+1	-1	0	± 1
1	1	True	π	+1	-1	0	± 1

3.1 QEPS for the Knapsack Problem

In this paper, we consider three types of QEPS based on various QIEA approaches. Consequently, we have QEPS with different Q-gate update methods, namely used by bQIEAo (QEPSo), bQIEAm (QEPSm), and bQIEAn

(QEPSn). The three variants of QEPS use different methods for deriving the rotation angle θ in $G(t)$, where θ is defined as $\theta = s(\alpha, \beta)\Delta\theta$, where $s(\alpha, \beta)$ and $\Delta\theta$ are the sign and the value of θ, respectively. $s(\alpha, \beta)$ and $\Delta\theta$ can be obtained by using Table 1 for QEPSo, Table 2 for QEPSm and Table 3 for QEPSn, respectively.

Table 2. Lookup table of θ, where $f(.)$ is the fitness, $s(\alpha, \beta)$ is the sign of θ, and b and x are certain bits of the current best solution $\boldsymbol{b}$ and the binary solution $\boldsymbol{x}$, respectively [4]

x	b	$f(\boldsymbol{x} \geq f(\boldsymbol{b})$	$\Delta\theta$	$s(\alpha, \beta)$
0	0	False	0	± 1
0	0	True	0	± 1
0	1	False	0.01π	+1
0	1	True	0	± 1
1	0	False	0.01π	-1
1	0	True	0	± 1
1	1	False	0	± 1
1	1	True	0	± 1

Table 3. Look-up table of θ, where $d_1 = \alpha_1\beta_1$, $\xi_1 = \arctan(\beta_1/\alpha_1)$, α_1, β_1 are the probability amplitudes of the current best solution, and $d_2 = \alpha_2\beta_2$, $\xi_2 = \arctan(\beta_2/\alpha_2)$, α_2, β_2 are the probability amplitudes of the current solution, and $e = 0.5\pi||\alpha_1| - |\alpha_2||$

			$f(\alpha, \beta)$	
$d_1 > 0$	$d_2 > 0$	$\Delta\theta$	$\lvert\xi_1\rvert \geq \lvert\xi_2\rvert$	$\lvert\xi_1\rvert < \lvert\xi_2\rvert$
True	Ture	e	+1	-1
True	False	e	-1	+1
False	True	e	-1	+1
False	False	e	+1	-1

3.2 QIEA for the Knapsack Problem

Any QIEA for the knapsack problem consists of a basic structure (see Fig. 1) and a repair process to match the capacity constraint, illustrated by Eq.

6. The pseudocode algorithm for the repair process is shown in Fig. 4. Four types of QIEA are described and tested for the knapsack problem: bQIEAo [3], bQIEAm [4], bQIEAcms [19] and bQIEAn [21]. The four algorithms can be regarded as special kinds of QEPS with a skin membrane and one elementary membrane. The pseudocode algorithm for bQIEAo is illustrated in Fig. 1; bQIEAo uses Table 1 as a look-up table for determining the rotation angle of the Q-gate. bQIEAm is an improved version of the bQIEAo algorithm that uses inserting migration operation in the bQIEAo algorithm; this uses Table 2 to decide the rotation angle of the Q-gate. A modified algorithm obtained by adding selection, quantum crossover and quantum mutation operators to bQIEAo, and using the same method for determining the rotation angle of the Q-gate as bQIEAo is represented by the bQIEAcms algorithm. Lastly the bQIEAn algorithm appears as a modified version of the bQIEAo algorithm by introducing a modified update method for the Q-gate, whose rotation angle is changed through the look up Table 3. Extensively convincing comparisons between bQIEAm and conventional genetic algorithms show the advantages of bQIEAm. In this paper we will start from these results.

3.3 Experimental Results

In the following experiments, strongly correlated sets of unsorted data are used

r_i=uniformly random [1, 10]
p_i=r_i+5

and the average knapsack capacity

$$C_a = \frac{1}{2}\sum_{i=1}^{k} r_i \quad .$$

Three knapsack problems with 200, 400, and 600 items are considered. Because r_i is a random value, the experimental results in [4] and [3] cannot be referenced directly in this paper.

For the seven algorithms, the population size is set to 20. The parameters g_i, $1 \leq i \leq m$, of QEPSo, QEPSm, and QEPSn are set to be uniformly random integers ranging from 1 to 10. The parameters m and n_i, $1 \leq i \leq m$, are 20 and 1, respectively. The parameter setting for QEPS will be discussed in detail in the next section. To guarantee the seven algorithms have identical stopping criteria, the executions of QEPSo, QEPSm, and QEPSn are stopped when the best profit cannot be further improved in successive 20 iterations, and the executions of bQIEAo, bQIEAm, bQIEAcms and bQIEAn are stopped when the best profit cannot further increase in successive 100 iterations. The performances of the seven algorithms are evaluated by using the criteria: the best solution and the worst solution over 30 runs, the mean best solution over 30 runs, the standard deviation and the elapsed time. Experimental results for the three cases of 200, 400, and 600 items are shown in Table 4. All the experiments are performed on a MATLAB platform and on one machine. If the

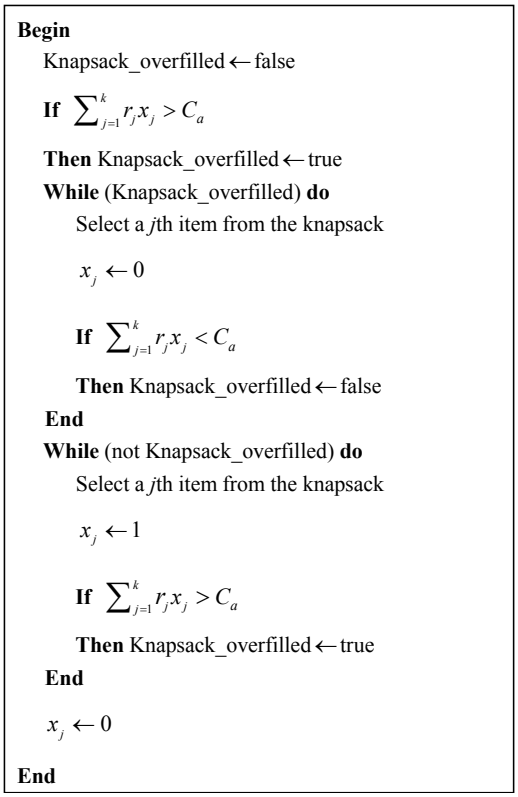


Fig. 4. Pseudocode algorithm for repair process [4]

experiments are conducted in a parallel-distributed way on several machines, the elapsed time can be greatly reduced.

As shown in Table 4, the three types of QEPS perform significantly better than the four types of QIEA in terms of profit results. QEPSm achieves the higher profit values than any other algorithm. Also, QEPSm, QEPSo and QEPSn outperform better than bQIEAm, bQIEAo and bQIEAn, respectively, with respect to profits. QEPSm and QEPSo are superior to bQIEAm and bQIEAo, respectively, with respect to the elapsed time.

The bQIEAm algorithm is the best out of the four types of QIEA and QEPSm is the best out of the seven algorithms. The profits increase at about 1.8% for 200 items, 2.6% for 400 items and 3.0% for 600 items with respect to MBS, which indicates that the increment is bigger as the number of items goes up.

Table 4. Experimental results of the knapsack problem: the number of items 200, 400 and 600, the number of runs 30. BS, MBS, WS, STD and ET represent best solution, mean best solution, worst solution, standard deviation and elapsed time (in seconds), respectively. IT and CRI are abbreviations for items and criteria, respectively

IT	CRI	QEPSm	QEPSo	QEPSn	bQIEAm	bQIEAo	bQIEAn	bQIEAcms
200	BS	**1188.31**	1089.90	1099.96	1178.33	1078.01	1088.27	1078.14
	MBS	**1179.65**	1056.24	1080.21	1159.27	1050.47	1064.90	1056.69
	WS	**1168.33**	1041.38	1057.85	1138.16	1032.56	1046.28	1032.90
	STD	**5.07**	10.82	9.81	9.26	10.91	11.56	12.43
	ET	2093.22	**847.64**	1076.95	2468.33	872.75	936.45	1014.00
400	BS	**2406.43**	2168.68	2215.23	2371.42	2150.47	2162.89	2170.44
	MBS	**2380.60**	2133.95	2177.03	2319.48	2130.82	2135.95	2132.92
	WS	**2361.43**	2101.38	2145.47	2281.34	2109.57	2110.97	2110.63
	STD	**8.91**	14.76	15.54	21.13	12.19	12.84	14.70
	ET	6988.12	**1495.03**	2129.05	7106.36	1574.77	1828.38	1757.16
600	BS	**3557.69**	3183.18	3262.69	3492.68	3172.15	3175.50	3177.64
	MBS	**3524.35**	3145.81	3202.06	3421.55	3143.61	3143.98	3177.64
	WS	**3492.68**	3116.26	3151.57	3362.53	3119.98	3115.38	3115.22
	STD	**14.81**	16.82	21.77	39.44	15.46	14.91	13.83
	ET	13231.31	**2216.11**	3557.66	13597.50	2525.98	2807.94	2372.56

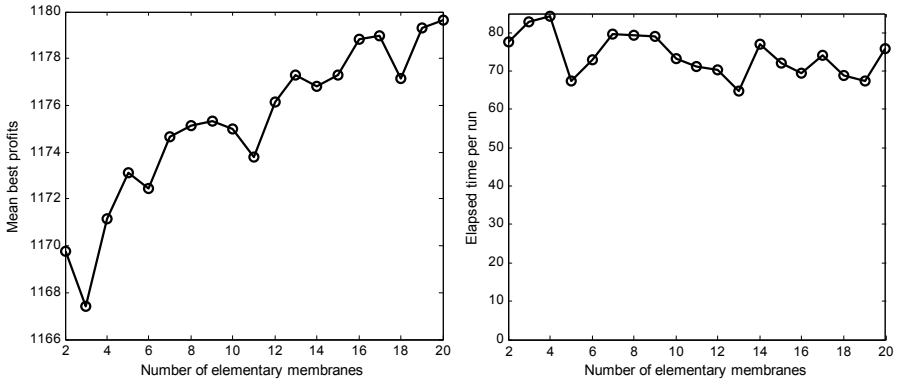
4 Discussion and Analysis

In this section we will discuss different values regarding the number m of elementary membranes, the number n_i, $1 \leq i \leq m$, of objects inside the ith elementary membrane, and the evolutionary generation g_i, $1 \leq i \leq m$, for the ith elementary membrane; finally different membrane structures will be considered and analyzed.

4.1 Parameters m and n_i

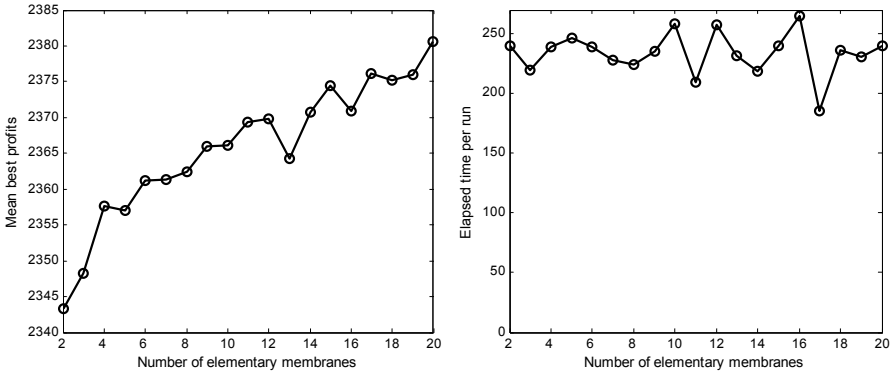
To investigate the effects of the parameters m and n_i, $1 \leq i \leq m$, on the performances of QEPS, experiments of QEPSm on the knapsack problems with 200, 400 and 600 items are tried. The population size is set to 20. For all experiments, when the best profit cannot be further improved in successive 20 iterations, the execution of the algorithm is stopped. The parameter m varies from 2 to 20. The parameter n_i, $1 \leq i \leq m$, is set to a uniformly random integers ranged from 1 to 20 on condition that the sum of $n_1, n_2, \cdots, n_m$ is 20. Also, the parameter g_i, $1 \leq i \leq m$, is set to a uniformly random integer ranged from 1 to 10. The mean best profits over 30 runs and the elapsed time per run for the three cases of 200, 400, and 600 items are shown in Fig. 5, Fig. 6 and

Fig. 7, respectively. From these experimental results, the parameters m could be assigned as 20, i.e., the number of elementary membranes is identical with the number of individuals in the population, which also means that $n_i = 1$, where $i = 1, 2, \cdots, m$. Fig. 5, Fig. 6 and Fig. 7 also illustrate that the elapsed time stays a steady level when the number of membranes increases from 2 to 20.



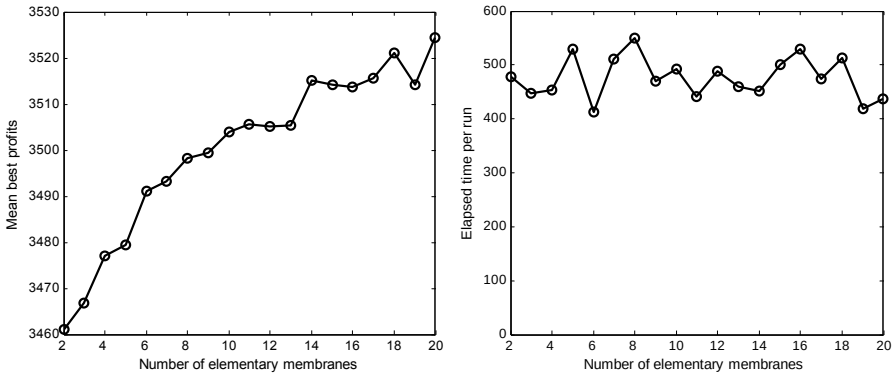
(a) Mean best profits (b) The elapsed time per run (Sec.)

Fig. 5. Experimental results of 200 items with different membranes



(a) Mean best profits (b) The elapsed time per run (Sec.)

Fig. 6. Experimental results of 400 items with different membranes



(a) Mean best profits (b) The elapsed time per run (Sec.)

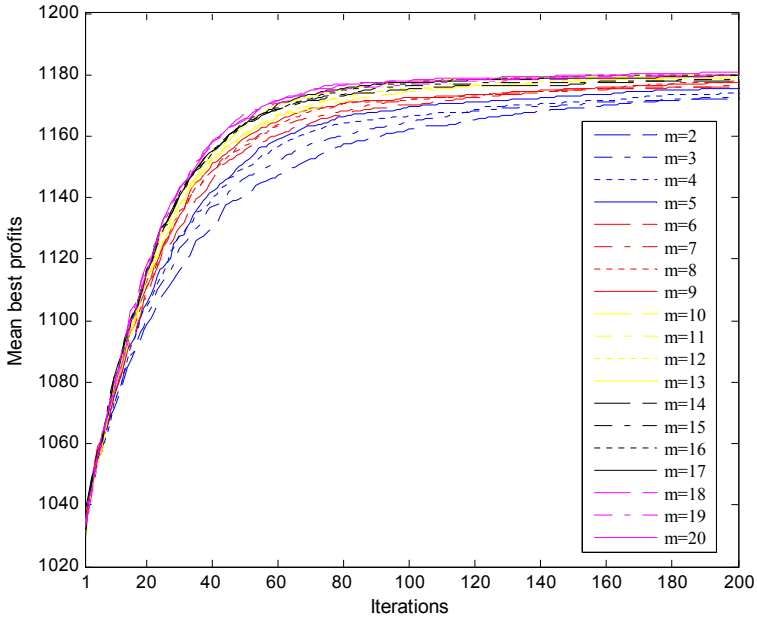
Fig. 7. Experimental results of 600 items with different membranes

Experiments are carried out with the knapsack problems for 200, 400 and 600 items to track the progress of the mean of best profits and the mean of average profits of all individuals. The population size is set to 20. The parameter n_i, $1 \leq i \leq m$, is assigned the value 1. The parameter g_i, $1 \leq i \leq m$, is set to a uniformly random integer ranged from 1 to 10. The execution of every algorithm is stopped when the best profit cannot be further improved in successive 20 iterations. The number m of elementary membranes varies from 2 to 20. Fig. 8 shows the progress of the mean of best profits and the mean of average profits of the population over 30 runs for 200, 400 and 600 items.

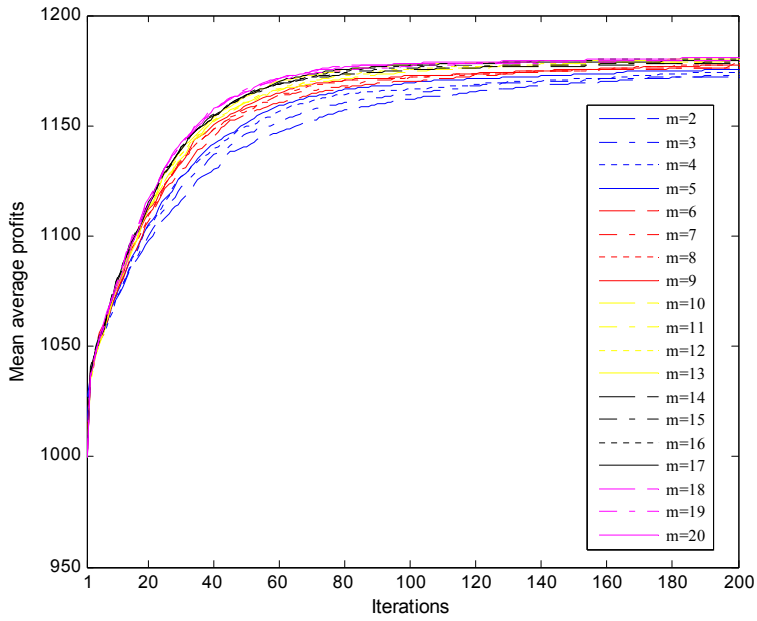
The experimental results in Fig. 8 show that the mean of best profits and the mean of average profits have steady increases with the number of membranes going up. These results indicate that QEPS has better balance between exploration and exploitation as the number of membranes rises from 2 to 20. The more the membranes are, the more directions toward the optimal solution can be explored by the QEPS. Also, the results of the mean of average profits of population show clearly the tendency of convergent rate.

4.2 Parameter g_i

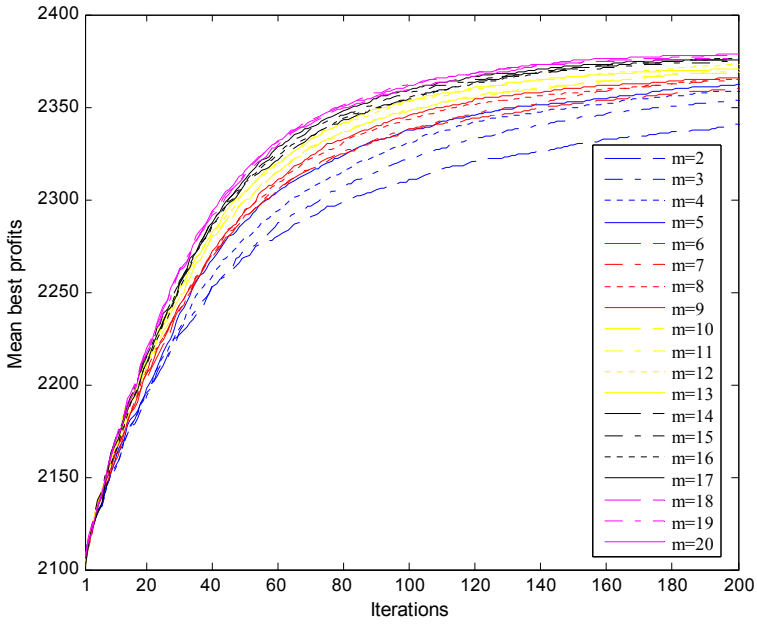
In this subsection, experiments of QEPSm for the knapsack problem with 200, 400 and 600 items are carried out to investigate the effect of the number of evolutionary generations parameter, g_i, $1 \leq i \leq m$, on the performances of this algorithm. Both the population size and the parameter m are set to 20. The parameter n_i, $1 \leq i \leq m$, is then becoming 1. For all experiments, when the best profit cannot be further improved in successive 20 iterations,



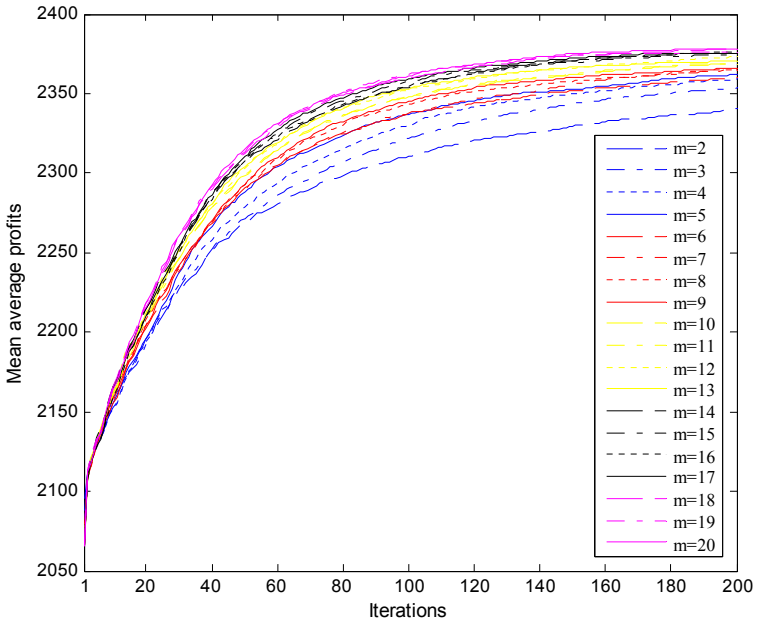
(a) Mean best profits of 200 items



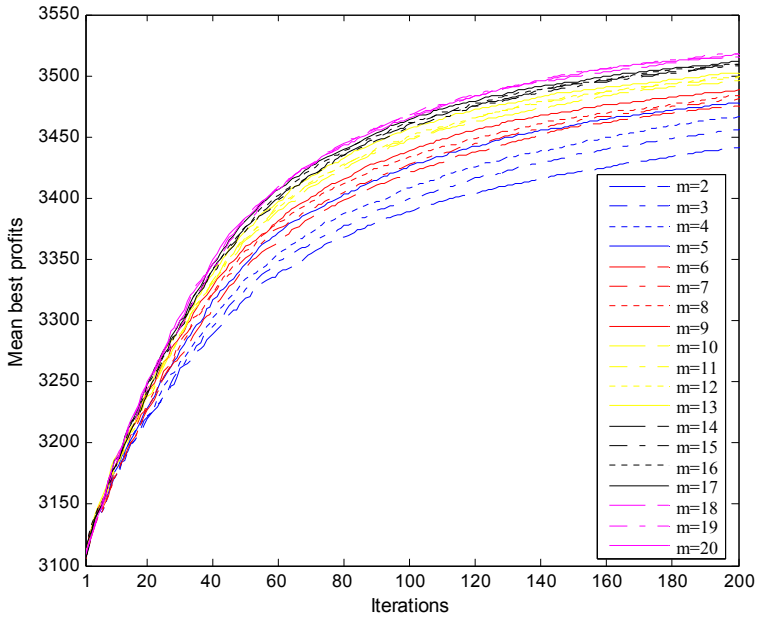
(b) Mean average profits of 200 items



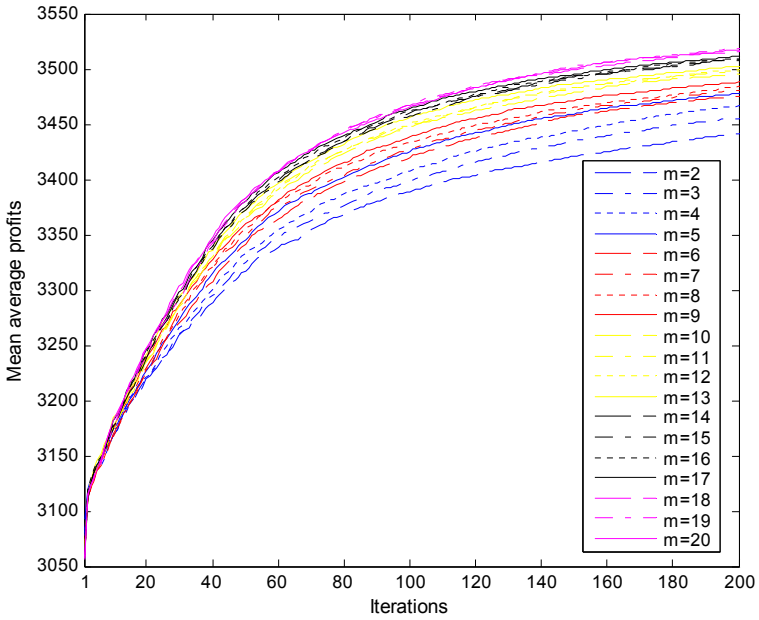
(c) Mean best profits of 400 items



(d) Mean average profits of 400 items



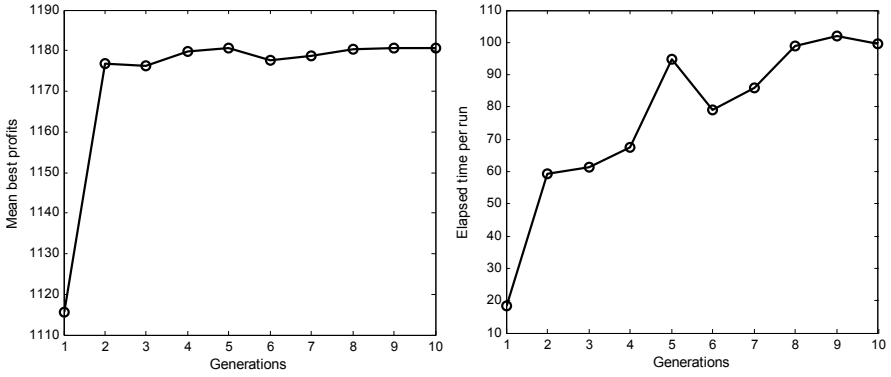
(e) Mean best profits of 600 items



(f) Mean average profits of 600 items

Fig. 8. Progress of solutions with different membranes

the execution of the algorithm is stopped. The parameter g_i, $1 \leq i \leq m$, varies between 1 and 10. In each of the above mentioned experiments, the mean best profits over 30 runs and the elapsed time per run are shown in Fig. 9, Fig. 10 and Fig. 11. The mean best profit values for $m = 20$ in Fig. 5, Fig. 6 and Fig. 7 are very close to the values shown in Fig. 9, Fig. 10 and Fig. 11, for the parameter g_i, $1 \leq i \leq m$, arbitrarily chosen between 2 and 10. These experiments indicate that the values associated to the parameter g_i, $1 \leq i \leq m$, do not influence the mean best profit when they are within the range 2 to 10.



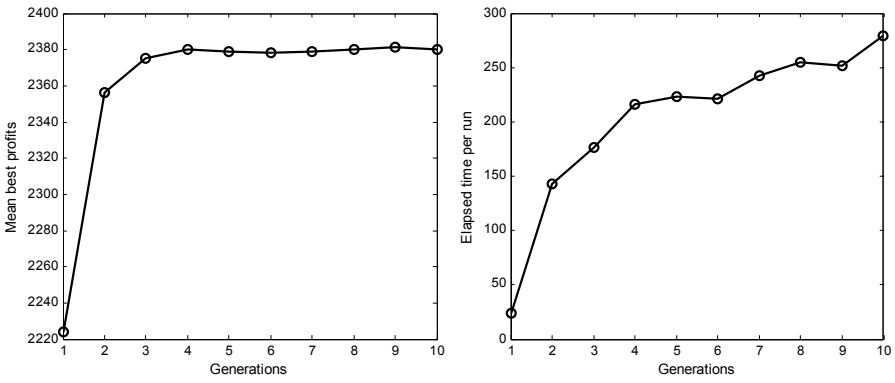
(a) Mean best profits (b) The elapsed time per run (Sec.)

Fig. 9. Mean best profits and elapsed time of 200 items

4.3 Membrane Structures

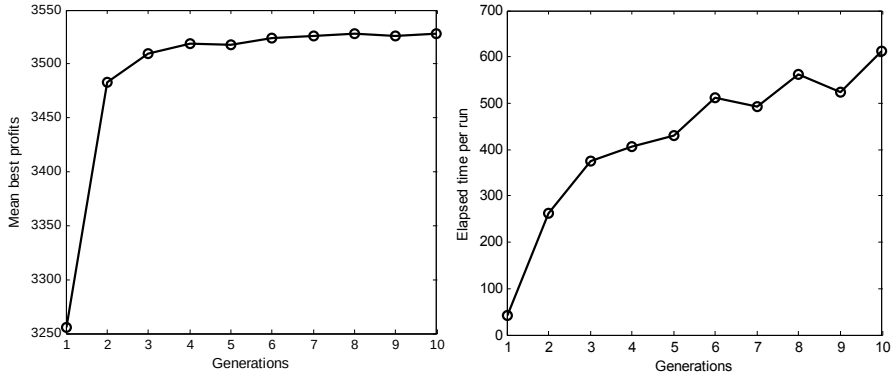
In the above experiments, the membrane structure consisted of a skin membrane and m elementary membranes inside. This membrane structure, called one level membrane structure (OLMS), is illustrated in Fig. 12, and considered in the context of QEPS. In the sequel another membrane structure will be discussed, a nested membrane structure (NMS) shown in Fig. 13. Experiments will be conducted with respect to the knapsack problem to assess the use of NMS. This membrane structure, also called linear topology in [20], was used in [9, 10, 11, 7] in combination with various evolutionary approaches.

In the case of the nested membrane structure we will run experiments under the same conditions we have considered for OLMS. Consequently, the number of individuals contained by each region is arbitrarily chosen between 1 and 20 under the condition that the overall sum equals the population size, which is 20 and experiments are carried out with the knapsack problem for



(a) Mean best profits (b) The elapsed time per run (Sec.)

Fig. 10. Mean best profits and elapsed time of 400 items



(a) Mean best profits (b) The elapsed time per run (Sec.)

Fig. 11. Mean best profits and elapsed time of 600 items

200, 400 and 600 items. All the other parameters and the stopping criterion are the same as those considered for OLMS. An important difference between the two approaches is given by the way the communication rules defined by the P system-like framework are applied. For the OLMS case we remember that the best fit individual binary representation from each of the m regions is sent into the skin membrane and then the overall best fit element is then sent back in each compartment. In the NMS case, the better fit individual will be selected between adjacent neighbours in compartments 2 to $m+1$ and the skin

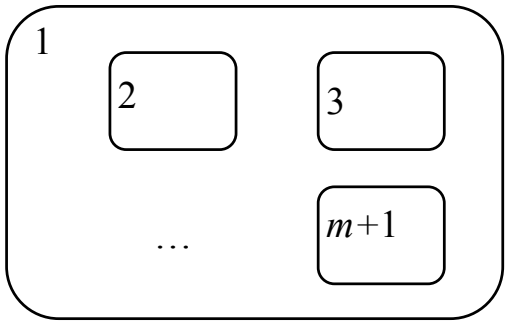


Fig. 12. A one level membrane structure

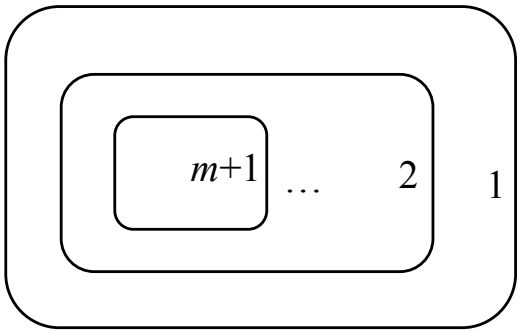


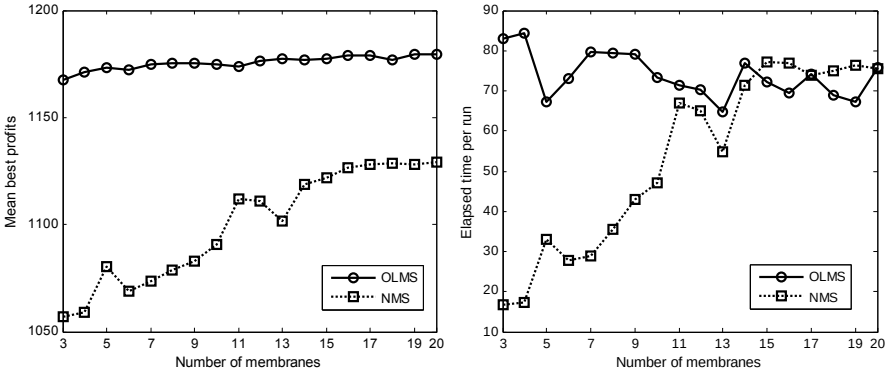
Fig. 13. A nested membrane structure

membrane, denoted by 1, does not play any role in this case. Subsequently, the better fit individual between any two adjacent compartments, i and $i+1$, will be pushed back into the lower compartment, i.e., $i+1$. Through this process the best fit individual will be popped up into the top compartment, i.e., 2. Fig. 14, Fig. 15 and Fig. 16 show the comparative results of using these two membrane structures. All the experimental results are averaged over 30 runs. Table 5 shows the best and worst solutions as well as the mean best solution over 30 runs; the standard deviations and the elapsed time for each of two membrane structures, when the number of membranes varies between 3 and 20 are also shown. Obviously, NMS and OLMS with two membranes show the same behaviour.

The experimental results shown in Fig. 14, Fig. 15 and Fig. 16, prove that, irrespective of the number of membranes used, the profit values obtained in the OLMS case are consistently better than those using NMS, but, on the other hand, the OLMS case requires more computing time than NMS. It is also worth pointing out that, when the number of membranes is above 15, the elapsed time for the QEPS algorithm using either OLMS or NMS is

approximately the same. These results indicate that QEPS with OLMS has better search capabilities than QEPS with NMS.

Table 4 and Table 5 show that QEPS with OLMS is better than QEPS with NMS with respect to the best and worst solutions, the mean best solution, the standard deviations and the elapsed time. These results show that, in the case of the knapsack problem, using the current best solution to control the production of the next generation of individuals (OLMS case) works better than using the best solution between two neighbouring regions (NMS case). Both these approaches produce, in general, better results than most of the bQIEA strategies. More precisely, QEPS with NMS performs better than QEPSo, QEPSn, bQIEAo, bQIEAn and bQIEAcms, but bQIEAm is between QEPSm with OLMS and QEPSm with NMS, in terms of profits. These results show that the choice of the membrane structure for a QEPS algorithm matters and the results might go either way with respect to bQIEA.

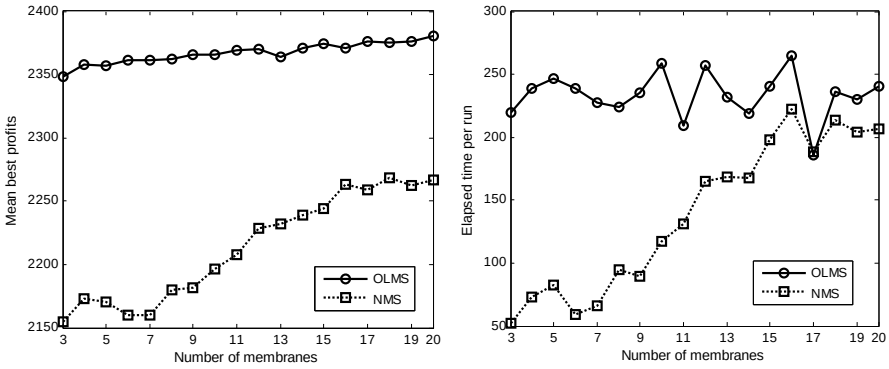


(a) Mean best profits (b) The elapsed time per run (Sec.)

Fig. 14. Comparisons of two structures with 200 items

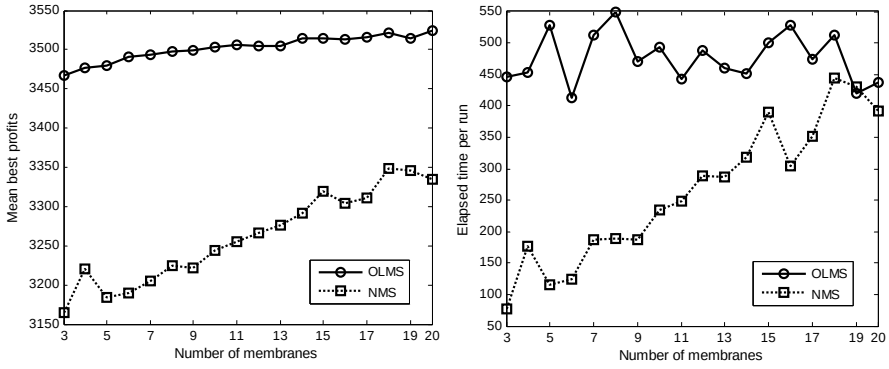
5 Conclusions

This paper proposes the use of quantum-inspired evolutionary algorithms within the parallel-distributed framework of the membrane computing. The algorithms defined in this respect are characterized by a certain membrane structure, string-like objects encoding for Q-bit individuals, and evolution rules usually defined for QIEA approaches. The knapsack problem is considered as an application example to investigate the performances of these



(a) Mean best profits (b) The elapsed time per run (Sec.)

Fig. 15. Comparisons of two structures with 400 items



(a) Mean best profits (b) The elapsed time per run (Sec.)

Fig. 16. Comparisons of two structures with 600 items

evolutionary algorithms. Experimental results show that QEPS algorithms perform in general better than their QIEA counterparts and they can be used to produce effective and efficient solutions to hard combinatorial optimization problems.

Table 5. Experimental results of two structures: the number of items 200, 400 and 600, the number of runs 30. BS, MBS, WS, STD and ET represent best solution, mean best solution, worst solution, standard deviation and elapsed time (in seconds), respectively

Items	Criterions	OLMS	NMS
200	BS	**1188.31**	1153.25
	MBS	**1179.65**	1129.18
	WS	**1168.33**	1052.51
	STD	**5.07**	19.53
	ET	**2093.22**	2261.94
400	BS	**2406.43**	2296.32
	MBS	**2380.60**	2268.46
	WS	**2361.43**	2236.28
	STD	**8.91**	16.66
	ET	6988.12	**6420.31**
600	BS	**3557.69**	3392.53
	MBS	**3524.35**	3348.84
	WS	**3492.68**	3142.04
	STD	**14.81**	45.52
	ET	**13231.31**	13349.63

Acknowledgements

The first author is supported by the National Natural Science Foundation of China (60702026, 60572143). The third author acknowledges the National Basic Research Program of China (2005CB724205, 2006CB705505).

References

1. T. Bäck, U. Hammel, H. P. Schwefel, Evolutionary computation: comments on the history and current state, *IEEE Transactions on Evolutionary Computation*, **1** (1997) 3–17.
2. G. Ciobanu. Distributed Algorithms over Communicating Membrane Systems, *BioSystems*, **70** (2003) 123–133.
3. K. H. Han, J. H. Kim, Genetic quantum algorithm and its application to combinatorial optimization problem, in: *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, 2000, 1354–1360.
4. K. H. Han, J. H. Kim, Quantum-inspired evolutionary algorithm for a class of combinatorial optimization, *IEEE Transactions on Evolutionary Computation*, **6** (2002) 580–593.
5. L. Huang and N. Wang, An optimization algorithm inspired by membrane computing, L. Jiao et al. (Eds.): *ICNC2006*, Part II, Lecture Notes in Computer Science, **4222** (2006) 49–52.

6. L. Huang, X. X. He, N. Wang and Y. Xie, P systems based multi-objective optimization algorithm, *Progress in Natural Science*, **17** (2007) 458–465.
7. A. Leporati and D. Pagani, A membrane algorithm for the min storage problem, H. J. Hoogeboom et al. (Eds.): *WMC2006*, Lecture Notes in Computer Science, **4361** (2006) 443–462.
8. A. Narayanan, M. Moore, Quantum-inspired genetic algorithm, in: *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996, 61–66.
9. T. Y. Nishida, An approximate algorithm for NP-complete optimization problems exploiting P systems, in: *Proceedings of Brainstorming Workshop on Uncertainty in Membrane Computing*, Palma de Majorca, 2004, 185–192.
10. T. Y. Nishida, Membrane algorithms: an approximate algorithm for NP-complete optimization problems exploiting P systems, in: *Application of Membrane Computing* (G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.), Springer, Berlin, 2005, 303–314.
11. T. Y. Nishida, Membrane Algorithms, R. Freund et al. (Eds.): *WMC2005*, Lecture Notes in Computer Science, **3850** (2006) 55–66.
12. L. Q. Pan, C. Martin-Vide, Solving multidimensional 0-1 knapsack problem by P systems with input and active membranes, *Journal of Parallel Distributed Computing*, **65**, (2005) 1578–1584.
13. Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences*, **61** (2000) 108–143.
14. Gh. Păun, P systems with active membranes: attacking NP-complete problems, *Journal of Automata Language Combinatorics*, **6** (2001) 75–90.
15. Gh. Păun and G. Rozenberg, A guide to membrane computing, *Theoretical Computer Science*, **287** (2002) 73–100.
16. Gh. Păun, Further twenty-six open problems in membrane computing, in: *Proceedings of 3rd Brainstorming Meeting on Membrane Computing*, Sevilla, Spain, 2005, 249–262.
17. M. J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, Computationally hard problems addressed through P systems, G. Ciobanu, G. Păun, M. J. Pérez-Jiménez (Eds.), *Applications of Membrane Computing*, Springer, 2006, 315–346.
18. M. Srinivas, L .M. Patnaik, Genetic algorithms: a survey, *Computer*, **27** (1994) 17–26.
19. S. Y. Yang, M. Wang, L. C. Jiao, A novel quantum evolutionary algorithm and its application, in: *Proc. of the 2004 Congress on Evolutionary Computation*, 2004, 820–826.
20. D. Zaharie and Gabriel Ciobanu, Distributed evolutionary algorithms inspired by membranes in solving continuous optimization problems, H. J. Hoogeboom et al. (Eds.): *WMC2006*, Lecture Notes in Computer Science, **4361** (2006) 536–553.
21. G. X. Zhang, L. Z. Hu, W. D. Jin, Resemblance coefficient and a quantum genetic algorithm for feature selection. E. Suzuki, S. Arikawa (Eds.): *DS2004*, Lecture Notes in Artificial Intelligence, **3245** (2004) 155–168.
22. G. X. Zhang, H. N. Rong, Real-observation quantum-inspired evolutionary algorithm for a class of numerical optimization problems. Y. Shi et al. (Eds.): *ICCS2007*, Part IV, Lecture Notes in Computer Science, Springer, **4490** (2007) 989–996.

Author Index