A Linear Solution of Subset Sum Problem by Using Membrane Creation

M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, and F.J. Romero-Campero

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

Abstract. Membrane Computing is a branch of Natural Computing which starts from the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. In this framework, the solution of NP problems is obtained by generating an exponential amount on workspace in polynomial time and using parallelism to check simultaneously all the candidates to solution. We present a solution to the Subset Sum problem for P systems where new membranes are generated from objects.

Keywords: Natural Computing, Membrane Computing, Cellular Complexity Classes, Subset Sum Problem.

1 Introduction

Membrane Computing is a cross-disciplinary field with contributions by computer scientists, biologists, formal linguists and complexity theoreticians, enriching each others with results, open problems and promising new research lines.

This emergent branch of Natural Computing was introduced by Gh. Păun in [11]. Since then it has received important attention from the scientific community. In fact, Membrane Computing has been selected by the Institute for Scientific Information, USA, as a fast *Emerging Research Front* in Computer Science, and [10] was mentioned in [14] as a highly cited paper in October 2003.

This new non-deterministic model of computation starts from the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. The devices of this model are called *P systems*.

Roughly speaking, a P system consists of a cell-like membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules in a synchronous non-deterministic maximally parallel manner[1].

In this paper we present a contribution to this new discipline from the computational eficiency point of view. We introduce a family of P systems, constructed

[1] A layman-oriented introduction can be found in [12] and further bibliography at [15].

in an *uniform way*, that solves the Subset Sum problem. The search of polynomial solution to NP-complete problems is done by trading time by space: An exponential amount of membranes (workspace) is built in polynomial time.

Inspired in living cells, P systems abstract the way of obtaining new membranes. These processes are basically two: *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 are universal from a computational point of view, but technically, they are pretty different. In fact, nowadays there does not exist any theoretical result which proves that these models can simulate each other in polynomial time.

P systems with active membranes have been successfully used to design solutions to well-known **NP**-complete problems, as SAT [9], Subset Sum [6], Knapsack [7], Bin Packing [8] and Partition [2], but as Gh. Păun pointed in [13] *"membrane division was much more carefully investigated than membrane creation as a way to obtain tractable solutions to hard problems"*. Recently, the first results related to the power and design of algorithms to solve NP problems in these model have arisen (see [3, 4]).

The paper is organized as follows: first P systems with membrane creation are recalled in the next section. In section 3 recognizer P systems (devices that capture the intuitive idea underlying the concept of algorithm) are presented. The solution in the framework of *membrane creation* to the Subset Sum problem is given in section 4. Finally, some formal details and conclusions are given in the last sections.

2 P Systems with Membrane Creation

Since Gh. Păun presented the cellular computation with membranes, many different variants have been proposed. If the membrane structure is considered to set a classification among these different variants, two big groups are obtained: P systems where the initial structure does not change along computations and P systems where the tree structure of the membranes vary (or can do it) along computation. The decrease of the number of membranes is made by applying a so-called *dissolution rule* $[a]_e \rightarrow b$ in which the object a inside a membrane with label e produces the dissolution of the rule, a disappears and a new element b and the rest of the multiset in the membrane go to its father (more precisely, they go to the closest non-dissolved ancestor in the membrane hierarchy, since several membranes can dissolve in the same step). Increasing the number of membranes are usually made via division of existing ones or creating new ones from objects[2]

[2] Recently, new operations to change the membrane structure have been explored as *merging membranes* or the operations of *endocytosis*, *exocytosis* or *gemmation*.

Membranes are created in living cells, for instance, in the process of vesicle mediated transport and in order to keep molecules close to each other to facilitate their reactions. Membranes can also be created in a laboratory - see [5]. Here we abstract the operation of creation of new membranes under the influence of existing chemical substances to define P systems with membrane creation. Recall that a *P system with membrane creation* is a construct of the form $\Pi = (O, H, \mu, w_1, \ldots, w_m, R)$ where:

1. $m \geq 1$ is the initial degree of the system;
2. O is the alphabet of *objects*;
3. H is a finite set of *labels* for membranes;
4. μ is a *membrane structure* consisting of m membranes labelled (not necessarily in a one-to-one manner) with elements of H;
5. $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;
6. R is a finite set of *rules*, of the following forms:
   (a) $[a \to v]_h$ where $h \in H$, $a \in O$ and v is a string over O describing a multiset of objects. These are *object evolution rules* associated with membranes and depending only on the label of the membrane.
   (b) $a[\,]_h \to [b]_h$ where $h \in H$, $a, b \in O$. These are *send-in communication rules*. An object is introduced in the membrane possibly modified.
   (c) $[a]_h \to [\,]_h\, b$ where $h \in H$, $a, b \in O$. These are *send-out communication rules*. An object is sent out of the membrane possibly modified.
   (d) $[a]_h \to b$ where $h \in H$, $a, b \in O$. These are *dissolution rules*. In reaction with an object, a membrane is dissolved, while the object specified in the rule can be modified.
   (e) $[a \to [v]_{h_2}]_{h_1}$ where $h_1, h_2 \in H$, $a \in O$ and v is a string over O describing a multiset of objects. These are *creation rules*. In reaction with an object, a new membrane is created. This new membrane is placed inside of the membrane of the object which triggers the rule and has associated an initial multiset and a label.

Rules are applied according to the following principles:

- Rules from (a) to (e) are used as usual in the framework of membrane computing, that is, in a maximal parallel way. 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 evolve by a rule of any form must do it (with the restrictions below indicated).
- If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external one. The skin membrane is never dissolved.
- All the elements which are not involved in any of the operations to be applied remain unchanged.
- The rules associated with the label h are used for all membranes with this label, irrespective of whether or not the membrane is an initial one or it was obtained by creation.

– Several rules can be applied to different objects in the same membrane simultaneously. The exception are the rules of type (d) since a membrane can be dissolved only once.

3 Recognizer P Systems with Membrane Creation

Recognizer P systems were introduced in [7] 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, recognizer 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 to compute the appropriate answer. This is done by codifying each instance as a *multiset*[3] placed in an *input membrane*. The output of the computation (*yes* or *no*) is sent to the environment. In this way, P systems with input and external output are devices which can be seen as black boxes, in which the user provides the data before the computation starts and the P system sends to the environment the output in the last step of the computation. Another important feature of P systems is the non-determinism. The design of a family of recognizer P system has to consider it, because all possibilities in the non-deterministic computations have to output the same answer. This can be summarized in the following definitions (taken from [1]).

Definition 1. *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* $w_1, \ldots, w_p$ *associated with them; (b)* Σ *is an (input) alphabet strictly contained in* Γ*; the initial multisets are over* $\Gamma - \Sigma$*; 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 $(\mu, w_1, \ldots, w_{i_\Pi} \cup m, \ldots w_p)$.

Definition 2. *A* recognizer P system *is a P system with input,* (Π, Σ, i_Π)*, and with external output such that:*

1. *The working alphabet contains two distinguished elements yes, no.*
2. *All its computations halt.*
3. *If* $\mathcal{C}$ *is a computation of* Π*, then either some object yes or some object no (but not both) must have been released into the environment, and only in the last step of the computation. We say that* $\mathcal{C}$ *is an accepting computation (respectively, rejecting computation) if the object yes (respectively, no) appears in the external environment associated to the corresponding halting configuration of* $\mathcal{C}$*.*

[3] Representing the data via multiset is inspired in the multiset of chemical compounds inside living cells.

We denote by $\mathcal{MC}$ the class of recognizer P systems with membrane creation.

In the next section we present a solution to the Subset Sum problem in linear time in the sense of the following definition.

Definition 3. *Let $\mathcal{F}$ be a class of recognizer P systems. 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 $\mathcal{F}$, and we denote this by $X \in \mathbf{PMC}_{\mathcal{F}}$, if the following is true:*

- *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine constructing $\Pi(n)$ from $n \in \mathbb{N}$ in polynomial time.*
- *There exists a pair (cod, s) of polynomial-time computable functions over the set of instances I_X such that:*
  - *for each instance $u \in I_X$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$.*
  - *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 must always give the *same* answer.

It can be proved that $\mathbf{PMC}_{\mathcal{F}}$ is closed under polynomial–time reduction and complement, see [9]. In this paper we will deal with the class $\mathcal{MC}$ of recognizer P systems with membrane creation.

4 Solving Subset Sum in Linear Time with Membrane Creation

In this section we present a family $\prod$ of recognizer P systems that solves the Subset Sum problem in linear time.

The Subset-Sum problem can be settled as follows: *Given a finite set, A, a weight function, $w : A \to N$, and a constant $k \in \mathbb{N}$, determine whether or not there exists a subset $B \subseteq A$ such that $w(B) = k$. If A has n elements with weights $w_1, \ldots, w_n$, one instance of the problem can be encoded as $(n, (w_1, \ldots, w_n), k)$.*

As usual in the framework of P systems, the solution of the problem is based on an algorithm of brute force where an exponential amount of workspace is built in linear time. The algorithm is split in the following stages:

– *Generation stage and weight calculation stage*: for every subset of A, a membrane is created. In each working membrane the weight of the associated subset is calculated.
– *Checking stage*: in each membrane it is checked whether or not the weight of its associated subset is exactly k.
– *Output stage*: when the previous stage has been completed in all membranes, the system sends out the answer (*yes* or *not*) to the environment.

• Working alphabet:

$$\Gamma = \left\{ \begin{array}{l} e_2, \ldots e_n, e_1^+, \ldots, e_n^+, e_1^-, \ldots, e_n^-, c_0, c_1, k_1, k_2, k_3, t_0, \ldots, t_{2n+2k+7}, \\ s, s^+, s^-, s_1, \ldots, s_n, s_2^+, \ldots, s_n^+, s_2^-, \ldots, s_n^-, d_1, \ldots, d_{k+1}, \\ a, p_0, p_1, p_2, p_3, q, q_0, q_1, yes_0, yes_1, yes, no_0, no_1, no_2, no \end{array} \right\}$$

• Initial membrane structure: $\mu = [\,]_0$
• Set of labels: $H = \{0, p, n, t, f, r, s, c\}$
• Initial Multiset: $w_0 = t_0 e_1^+ e_1^-$
• Input: $s_1^{w_1} \ldots s_n^{w_n}$
• The set of evolution rules, $R(\langle n, k \rangle)$, consists of the following rules (recall that λ denotes the empty string):

1. $[e_j^+ \to [e_{j+1}\, k_0]_p]_l$ for $l = 0, p, n \qquad j = 1, \ldots, n-1$
   $[e_j^- \to [e_{j+1}\, k_0]_n]_l$ for $l = 0, p, n \qquad j = 1, \ldots, n-1$
   $[e_j \to e_j^+\, e_j^-]_l$ for $l = p, n \qquad j = 2, \ldots, n$
   $[e_n^+ \to [c_0]_t]_l$ for $l = 0, p, n$
   $[e_n^- \to [c_0]_f]_l$ for $l = 0, p, n$

The goal of these rules is to create one membrane for each possible subset of A. The new membrane with label p, represents the partial subset in which we place the the sum of the object a_1; on the other hand the new membrane with label n, represents the partial subset in which a_1 is not considered.

2. $[k_i \to k_{i+1}]_l$ for $l = p, n \qquad i = 0, 1, 2$
   $[k_3]_l \to \lambda$ for $l = p, n$
   $[t_i \to t_{i+1}]_0$ for $i = 0, \ldots, 2n + 2k + 6$
   $[t_{2n+2k+7} \to [no_0]_c\,]_0$

These rules manage the counters k and t. When a new membrane labelled with p or n is created, an object k_0 is placed inside it. When this counter reaches k_3, the membrane is dissolved in the next stage. The counter t creates a new membrane with the object no_0 after an appropriate number of steps. If this membrane is not dissolved by an element yes_1, the answer *no* will be sent to the environment.

3. $s_j^+\,[\,]_p \to [s_{j-1}]_p$ for $j = 2, \ldots, n$
   $s_j^-\,[\,]_n \to [s_{j-1}]_n$ for $j = 2, \ldots, n$
   $[s_j \to s_j^+ s_j^-]_l$ for $l = 0, p, n \qquad j = 2, \ldots, n$
   $[s \to s^+ s^-]_l$ for $l = 0, p, n$
   $s^+[\,]_l \to [s]_l$ for $l = p, t$
   $s^-[\,]_l \to [s]_l$ for $l = n, f$
   $[s_1 \to s^+]_l$ for $l = 0, p, n$

These rules manage the weights of the elements and are applied simultaneously with the rules of the set **1** which create the new workspace. By using the duplication of symbols and controlling the labels we can place the weight of each different subset into a membrane. For that we need an exponential amount of membranes.

4. $[c_0 \to c_1]_l$ for $l = t, f$
   $[c_1 \to d_1]_l$ for $l = t, f$
   $[s \to [\,]_s]_l$ for $l = t, f$
   $d_i[\,]_s \to [d_i]_s$ for $i = 1, \ldots, k$
   $[d_i]_s \to d_{i+1}$ for $j = 1, \ldots, k$

For each possible subset of A we have one membrane in which the weight of the subset is represented in unary form via the object s. With help of these rules we create as many new membranes labelled by s as objects s there are in the membrane. If k is greater or equal to the number of objects s it the membrane, an object d_{k+1} appears. If not, the computation inside this membrane halts.

5. $[d_{k+1} \to a\, q]_l$ for $l = t, f$
   $[a \to [p_0]_r]_l$ for $l = t, f$
   $q[\,]_s \to [q_0]_s$
   $[p_i \to p_{i+1}]_r$ for $i = 0, 1, 2$
   $[q_0]_s \to q_1$
   $q_1[\,]_r \to [q_1]_r$
   $[q_1]_r \to \lambda$
   $[p_3]_r \to yes_0$

As we saw above, if an object d_{k+1} appears in a membrane, then the weight of the associated subset is less or equal to k. This set of rules check that both amounts are the same. If so, an object yes_0 is produced in the membrane.

6. $[yes_0]_l \to yes_1$ for $l = t, f$
   $yes_1[\,]_c \to [yes_1]_c$
   $[yes_1]_c \to yes$
   $[yes]_0 \to yes[\,]_0$
   $[no_i \to no_{i+1}]_c$ for $l = 0, 1$
   $[no_2]_c \to no$
   $[no]_0 \to no[\,]_0$

There is a counter no_i in the membrane labelled with c. If an object yes_1 is obtained from one (or more) of the exponential amount of membranes which check the subsets of A, this object will stop the counter no_i and send the object yes to the environment. If not, i.e., if in the checking stage none of the membranes output yes_1, the counter no_i will not be stopped and an object no will be sent to the environment.

4.1 An Overview of the Computation

First of all we prepare an input for the instance $u = (n, (w_1, \ldots, w_n), k)$ of the Subset Sum problem. This instance will be processed by $\Pi(s(u))$, being

$s(u) =< n, k >= \frac{(n+k)(n+k+1)}{2}+n$. The input is the multiset $cod(u) = s^{w_1} \dots s^{w_n}$ and we place it in the unique membrane of the initial configuration of $\Pi(s(u))$ and the computation starts.

At the beginning, the counter t_i is started from t_0 and when it reaches the object $t_{2n+2k+7}$ it will create a new membrane with the object no_0 inside. If the process is not stopped, the evolution of that object no_0 will send the answer no to the environment. This process only can be stopped if an object yes_0 is produced in the checking stage, i.e., we will create one membrane for each subset of the initial set A and check if the whole sum in that subset is equal to k. If this happens in one membrane, this membrane produces yes_1. This object dissolves the membrane where the counter no_i is placed and send the object yes to the environment.

In the first stage, from each object e_j we obtain two copies: e_j^+ and e_j^-. These object create new membranes labelled, respectively, with p and n. Since the index j vary along the computation, we obtain an exponential amount of membranes in linear time (it only depends on n).

Simultaneously, the multiset $s^{w_1} \dots s^{w_n}$ which codifies the *input* trigger the appropriate rules to copy the weights in the new membranes. The use of the labels allow us to handle the flow of elements from one membrane to others and after $2n$ steps we have 2^n membranes which the appropriate number of objects s inside. At this point, the checking stage starts.

In this stage, for each membrane we have an object d_1 and as many membranes with label s as the weight of the subset associated to the membrane. The element d_1 goes inside one of this membranes, dissolves it and is changed to d_2. This new object d_2 does the same: it dissolves a membrane with label s and changes to d_3. If the number of membranes labelled with s is greater or equal to k, a new object d_{k+1} is created after $2k$ steps.

Then we have to check if k is greater of equal to number of membranes labelled with s. In other words, if there remains any membrane labelled with s after the apparition of d_{k+1}. This is checked by the rules of the set 5. If there does not remain any membrane; i.e., if the weight of the subset associated to this membrane is equal to k an object yes_0 is produced. If not, the computation in this membrane halts.

When the checking stage finish, for each of the 2^n checking membranes we have one of the following cases: The membrane produces yes_1 in the skin or the computation has halted and nothing has been sent to the skin. At this point the output stage starts.

In the skin we have a membrane labelled by c, produced by the counter t_i when it reached $t_{2n+2k+7}$, with a counter no_i inside. If an element yes_1 has been produced, this means that at least in one of the 2^n checking membrane the weight of the associated subset is equal to k. In this case, the object yes_1 goes inside the membrane with label c, it dissolves it (it stops the counter no_i) and finally the object yes is sent to the environment. If not, the counter no_i is not stopped and in the last step, the object no is sent to the environment.

5 Some Formal Details

In the previous section we have presented a uniform family $\mathbf{\Pi}$ of recognizer P systems which solves the Subset Sum problem. For each n and k a P system $\Pi(\langle n, k\rangle)$ is constructed, where n is the number of elements of the initial set and k is the constant to be reached. First of all, observe that the evolution rules of $\Pi(\langle n, k\rangle)$ are defined in a recursive manner from n and k. The necessary resources to construct the P system are polynomially bounded by n and k, therefore a Turing machine can build the P system in polynomial time with respect to n and k. It can also be proved that the family $\mathbf{\Pi}$ solves the Subset Sum problem in the sense of definition 3 in section 3. Recall that the input of $\Pi(< n, k >)$ is given in a unary representation.

Finally, a formal description of the computation let prove that the P system always halts and sends to the environment the object *yes* or *no* in the last step. The number of steps of the P system is $2n + 2k + 11$ if the output is *yes* and $2n + 2k + 12$ if the output is *no*, therefore there exists a linear bound for the number of steps of the computation.

From the above discussion we deduce that Subset Sum belongs to $\mathbf{PMC}_{\mathcal{MC}}$, therefore since this class is closed under polynomial-time reduction and complement we have $\mathbf{NP} \cup \mathbf{co\text{-}NP} \subseteq \mathbf{PMC}_{\mathcal{MC}}$.

Recently, we have proved that this variant of P systems with membrane creation is **PSPACE** powerful; this result will be published in a forthcoming paper.

6 Conclusions and Future Work

Membrane Computing is a young branch of Natural Computing which has reached an important success in its short life. In these years many results have been presented related to the computational power of membrane devices, but up to now no implementation in electronic or biochemical media has been carried out. This paper deals with the study of *algorithms* to solve well-known problems and in this sense it is placed between the theoretical results, mainly related to computational completeness and computational efficiency, and the real implementation of the devices. Moreover this paper represents a new step in the study of algorithms in the framework of P systems because it exploits membrane creation (a variant poorly studied) to solve **NP**-complete problems. The next steps are, on the one hand, a deeper study of the processes inside living cells in order to improve the models and make them closer to Biology and, on the other hand, to go on with theoretical and computational aspects which allow us to improve the designs and algorithms.

Acknowledgement

This work is supported by Ministerio de Ciencia y Tecnología of Spain, by *Plan Nacional de I+D+I (2000–2003)* (TIC2002-04220-C03-01), cofinanced by

FEDER funds, and by a FPI fellowship (of the third author) from the University of Seville.

References

1. Gutiérrez-Naranjo,M.A; Pérez-Jiménez, M.J. ; Riscos-Núñez, A.: Towards a programming language in cellular computing. *Proceedings of the 11th Workshop on Logic, Language, Information and Computation* (WoLLIC'2004), July 19-22, 2004, 1-16 Campus de Univ. Paris 12, Paris, France.
2. Gutiérrez-Naranjo, M.A.; Pérez-Jiménez, M.J.; Riscos-Núñez, A.: A fast P system for finding a balanced 2-partition, *Soft Computing*, in press.
3. Gutiérrez-Naranjo, M.A.; Pérez-Jiménez, M.J.; Romero-Campero, F.J.: Solving SAT with Membrane Creation. Accepted paper for CiE 2005.
4. Gutiérrez-Naranjo, M.A.; Pérez-Jiménez, M.J.; Romero-Campero, F.J.: A linear solution for QSAT with Membrane Creation. Submitted, 2005.
5. Luisi, P.L.: The Chemical Implementation of Autopoiesis, *Self-Production of Supramolecular Structures* (G.R. Fleishaker et al., eds.), Kluwer, Dordrecht, 1994
6. Pérez-Jiménez, M.J.; Riscos-Núñez, A.: Solving the Subset-Sum problem by active membranes, *New Generation Computing*, in press.
7. Pérez-Jiménez, M.J.; Riscos-Núñez, A.: A linear solution for the Knapsack problem using active membranes, in C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg and A. Salomaa (eds.), *Membrane Computing*, Lecture Notes in Computer Science, **2933**, 2004, 250–268.
8. Pérez-Jiménez, M.J.; Romero-Campero, F.J.: Solving the BIN PACKING problem by recognizer P systems with active membranes, *Proceedings of the Second Brainstorming Week on Membrane Computing*, Gh. Păun, A. Riscos, A. Romero and F. Sancho (eds.), Report RGNC 01/04, University of Seville, 2004, 414–430.
9. Pérez-Jiménez, M.J.; Romero-Jiménez, A.; Sancho-Caparrini, F.: 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 and Gy. Vaszyl (eds.), 2003, 284-294.
10. Păun, A.; Păun, Gh.: The power of communication: P systems with symport/antiport, *New Generation Computing*, **20**, 3 (2002), 295–305
11. Păun, Gh.: Computing with membranes, *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143.
12. Păun, Gh.; Pérez-Jiménez, M.J.: Recent computing models inspired from biology: DNA and membrane computing, *Theoria*, **18**, 46 (2003), 72–84.
13. Păun, Gh.: Further Open Problems in Membrane Computing, *Proceedings of the Second Brainstorming Week on Membrane Computing*, Gh. Păun, A. Riscos, A. Romero and F. Sancho (eds.), Report RGNC 01/04, University of Seville, 2004, 354–365.
14. ISI web page `http://esi-topics.com/erf/october2003.html`
15. P systems web page `http://psystems.disco.unimib.it/`