An Efficient Cellular Solution for the Partition Problem

Miguel Angel 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-mail: {magutier, marper, ariscosn}@us.es

Abstract. Numerical problems are not very frequently addressed in the P systems literature. In this paper we present an effective solution to the Partition problem via a family of deterministic P systems with active membranes using 2-division. The design of this solution is a sequel of several previous works on other problems, mainly the Subset-Sum and the Knapsack problems but also the VALIDITY and SAT. Several improvements are introduced and explained.

1 Introduction

Cellular Computing is a recent branch of Natural Computing initiated in [4]. Its goal is to abstract computing models from the structure and the functioning of living cells.

The present paper is focused in the design of a family of P systems that solves a numerical NP-complete problem, and in the formal verification of this solution. Also the similarities with the solutions presented in [6], [7], [9] and [10] will be highlighted and some conclusions will be extracted from them.

The analysis of the solution presented here will be done from the point of view of the complexity classes. A *complexity class* for a model of computation is a collection of problems that can be solved (or languages that can be decided) by some devices of this model with *similar* computational resources.

In this paper we present a *polynomial complexity class* in cellular computing with membranes inspired in some ideas of Gh. Păun ([3], section 7.1) discussed with some members of the Research Group on Natural Computing from the University of Seville. This class allows us to detect some intrinsic difficulties of the resolution of a problem in the model above mentioned.

The paper is organized as follows: first a formal definition of recognizer P systems is given in the next section; then, in section 3 the polynomial complexity class $\mathbf{PMC}_{\mathcal{AM}}$ is introduced; in sections 4 and 5 a cellular solution for the Partition problem is presented, together with some comments; and finally some final remarks are given in section 6.

2 Preliminaries

Recall that a decision problem, X, is a pair (I_X, θ_X) such that 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.

Definition 1 *A* P system with input *is a tuple* (Π, Σ, i_Π)*, where:*

- Π *is a P system, with working alphabet* Γ*, with* p *membranes labelled by* $1, \dots, p$*, and initial multisets* $\mathcal{M}_1, \dots, \mathcal{M}_p$ *associated with them.*
- Σ *is an (input) alphabet strictly contained in* Γ*.*
- *The initial multisets are over* $\Gamma - \Sigma$*.*
- i_Π *is the label of a distinguished (input) membrane.*

Definition 2 *Let* (Π, Σ, i_Π) *be a P system with input. Let* Γ *be the working alphabet of* Π*,* μ *the membrane structure, and* $\mathcal{M}_1, \dots, \mathcal{M}_p$ *the initial multisets of* Π*. Let* m *be a multiset over* Σ*. The* initial configuration of (Π, Σ, i_Π) with input m *is* (μ_0, M_0)*, where* $\mu_0 = \mu$*,* $M_0(j) = \mathcal{M}_j$*, for each* $j \neq i_\Pi$*, and* $M_0(i_\Pi) = \mathcal{M}_{i_\Pi} \cup m$*.*

Remark. 1 *We denote by* I_Π *the set of all inputs of the P system* Π*. That is,* I_Π *is a collection of multisets over* Σ*.*

The computations of a P system with input $m \in M(\Sigma)$, a multiset over Σ, are defined in a natural way. The only novelty is that the initial configuration must be the initial configuration of the system associated with the input multiset $m \in M(\Sigma)$.

In the case of P systems with input and with external output, the concept of computation is introduced in a similar way but with a small change. In the configurations, we will not work directly with the membrane structure μ but with another structure associated with it including, in some sense, the environment.

Definition 3 *Let* $\mu = (V(\mu), E(\mu))$ *be a membrane structure. The* membrane structure with environment *associated with* μ *is the rooted tree* $Ext(\mu)$ *such that:* (a) *the root of the tree is a new node that we will denote* env*;* (b) *the set of nodes is* $V(\mu) \cup \{env\}$*; and* (c) *the set of edges is* $E(\mu) \cup \{\{env, skin\}\}$*. The node* env *is called* environment *of the structure* μ*.*

Note that we have only included a new node representing the environment which is only connected with the skin, while the original membrane structure remains unchanged. In this way, every configuration of the system informs about the contents of the environment.

Definition 4 *A* language accepting P system *is a P system with input,* (Π, Σ, i_Π)*, and with external output, such that the output alphabet contains only two elements:* Yes *and* No*.*

This definition is stated in a general way, but in this paper P systems within the active membrane model will be used. We refer to [3] (see chapter 7) for a detailed definition of evolution rules, transition steps, and configurations in this model.

Now let us define the *Output* function for our P systems. Given a computation $\mathcal{C} = \{C^i\}_{i<r}$, we will denote by M^j_{env} the content of the environment in the configuration C^j.

Definition 5 *The output of a computation $\mathcal{C} = \{C^i\}_{i<r}$ is:*

$$Output(\mathcal{C}) = \begin{cases} Yes, & if\ \mathcal{C}\ is\ halting,\ Yes \in M_{env}^{r-1}\ and\ No \notin M_{env}^{r-1}, \\ No, & if\ \mathcal{C}\ is\ halting,\ No \in M_{env}^{r-1}\ and\ Yes \notin M_{env}^{r-1}, \\ not\ defined, & otherwise. \end{cases}$$

If $\mathcal{C}$ satisfies any of the two first conditions, then we say that it is a *successful computation.*

Definition 6 *A language accepting P system is said to be* valid *if for every halting computation, and only for them, one symbol Yes or one symbol No (but not both) is sent out (in the last step of the computation).*

Definition 7 *We say that $\mathcal{C}$ is an* accepting *computation (respectively,* rejecting *computation) if the object Yes (respectively, No) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$; that is, if $Yes = Output(\mathcal{C})$ (respectively, $No = Output(\mathcal{C})$).*

Definition 8 *A* language recognizer P system *is a valid language accepting P system such that all its computations halt.*

This recognizer systems are specially suitable when trying to solve decision problems.

3 The Complexity Class $\mathbf{PMC}_{\mathcal{AM}}$

Roughly speaking, a computational complexity study of a solution for a problem is an estimation of the resources (time, space, ...) that are required through all the processes that take place in the way from the bare instance of the problem up to the final answer.

The first results about "solvability" of **NP**–complete problems in polynomial time (even linear) by cellular computing systems with membranes were obtained using variants of P systems that lack an input membrane. Thus, the constructive proofs of such results need to design one system for each instance of the problem.

If we wanted to perform such a solution of some decision problem in a laboratory, we will find a drawback on this approach: a system constructed to solve a concrete instance is useless when trying to solve another instance. This handicap can be easily overtaken if we consider a P system with input. Then, the same system could solve different instances of the problem, provided that the corresponding input multisets are introduced in the input membrane.

Instead of looking for a single system that solves a problem, we prefer designing a family of P systems such that each element decides all the instances of "equivalent size", in certain sense.

Before introducing the definition of the complexity class we deal with, we need some preliminary notions.

Let us denote by $\mathcal{AM}$ the class of language recognizer P systems with active membranes using 2-division (see [3], section 7.2).

Definition 9 *Let L be a language and $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbf{N}}$ a family of P systems with active membranes using 2-division. A* polynomial encoding *of L in $\mathbf{\Pi}$ is a pair (cod, s) of polynomial-time computable functions, $cod : L \to \bigcup_{t\in\mathbf{N}} I_{\Pi(t)}$, and $s : L \to \mathbf{N}$ such that for every $u \in L$ we have $cod(u) \in I_{\Pi(s(u))}$.*

That is, for each word u of the language L, we have a multiset $cod(u)$ and a number $s(u)$ associated with it such that $cod(u)$ is a multiset of input for the P system $\Pi(s(u))$.

Lemma 3.1 *Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be languages. Let $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbf{N}}$ a family of P systems with active membranes using 2-division. If $r : \Sigma_1^* \to \Sigma_2^*$ is a polynomial-time reduction from L_1 to L_2, and (cod, s) is a polynomial encoding of L_2 in $\mathbf{\Pi}$, then $(g \circ r, h \circ r)$ is a polynomial encoding of L_1 in $\mathbf{\Pi}$.*

Proof. This result follows directly from the previous definition. For a detailed proof, we refer the reader to [9]. □

Considering all the definitions already presented, we are now ready to give the definition of the complexity class $\mathbf{PMC}_{\mathcal{AM}}$, which is based on the one given in [9].

Definition 10 *We will say that a decision problem, $X = (I_X, \theta_X)$, is* solvable *in polynomial time by a family of language recognizer P systems with active membranes using 2-division, and we denote this by $X \in \mathbf{PMC}_{\mathcal{AM}}$, if there exists a family of P systems, $\mathbf{\Pi} = \big(\Pi(t)\big)_{t\in\mathbf{N}}$, with the following properties:*

1. *The family $\mathbf{\Pi}$ is consistent, with regard to the class $\mathcal{AM}$; that is, $\forall t \in \mathbf{N}$ $(\Pi(t) \in \mathcal{AM})$.*
2. *The family $\mathbf{\Pi}$ is polynomially uniform, by Turing machines; that is, there exists a deterministic Turing machine constructing $\Pi(t)$ from t in polynomial time.*
3. *There exist two functions, $cod : I_X \to \bigcup_{t\in\mathbf{N}} I_{\Pi(t)}$ and $s : I_X \to \mathbf{N}^+$, computable in polynomial time, such that:*
   - *For every $u \in I_X$, $cod(u) \in I_{\Pi(s(u))}$.*
   - *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 the system $\Pi(s(u))$ 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 $u \in I_X$ it is verified that if there exists an accepting computation of the system $\Pi(s(u))$ 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 $u \in I_X$ it is verified that if $\theta_X(u) = 1$, then every computation of the system $\Pi(s(u))$ with input $cod(u)$ is an accepting one.*

Remark. 2 *Note that, as a consequence of the above definition, the complexity class $\mathbf{PMC}_{\mathcal{AM}}$ is closed under complement (because we use recognizer P systems).*

Proposition 1 *Let X and Y be decision problems such that X is reducible to Y in polynomial time. If $Y \in \mathbf{PMC}_{\mathcal{AM}}$, then $X \in \mathbf{PMC}_{\mathcal{AM}}$.*

That is, the complexity class $\mathbf{PMC}_{\mathcal{AM}}$ is stable under polynomial-time reduction. The proof of this result can also be found in [9].

4 Solving the Partition Problem in Linear Time

The *Partition* problem can be stated as follows:

> *Given a set A of n elements, where each element has a "weight" $w_i \in \mathbf{N}$, decide whether or not there exists a partition of A into two subsets such that they have the same weight.*

We will represent the instances of the problem using tuples of the kind $(n, (w_1, \ldots, w_n))$, where n is the size of the set A and $(w_1, \ldots, w_n)$ is the list of weights of the elements from A. We can define in a natural way an additive function w that corresponds to the data in the instance.

We will address the resolution of the problem via a brute force algorithm, in the framework of language recognizer P systems with active membranes using 2-division, without cooperation nor priority among rules. Our strategy will consist in:

- *Generation stage*: membrane division is used until a specific membrane for each pair (B, B^c) is obtained, where B is a subset of A that contains the element a_1 (this condition is stated to avoid considering twice the same pair).
- *Calculation stage*: in each membrane the weight of the associated subset and of its complementary are calculated.
- *Checking stage*: in each membrane it is checked whether or not these two weights coincide.
- *Output stage*: the answer is delivered according to the results of the checkings.

The family presented here is

$$\mathbf{\Pi} = \{(\Pi(n), \Sigma(n), i(n)) : n \in \mathbf{N}\}.$$

For each element of the family, the input alphabet is $\Sigma(n) = \{x_1, \ldots, x_n\}$, the input membrane is $i(n) = e$, and the P system $\Pi(n) = (\Gamma(n), \{e, r, s\}, \mu, \mathcal{M}_e, \mathcal{M}_r, \mathcal{M}_s, R)$ is defined as follows:

• Working alphabet:

$$\Gamma(n) = \{a_0, a, b_0, b, c, d_0, d_1, d_2, e_1, \ldots, e_n, g, \bar{g}, \hat{g}, h_0, h_1, i_1, i_2, i_4, i_5, p, \bar{p}, q, x_1, \ldots, x_n, Yes, No, No_0, z_1, \ldots, z_{2n+1}, \#\}.$$

• Membrane structure: $\mu = [\ [\ ]_e\ [\ ]_r\ ]_s$.

• Initial multisets: $\mathcal{M}_e = e_1 g$; $\mathcal{M}_r = b_0 h_0$ and $\mathcal{M}_s = z_1$.

• The set of evolution rules, R, consists of the following rules:

(a) $[e_i]_e^0 \rightarrow [q]_e^- [e_i]_e^+$, for $i = 1, \ldots, n$,
$[e_i]_e^+ \rightarrow [e_{i+1}]_e^0 [e_{i+1}]_e^+$, for $i = 1, \ldots, n-1$.

The goal of these rules is to generate one membrane for each subset of A that contains a_1. In each step (according to the index of e_i), we consider an element of A and either we add it to the subset associated with the membrane, B, or we put it in the complementary subset, B^c.

(b) $[x_1 \to a_0]_e^0$; $\quad [x_1 \to \bar{p}]_e^+$,
$[x_i \to x_{i-1}]_e^+$, for $i = 2, \dots, n$,
$[x_i \to \bar{p}]_e^-$, for $i = 2, \dots, n$.

In the beginning, the multiplicities of the objects x_j (with $1 \le j \le n$) encode the weights of the corresponding elements of A. They are not present in the definition of the system, but they are inserted as input in the membrane labelled by e before starting the computation: for each $a_j \in A$, w_j copies of x_j have to be added to the input membrane. During the computation, at the same time as elements are added to the subset associated with a membrane, objects a_0 and $\bar{p}$ are generated to store the weight of such subset and of its complementary.

(b2) $[e_n]_e^+ \to \#$,
$[a_0 \to \#]_s^0$; $\quad [\bar{p} \to \#]_s^0$, $\quad [g \to \#]_s^0$.

This rules perform a "cleaning" task dissolving the membranes that are not meaningful and erasing the contents that these dissolutions spill in the skin membrane. This is not essential in the design, but it is helpful.

(c) $[q \to i_1]_e^-$, $\quad [\bar{p} \to p]_e^-$, $\quad [a_0 \to a]_e^-$,
$[g]_e^- \to [\ ]_e^- g$.

When a membrane gets negatively charged, the two first stages (i.e. generation and calculation stages) end, and then some transition rules are applied. Objects a_0 and $\bar{p}$, whose multiplicities encode the weights of the associated subset and of its complementary, are renamed for the next stage, when their multiplicities are compared. Also an object g is sent out and the total weight of all the elements that have not been considered in the generation stage is added to the complementary.

(d) $[a]_e^- \to [\ ]_e^0 \#$, $\quad [p]_e^0 \to [\ ]_e^- \#$.

These rules implement the comparison above mentioned (that is, they check whether $w(B) = w(B^c)$ holds or not). They work as a loop that erases objects a and p one by one alternatively, changing the charge of the membrane in each step.

(e) $[i_1 \to i_2]_e^-$, $\quad [i_2 \to i_1]_e^0$.

A marker that controls the previous loop is described here. The index of i_j and the electric charge of the membrane give enough information to point out if the number of objects a is greater than (less than or equal to) the number of objects p.

(f) $[i_1]_e^0 \to [\ ]_e^+ No$.

If a subset $B \subseteq A$ verifies that $w(B) > w(B^c)$, then inside the relevant membrane that encodes it (this will be defined later) there will be less objects p than a. This forces the loop described in (e) to halt: the moment will come when there are no objects p left, and then the rule $[i_2 \to i_1]_e^-$ will be applied but it will not be possible to apply the rule $[p]_e^0 \to [\ ]_e^- \#$ at the same time. Thus, an object i_1 will be present in the membrane and the latter will be neutrally charged, so the rule (f) will be applied ending the checking stage with negative result.

(g) $[i_2 \to i_4 c]_e^-$,
$[c]_e^- \to [\ ]_e^0 \#$, $\quad [i_4 \to i_5]_e^0$,
$[i_5]_e^0 \to [\ ]_e^+ Yes$, $\quad [i_5]_e^- \to [\ ]_e^+ No$.

If, on the contrary, $w(B) \leq w(B^c)$ holds, then the objects a will be exhausted before the objects p. It is important to distinguish between the cases where the multiplicity of p is strictly greater than the multiplicity of a and the cases where both multiplicities coincide. This is why object c gives again neutral charge to the membrane and then object i_5 checks if a rule $[p]_e^0 \to [_e]_e^- \#$ is applied or not.

(h) $[p \to \#]_e^+, \quad [a \to \#]_e^+ \#$.

If after the checking loop of rules in (d) has finished there are still some objects p or a in the membrane, they can be erased (again, just for "cleaning" purposes).

(i) $[\bar{g} \to \hat{g}]_s^+$,
$\hat{g}[\ ]_e^+ \to [\hat{g}]_e^0$.

Before the answer is sent out, the system has to make sure that all the relevant membranes have finished their checking stages. This is done using the objects g that are present in the skin and the auxiliary membrane labelled by r (see the next set of rules). There must be 2^{n-1} copies of g, because each relevant membrane sends one, and there is one relevant membrane for each $B \subseteq A$ such that $a_1 \in B$, that is 2^{n-1} in all.

(j) $d_0[\ ]_r^- \to [d_0]_r^0$,
$[h_0 \to h_1]_r^0, \quad [h_1 \to h_0]_r^+$,
$[b_0]_r^0 \to [\ ]_r^+ b, \quad \hat{g}[\ ]_r^+ \to [\hat{g}]_r^0$,
$b[\ ]_r^0 \to [b_0]_r^+, \quad [\hat{g}]_r^+ \to [\ ]_r^0 \hat{g}$,
$[h_0]_r^1 \to [\ ]_r^+ d_2, \quad [d_2]_s^+ \to [\ ]_s^- d_2$.

The membrane labelled by r is present in the initial configuration, but remains inactive until an object d_0 "wakes it up". The purpose of the membrane is to perform a loop where the objects $\hat{g}$ are involved, and thus we can detect if there are no objects $\hat{g}$ present in the skin region. This fact will mean that all the relevant membranes have finished their checking stage, and that the system is ready to send out the answer (Yes or No).

(k) $[No \to No_0]_s^-$,
$[Yes]_s^- \to [\ ]_s^0 Yes$,
$[No_0]_s^- \to [\ ]_s^0 No$.

Finally, the output process is activated. The skin membrane needs to be negatively charged before the answer is sent out. Object d_2 takes care of this (see the previous set of rules) and then, if the answer is affirmative, an object Yes will be sent out recovering the neutral charge for the skin. Note that the answer Yes has some priority over the negative answer, in the sense that we first check if there is any object Yes and then, if it is not the case, the answer No will be sent out.

5 Improving the Design

If one studies how the generation stage works, one can notice that the number of spare membranes that are generated and immediately dissolved (the ones with positive charge and containing the object e_n) is actually 2^{n-1}, the same amount that of relevant membranes. In the formal model we do not worry about this, because this space is created during the computation, and thus it is not needed *a priori*. But if we try to run a simulation of the design in a computer, then the space complexity becomes much more important.

Even if we use the trick of dissolving the membranes immediately after being generated, the resources used are too much.

A possible solution is to avoid the generation of such useless membranes. This can be done for example using a division strategy that follows a complete binary tree structure. We are not using this strategy because we have the intention to get some of the relevant membranes before the generation stage ends, instead of getting all the membranes together after a linear number of steps. This is motivated because we are looking for better efficiency in the best or average case.

Here is a proposal:

Generation stage

$[e_i]_e^0 \to [q]_e^0[e_i]_e^+$, for $i = 1, \ldots, n-1$,

$[e_n \to q]_e^0$,

$[e_i]_e^+ \to [e_{i+1}]_e^0[e_{i+1}]_e^+$, for $i = 1, \ldots, n-2$,

$[e_{n-1}]_e^+ \to [\ ]_e^0\#$,

$[e'_i \to e'_{i+1}]_e^+$, $\quad [e''_i \to e'_{i+1}]_e^+$, for $i = 1, \ldots, n-2$,

$[e'_i \to e''_i]_e^0$, $[e''_i \to \lambda]_e^0$, for $i = 1, \ldots, n-1$,

$[e'_{n-1} \to e_n]_e^+$, $\quad [e''_{n-1} \to e_n]_e^+$.

In this new approach, we do not produce any useless membrane, so the dissolving rules are no longer needed. Furthermore, only two electrical charges are used. Although the sequence of electrical charges of a membrane is still meaningful, the end of the stage is not marked anymore now by getting negative charge, but by having neutral charge for two consecutive evolution steps (see [1] for an example of using active membranes with only two charges). This condition is controlled by the "witness-objects" e'_i and e''_i, that show whether in the previous step the charge was 0 (e''_i) or + (e'_i).

If we want to use these rules instead of the rules in (a), then the checking stage needs also to be adapted, and this could be done as follows:

Weight calculation stage

$[x_1 \to a_0]_e^0$, $\quad [x_1 \to \bar{p}]_e^+$,

$[x'_1 \to a_0]_e^0$, $\quad [x'_1 \to \bar{p}]_e^+$,

$[x_i \to x_{i-1}]_e^+$, for $i = 2, \ldots, n$,

$[x'_i \to x_{i-1}]_e^+$, for $i = 2, \ldots, n$,

$[x_i \to x'_i]_e^0$, for $i = 2, \ldots, n$,

$[x'_i \to \bar{p}]_e^0$, for $i = 2, \ldots, n$.

This stage is almost the same as it was in the former designs, but again it is necessary to introduce "witness-objects" to detect when the generation stage finishes, because in this moment the calculation stage must also conclude.

6 Final Remarks

The designs proposed here try to be as general as possible, and at the same time we try to be *uniform*, in the sense that the design of a family of P systems that solves a problem is not made thinking on one P system for each instance of the problem. Instead, each P system of the family can deal with a set of instances of the *same size* (in this case, with the same value of n, independently of the values of the weight function), receiving at the beginning of the computation an input that encodes the concrete instance. It is also important that the number of steps of the computations is polynomial (preferably linear) with respect to the input given; in the case of Partition the number of steps is of a linear order.

Several numerical problems have already been solved with similar techniques: the Subset-Sum problem ([6]), the Knapsack problem ([7]) and the Partition problem (in this paper), among others. This fact gives rise to the following question: is it possible to formalize a procedure of "reusing rules"? This question is addressed in [11], in this volume.

Some first attempts in this direction have already been made, in the framework of the P systems simulator in Prolog (see [2]). Several files have been created, containing the instructions to generate the evolution rules that deal with the different problems (following the schemes given in the corresponding designs), and now we are working to put these files together and reuse somehow the information.

Another research topic related with these ideas is trying to formalize what means polynomial reduction performed by P systems. It will be nice to have a definition of a complexity class that only depends on P systems parameters, without including a polynomial-time precomputing process (which is somehow unnatural).

Finally, the omnipresent burden of membrane computing: still not implemented in labs (neither in other physical means). Chemical processes in nature are often cyclic, or reversible. Maybe instead of trying to bridge the definition of the P systems model with biology, a more reachable goal is to bridge the "subroutines language" of membrane computing with cellular biochemistry.

Acknowledgements. The support of this research through the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds, is gratefully acknowledged.

References

[1] Alhazov, A., Freund, R., Păun, Gh.: P systems with active membranes and two polarizations, in the present volume.

[2] Cordón-Franco, A., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J. Sancho-Caparrini, F.: A Prolog simulator for deterministic P systems with active membranes, *New Generation Computing*, to appear.

[3] Păun, Gh.: *Membrane Computing. An introduction*, Springer-Verlag, Berlin, 2002.

[4] Păun, Gh.: Computing with membranes, *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143.

[5] Păun, Gh., Rozenberg, G.: A guide to membrane computing, *Theoretical Computer Sciences*, **287** (2002), 73–100.

[6] Pérez-Jiménez, M.J., Riscos-Núñez, A.: Solving the Subset-Sum problem by active membranes, *New Generation Computing*, to appear.

[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*, vol. 2933, 2004, 250–268.

[8] Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: *Teoría de la Complejidad en modelos de computation celular con membranes*, Editorial Kronos, Sevilla, 2002.

[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*, Budapest, Hungary.

[10] Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: Solving VALIDITY problem by active membranes with input, *Proceedings of the Brainstorming Week on Membrane Computing*, M. Cavaliere, C. Martín-Vide, Gh. Păun (eds.), Report GRLMC 26/03, 2003, 279–290.

[11] Riscos-Núñez, A., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Towards a programming language in cellular computing, in the present volume.