# Characterizing Tractability with Membrane Creation

Miguel A. Gutiérrez–Naranjo, Mario J. Pérez–Jiménez, Agustín Riscos–Núñez, Francisco J. Romero–Campero
Research Group on Natural Computing
Dpt. Computer Science and Artificial Intelligence, University of Sevilla
Avda. Reina Mercedes s/n, 41012 - Sevilla, SPAIN
Email: {magutier, marper, ariscosn, fran}@us.es

***Abstract*— This paper analyzes the role that membrane dissolution rules play in order to characterize (in the framework of recognizer P systems with membrane creation) the tractability of decision problems –that is, the efficient solvability of problems by deterministic Turing machines. In this context, the use or not of these rules provides an interesting borderline between tractability and (presumable) intractability.**

## I. Introduction

Membrane Computing is a cross-disciplinary field of Natural Computing with contributions by computer scientists, biologists and formal linguists that was introduced by Gh. Păun in [6]. Since then it has received important attention from the scientific community. In fact, Membrane Computing has been chosen by the Institute for Scientific Information as a fast *Emerging Research Front* in Computer Science in October 2003 [14].

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 living cells, new membranes are produced basically through two processes, *mitosis* (membrane division) and *autopoiesis* (membrane creation)[2]. These two processes have inspired two variants of P systems, namely *P systems with active membranes* and *P systems with membrane creation*.

P systems with active membranes have been successfully used to design solutions to well-known **NP**-complete problems (e.g. [11], [12], [13]). Recently, the first (*uniform*) results related to the computational efficiency using membrane creation have arisen (see [2], [3]).

The paper is organized as follows. First, we summarize basic notions on computational complexity in P systems. Section III recalls P systems with membrane creation. The concept of dependency graph is defined in Section IV, providing a characterization of standard class **P** in terms of recognizer P systems with membrane creation without using dissolution rules. A linear solution to QSAT by P systems with membrane creation is presented in Section V, showing a surprising role of dissolution rules: using them we go beyond tractability. Finally, some conclusions are given in Section VI.

[1] A detailed description can be found in [7] and further bibliography at [15].

[2] 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 [4].

## II. Recognizer P Systems

In the structure and functioning of a cell, biological *membranes* play an essential role. The cell is separated from its environment by means of a *skin membrane*, and it is internally compartmentalized by means of *internal membranes*. Within the cell there are chemical substances that can participate in various reactions, depending on the compartment where they reside.

In this way, the main *syntactic* ingredients of a cell–like membrane system (P system) are the *membrane structure*, the *multisets*, and the *evolution rules*.

- A *membrane structure* consists of several membranes arranged hierarchically inside a main membrane (the *skin*), and delimiting *regions* (the space in–between a membrane and the immediately inner membranes, if any). Each membrane identifies a region inside the system. A membrane structure can be represented by a rooted tree.
- Regions defined by a membrane structure contain objects. We shall describe such objects by symbols, in such a way that *multisets of objects* are located in the regions of the membrane structure.
- The objects can evolve according to given *evolution rules*, associated with the regions (hence, with the membranes).

For the *semantics* of the cell–like membrane systems we take a non–deterministic and synchronous mode (a global clock is assumed).

- A *configuration* of a cell–like membrane system consists of a membrane structure and a family of multisets of objects associated with each region of the structure. At the beginning, there is a configuration called the *initial configuration* of the system.
- In each time unit we can transform a given configuration into another one by applying evolution rules to the objects placed inside the regions, in a non–deterministic and maximally parallel manner (the rules are chosen in a non–deterministic way, and in each region all objects that can evolve must do it). In this way, we get *transitions* between configurations.
- A *computation* of the system is a (finite or infinite) sequence of configurations such that each configuration

–except the initial one– is obtained from the previous one by a transition.
- A computation which reaches a configuration where no more rules can be applied to the existing objects is called a *halting computation*.
- The result of a halting computation is usually defined through the multiset associated with a specific output membrane (or the environment) in the final configuration. If the output is collected in the environment, then we say that the system has *external output*.

Thus, a computation in a P system is summarized as follows: it starts with the initial configuration of the system, then the computation proceeds, and when it stops the result is to be found in the output membrane (or in the environment).

In this paper we use membrane computing as a framework to address the resolution of decision problems. In order to solve these kinds of problems, having in mind that solving them is equivalent to recognizing the languages associated with them, we consider P systems as *language recognizer* devices.

*Definition 1:* A *P system with input* is a tuple $(\Pi, \Sigma, i_\Pi)$, where: (a) $\Pi$ is a P system with working alphabet $\Gamma$, with $p$ membranes labelled by $1, \ldots, p$, and initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_p$ associated with them; (b) $\Sigma$ is an (input) alphabet strictly contained in $\Gamma$ and the initial multisets are over $\Gamma - \Sigma$; (c) $i_\Pi$ is the label of a distinguished (input) membrane.

The semantics of a P system with input $(\Pi, \Sigma, i_\Pi)$ is esentially the same as for $\Pi$. However, now the initial configuration of $(\Pi, \Sigma, i_\Pi)$ is not unique, for every possible input multiset over $\Sigma$ there is an associated initial configuration:

*Definition 2:* Let $(\Pi, \Sigma, i_\Pi)$ be a P system with input. Let $\Gamma$ be the working alphabet of $\Pi$, $\mu$ the membrane structure, and $\mathcal{M}_1, \ldots, \mathcal{M}_p$ the initial multisets of $\Pi$. Let $m$ be a multiset over $\Sigma$. The *initial configuration of* $(\Pi, \Sigma, i_\Pi)$ *with input* $m$ is $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots, \mathcal{M}_p)$.

Let $(\Pi, \Sigma, i_\Pi)$ be a P system with input. Let $\Gamma$ be the working alphabet of $\Pi$, $\mu$ the membrane structure and $\mathcal{M}_1, \ldots, \mathcal{M}_p$ the initial multisets of $\Pi$. Let $m$ be a multiset over $\Sigma$. Then we denote $\mathcal{M}_j^* = \{(a, j) : \ a \in \mathcal{M}_j\}$, for $1 \leq j \leq p$, and $m^* = \{(a, i_\Pi) : \ a \in m\}$. This notation will be useful in Section IV.

*Definition 3:* A *recognizer P system* is a P system with external output such that:

1) The working alphabet contains two distinguished elements *yes* and *no*.
2) All computations halt.
3) If $\mathcal{C}$ is a computation of the system, then either object *yes* or object *no* (but not both) must have been released into the environment, and only in the last step of the computation.

In recognizer P systems, we say that a computation $\mathcal{C}$ is an *accepting computation* (respectively, *rejecting computation*) if the object *yes* (respectively, *no*) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$. Hence, these devices send to the environment an accepting or rejecting answer, at the end of their computations.

Let us recall that a decision problem $X$ is a pair $(I_X, \theta_X)$ where $I_X$ is a language over a finite alphabet (whose elements are called *instances*) and $\theta_X$ is a predicate (a total boolean function) over $I_X$.

*Definition 4:* Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer P systems with input. A *polynomial encoding* from $X$ to $\mathbf{\Pi}$ is a pair $(cod, s)$ of polynomial time computable functions over $I_X$ such that for each instance $w \in I_X$, $s(w)$ is a natural number and $cod(w)$ is an input multiset for the system $\Pi(s(w))$.

It is easy to prove that polynomial encodings are stable under polynomial time reductions. More formally:

*Proposition 1:* Let $X_1, X_2$ be decision problems. Let $r$ be a polynomial time reduction from $X_1$ to $X_2$. Let $(cod, s)$ be a polynomial encoding from $X_2$ to $\mathbf{\Pi}$. Then $(cod \circ r, s \circ r)$ is a polynomial encoding from $X_1$ to $\mathbf{\Pi}$.

*Definition 5:* Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer P systems *with input*. Let $(cod, s)$ be a polynomial encoding from $X$ to $\mathbf{\Pi}$.

- We say that the family $\mathbf{\Pi}$ is *sound* with regard to $(X, cod, s)$ if the following holds: for each instance of the problem $w \in I_X$, if there exists an accepting computation of $\Pi(s(w))$ with input $cod(w)$, then $\theta_X(w) = 1$.
- We say that the family $\mathbf{\Pi}$ is *complete* with regard to $(X, cod, s)$ if the following holds: for each instance of the problem $u \in I_X$, if $\theta_X(u) = 1$, then every computation of $\Pi(s(u))$ with input $cod(u)$ is an accepting computation.

Next, we propose to solve a decision problem through a family of P systems (constructed in polynomial time by a deterministic Turing machine) where each element of the family processes, in a specified sense, all the instances of *equivalent size*.

*Definition 6:* Let $\mathcal{R}$ be a class of recognizer P systems with input membrane. A decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ of P systems from $\mathcal{R}$, and we denote this by $X \in \mathbf{PMC}_\mathcal{R}$, 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 $\Pi(w)$ from the instance $w \in I_X$.
- There exists a polynomial encoding $(cod, s)$ from $X$ to $\mathbf{\Pi}$ such that
  - 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 and complete with regard to $(X, cod, s)$.

It is easy to see that the class $\mathbf{PMC}_\mathcal{R}$ is closed under polynomial–time reduction and complement (see [9] for details).

## III. P SYSTEMS WITH MEMBRANE CREATION

In this section we recall the description of cellular devices with membrane creation. Basically there are two ways of producing new membranes in living cells: *mitosis* (membrane division) and *autopoiesis* (membrane creation, see [4]).

The replication is one of the most important functions of a cell and, in ideal circumstances, by division we can obtain an exponential number of cells in linear time.

One of the roles of membranes is to keep the molecules of a compartment close to each other, in order to facilitate their reactions; when a compartment becomes too large, it often happens that new membranes appear (are created) inside it (new membranes are produced under the influence of the existing objects).

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 have been proved to be universal, but up to now there is no theoretical result proving that these models simulate each other in polynomial time. P systems with active membranes have been successfully used to design solutions to many **NP**-complete problems[3], but as Gh. Păun pointed in [8] *"membrane division was much more carefully investigated than membrane creation as a way to obtain tractable solutions to hard problems"*. Recently it has been shown that **NP**–complete problems can also be uniformly solved in the membrane creation framework (see, e.g., [2], [3]).

Recall that a *P system with membrane creation* is a construct of the form $\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_p, R)$ where:

1) $p \geq 1$ is the initial degree of the system; $\Gamma$ is the alphabet of *objects* and $H$ is a finite set of *labels* for membranes;
2) $\mu$ is a *membrane structure* consisting of $p$ membranes, with the membranes injectively labelled with elements of $H$, and $\mathcal{M}_1, \ldots, \mathcal{M}_p$ are strings over $\Gamma$, describing the *multisets of objects* placed in the $p$ regions of $\mu$;
3) $R$ is a finite set of *rules*, of the following forms:
   (a) $[a \to u]_h$ where $h \in H$, $a \in \Gamma$ and $u$ is a string over $\Gamma$ 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 $h \in H$, $a, b \in \Gamma$. These are *send-in communication rules*. An object is introduced in the membrane, possibly modified.
   (c) $[a]_h \to [\,]_h\, b$ where $h \in H$, $a, b \in \Gamma$. These are *send-out communication rules*. An object is sent out of the membrane, possibly modified.
   (d) $[a]_h \to b$ where $h \in H$, $a, b \in \Gamma$. These are *dissolution rules*. Under the influence of an object, a membrane is dissolved, while the object specified in the rule can be modified.
   (e) $[a \to [u]_{h'}]_h$ where $h, h' \in H$, $a \in \Gamma$ and $u$ is a string over $\Gamma$ describing a multiset of objects. These are *creation rules*. As the effect of the evolution of an object, $a$, a new membrane is created. This new membrane is placed inside the membrane of the object which triggers the rule and has associated an initial multiset, $u$, and a label, $h'$.

Rules are applied according to the following principles:

- Rules from (a) to (d) are used as usual in the framework of membrane computing, that is, in a maximally 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.
- Rules of type (e) are also used in a maximally parallel way. Each object $a$ in a membrane labelled with $h$ produces a new membrane with label $h'$ placing in it the multiset of objects described by the string $u$.
- If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external membrane. 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, irrespectively 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.

We denote by $\mathcal{MC}_{-d}$ (respectively, $\mathcal{MC}_{+d}$) the class of recognizer P systems with membrane creation and without dissolution rules (respectively, with dissolution rules).

## IV. DEPENDENCY GRAPH OF A RECOGNIZER P SYSTEM WITH MEMBRANE CREATION

Let $\Pi$ be a recognizer P system with membrane creation and without dissolution. Let $R$ be the set of rules associated with $\Pi$.

Each rule can be considered, in some sense, as a *dependency* relationship between the object triggering the rule and the object(s) produced by its application.

We can consider a general pattern for all kinds of rules of such systems, except for dissolution rules, as follows: $(a, h) \to (a_1, h')(a_2, h') \ldots (a_s, h')$, according to the following criterion:

- The rules of type (a) correspond to the case $h = h'$, $u = a_1 \ldots a_s$, and $s \geq 1$.
- The rules of type (b) correspond to the case $h \in F(h')$ and $s = 1$.
- The rules of type (c) correspond to the case $h' \in F(h)$ and $s = 1$.
- The rules of type (e) correspond to the case $u = a_1 \ldots a_s$, and $s \geq 1$.

If $h$ is the label of a membrane, then $F(h)$ denotes the set of labels $h' \in H$ such that $h'$ is the label of the father of the membrane labelled by $h$ in the initial configuration, or there exist $a \in \Gamma, u \in \Gamma^*$ verifying $[a \to [u]_h]_{h'} \in R$, where $R$ is the set of rules associated with the system. Given $h, h' \in H$

[3]See [15] for a comprehensive bibliography of such solutions.

the cost of determining whether or not $h' \in F(h)$ is of the order $O(|R|+p)$, being $p$ the number of initial membranes.

We adopt the convention that the set $F(h)$ associated with the skin membrane is the singleton whose only element is the label of the environment, denoted by *env* $\in H$.

For example, let us consider a general rule $(a,h) \to (a_1,h')\dots(a_s,h')$. Then we can interpret that from the object $a$ in membrane labelled by $h$ we can *reach* the objects $a_1,\dots,a_s$ in membrane labelled by $h'$.

We formalize these ideas in the following definition.

*Definition 7:* Let $\Pi$ be a recognizer P system with membrane creation and without dissolution. Let $R$ be the set of rules associated with $\Pi$. The *dependency graph associated with* $\Pi$ is the directed graph $G_\Pi = (V_\Pi, E_\Pi)$ defined as follows:

$V_\Pi = VL_\Pi \cup VR_\Pi$,

$(a,h) \in VL_\Pi$ if and only if:

- $\exists u \in \Gamma^*$ $([a \to u]_h \in R)$, or
- $\exists b \in \Gamma$ $([a]_h \to [\ ]_h b \in R)$, or
- $\exists b \in \Gamma$ $\exists h' \in H$ $(h \in F(h') \wedge a[\ ]_{h'} \to [b]_{h'} \in R)$, or
- $\exists u \in \Gamma^*$ $\exists b \in \Gamma$ $\exists h' \in H$ $(b \in \text{alph}(u) \wedge h \in F(h') \wedge [a \to [u]_{h'}]_h \in R)$;

$(b,h) \in VR_\Pi$ if and only if:

- $\exists a \in \Gamma$ $\exists u \in \Gamma^*$ $([a \to u]_h \in R \wedge b \in \text{alph}(u))$, or
- $\exists a \in \Gamma$ $\exists h' \in H$ $(h \in F(h') \wedge [a]_{h'} \to [\ ]_{h'} b \in R)$, or
- $\exists a \in \Gamma$ $(a[\ ]_h \to [b]_h \in R)$, or
- $\exists a \in \Gamma$ $\exists u \in \Gamma^*$ $\exists h' \in F(h)$ $(b \in \text{alph}(u) \wedge [a \to [u]_h]_{h'} \in R)$;

$((a,h),(b,h')) \in E_\Pi$ if and only if:

- $\exists u \in \Gamma^*$ $([a \to u]_h \in R \wedge b \in \text{alph}(u) \wedge h = h') \vee$ $([a]_h \to [\ ]_h b \in R \ \wedge \ h' \in F(h)) \vee$ $(a[\ ]_{h'} \to [b]_{h'} \in R \ \wedge h \ \in F(h'))$, or
- $\exists u \in \Gamma^*$ $(b \in \text{alph}(u) \wedge h \in F(h') \wedge [a \to [u]_{h'}]_h \in R)$.

*Proposition 2:* Let $\Pi$ be a recognizer P system with membrane creation and without dissolution. There exists a Turing machine that constructs the dependency graph, $G_\Pi$, associated with $\Pi$, in polynomial time, that is, in a time bounded by a polynomial function depending on the total number of rules and the maximum length of the rules.

*Proof:* A deterministic algorithm that, given a P system $\Pi$ with the set $R$ of rules, constructs the corresponding dependency graph, is the following:

```
Input: Π (with R as its set of rules)
V_Π ← ∅;  E_Π ← ∅
for each rule r ∈ R of Π do
  if r = [a → u]_h ∧ alph(u) = {a_1,...,a_s} then
     V_Π ← V_Π ∪ ⋃_{j=1}^{s} {(a,h),(a_j,h)};
     E_Π ← E_Π ∪ ⋃_{j=1}^{s} {((a,h),(a_j,h))}
  if r = [a]_h → [ ]_h b then
   for each h' ∈ F(h) do
     V_Π ← V_Π ∪ {(a,h),(b,h')};
     E_Π ← E_Π ∪ {((a,h),(b,h'))}
  if r = a[ ]_h → [b]_h then
   for each h' ∈ F(h) do
     V_Π ← V_Π ∪ {(a,h'),(b,h)};
     E_Π ← E_Π ∪ {((a,h'),(b,h))}
  if r = [a → [u]_{h'}]_h ∧ alph(u) = {a_1,...,a_s} then
   V_Π ← V_Π ∪ ⋃_{j=1}^{s} {(a,h),(a_j,h')};
   E_Π ← E_Π ∪ ⋃_{j=1}^{s} {((a,h),(a_j,h'))}
```

The cost of this algorithm is of the order $O(|R|\cdot(q+|R|+p))$, where $q = \max\{length(r) : r \in R\}$, and $p$ is the number of initial membranes. ■

*Proposition 3:* Let $\Pi$ be a recognizer P system with membrane creation and without dissolution. Let $\Delta_\Pi$ be defined as follows: $(a,h) \in \Delta_\Pi$ if and only if there exists a path (within the dependency graph) from $(a,h)$ to $(\texttt{yes}, env)$. Then, there exists a Turing machine that constructs the set $\Delta_\Pi$ in polynomial time, that is, in a time bounded by a polynomial function depending on the total number of rules and the maximum length of the rules.

*Proof:* We can construct the set $\Delta_\Pi$ from $\Pi$ as follows:

- Construct the dependency graph $G_\Pi$ associated with $\Pi$.
- Consider the following algorithm:

```
Input: G_Π = (V_Π, E_Π)
Δ_Π ← ∅
for each (a,h) ∈ V_Π do
 if there exists a path from (a,h) to
 (yes, env) in G_Π, then
   Δ_Π ← Δ_Π ∪ {(a,h)}
```

The running time of this algorithm is of the order $O(|V_\Pi| \cdot |V_\Pi|^2)$, hence it is of the order $O(|\Gamma|^3 \cdot |H|^3)$, where $\Gamma$ is the working alphabet and $H$ the set of labels of $\Pi$. ■

*Remark 1:* It is easy to design an algorithm running in polynomial time solving the following decision problem: *given a directed graph G, and two nodes $a, b$, determine whether or not the node $b$ is reachable from $a$, that is, whether or not there exists a path in the graph from $a$ to $b$.* For example, given a directed graph $G$, and two nodes $a, b$, we consider a depth–first–search with source $a$, and we check if $b$ is in the tree of the computation forest whose root is $a$. The total running time of this algorithm is $O(|V|+|E|)$, that is, in the worst case is quadratic in the number of nodes. Moreover, this algorithm needs to store a linear number of items.

*Proposition 4:* Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$ be a family of recognizer P systems with input membrane solving $X$, according to Definition 6. Let $(cod, s)$ be the polynomial encoding associated with that solution. Then, for each instance $w$ of the problem $X$ the following assertions are equivalent:

(a) $\theta_X(w) = 1$ *(*that is, the answer to the problem is `yes` for $w$*).*

(b) $\Delta_{\Pi(s(w))} \cap ((cod(w))^* \cup \bigcup_{j=1}^{p} \mathcal{M}_j^*) \neq \emptyset$, where $\mathcal{M}_1,\dots,\mathcal{M}_p$ are the initial multisets of the system $\Pi(s(w))$.

*Proof:* Let $w \in I_X$. Then $w \in L_X$ if and only if there exists an accepting computation of the system $\Pi(s(w))$ with input multiset $cod(w)$. But this condition is equivalent to the following: in the initial configuration of $\Pi(s(w))$ with input multiset $cod(w)$ there exists an object $a \in \Gamma$ in a membrane labelled by $h$ such that in the dependency graph the node $(\texttt{yes}, env)$ is reachable from $(a, h)$.

Hence, $\theta_X(w) = 1$ if and only if $\Delta_{\Pi(s(w))} \cap \mathcal{M}_1^* \neq \emptyset$, or ..., or $\Delta_{\Pi(s(w))} \cap \mathcal{M}_p^* \neq \emptyset$, or $\Delta_{\Pi(s(w))} \cap (cod(w))^* \neq \emptyset$. ■

Next, we show that, in the framework of recognizer P systems with membrane creation (but not using dissolution rules), constructing in polynomial time an exponential workspace (number of membranes) is not enough to solve **NP**–complete problems in polynomial time (unless $\mathbf{P} = \mathbf{NP}$).

*Theorem 1:* $\mathbf{PMC}_{\mathcal{MC}_{-d}} = \mathbf{P}$.

*Proof:* We have $\mathbf{P} \subseteq \mathbf{PMC}_{\mathcal{MC}_{-d}}$ because the class $\mathbf{PMC}_{\mathcal{MC}_{-d}}$ is closed under polynomial time reduction. Next, we show that $\mathbf{PMC}_{\mathcal{MC}_{-d}} \subseteq \mathbf{P}$. For that, let $X \in \mathbf{PMC}_{\mathcal{MC}_{-d}}$. Let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer P systems with membrane creation and without dissolution solving $X$, according to Definition 6. Let $(cod, s)$ be the polynomial encoding associated with that solution.

We consider the following deterministic algorithm:

```
Input: An instance w of X
  - Construct the system Π(s(w)) with input multiset cod(w)
  - Construct the dependency graph G_{Π(s(w))} associated with Π(s(w))
  - Construct the set Δ_{Π(s(w))} according to Proposition 3
  - Consider the following algorithm:
    answer ← no; j ← 1
    while j ≤ p ∧ answer =no do
       if Δ_{Π(s(w))} ∩ M_j^* ≠ ∅ then
          answer ← yes
       j ← j + 1
    endwhile
    if Δ_{Π(s(w))} ∩ (cod(w))^* ≠ ∅ then
       answer ← yes
```

On one hand, the answer of this algorithm is `yes` if and only if there exists a pair $(a, h)$ belonging to $\Delta_{\Pi(s(w))}$ such that in the membrane labelled by $h$ in the initial configuration (with input the multiset $cod(w)$) appears the symbol $a$.

On the other hand, a pair $(a, h)$ belongs to $\Delta_{\Pi(s(w))}$ if and only if there exists a path from $(a, h)$ to $(\texttt{yes}, env)$; that this, if and only if we can obtain an accepting computation of $\Pi(s(w))$ with input $cod(w)$. Hence, the algorithm above described solves the problem $X$.

The cost to determine whether or not $\Delta_{\Pi(s(w))} \cap \mathcal{M}_j^* \neq \emptyset$ (or $\Delta_{\Pi(s(w))} \cap (cod(w))^* \neq \emptyset$) is of the order $O(|\Gamma|^2 \cdot |H|^2)$.

Hence, the running in time of this algorithm can be bounded by $f(|w|) + O(|R| \cdot q) + O(p \cdot |\Gamma|^2 \cdot |H|^2)$, where $f$ is the (total) cost of a polynomial encoding from $X$ to $\mathbf{\Pi}$, $R$ the set of rules of $\Pi$, $p$ is the initial number of membranes, and $q = \max\{length(r) : r \in R\}$. But from Definition 6 we have that all involved parameters are polynomials in $|w|$. That is, the algorithm is polynomial in the size $|w|$ of the input. ■

## V. Solving QSAT in linear time

In this section we design a family of recognizer P systems with membrane creation (and using dissolution rules) which solves QSAT (the quantified satisfiability problem) in linear time.

Given a boolean formula $\varphi(x_1, \ldots, x_n)$ in conjunctive normal form, with boolean variables $x_1, \ldots, x_n$, the sentence $\varphi^* = \exists x_1 \forall x_2 \ldots Q_n x_n \varphi(x_1, \ldots, x_n)$ (where $Q_n$ is $\exists$ if $n$ is odd, and $Q_n$ is $\forall$ otherwise) is said to be the (existential) *fully quantified* formula associated with $\varphi(x_1, \ldots, x_n)$. Recall that a sentence is a boolean formula in which every variable is in scope of a quantifier.

We say that $\varphi^*$ is satisfiable if there exists a truth assignment, $\sigma$, over $\{i : 1 \leq i \leq n \wedge i \text{ odd}\}$ such that each extension, $\sigma^*$, of $\sigma$ over $\{1, \ldots, n\}$ verifies $\sigma^*(\varphi(x_1, \ldots, x_n)) = 1$.

The QSAT problem is the following one: *Given the (existential) fully quantified formula $\varphi^*$ associated with a boolean formula $\varphi(x_1, \ldots, x_n)$ in conjunctive normal form, determine whether or not $\varphi^*$ is satisfiable.*

It is well known that QSAT is a **PSPACE**–complete problem [5].

Next, we provide a polynomial time solution to QSAT by a family of recognizer P systems with membrane creation and using dissolution rules, according to Definition 6.

The solution follows a brute force approach, in the framework of recognizer P systems with membrane creation, and consists in the following phases:

- *Generation and Evaluation Stage*: Using membrane creation, a binary complete tree is constructed. The leaves of that tree encode all possible truth assignments associated with the formula. Nodes whose level is even (respectively, odd) are codified by an OR gate (respectively, AND gate). So, we can consider the constructed tree as a boolean circuit that only have gates AND, OR. In this stage, the values of the formula corresponding to each assignment is obtained in the leaves.
- *Checking Stage*: We proceed to compute the output of that boolean circuit from the inputs obtained in the leaves by propagating values along the wires and computing the respective gates until the output gate (the root of the tree) has assigned a value.
- *Output Stage*: The systems sends out to the environment the right answer according to the result of the previous stage.

Let us consider the pair function $\langle\ ,\ \rangle$ defined by

$$\langle n, m \rangle = ((n + m)(n + m + 1)/2) + n.$$

This function is polynomial-time computable (it is primitive recursive and bijective from $\mathbf{N}^2$ onto $\mathbf{N}$).

For any given boolean formula, $\varphi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$, in conjunctive normal form, with $n$ variables and

$m$ clauses, we construct a P system $\Pi(\langle n,m\rangle)$ processing the (existential) fully quantified formula $\varphi^*$ associated with $\varphi$ (when an appropriate input is supplied).

The family of recognizer P systems with membrane creation and using dissolution rules presented here is

$$\mathbf{\Pi} = \{(\Pi(\langle n,m\rangle), \Sigma(\langle n,m\rangle), i(\langle n,m\rangle)) \ : \ (n,m) \in \mathbf{N}^2\}$$

where the input alphabet is

$$\Sigma(\langle n,m\rangle) = \{x_{i,j}, \overline{x}_{i,j} \ : \ 1 \leq i \leq m, 1 \leq j \leq n\}$$

the input membrane is $i(\langle n,m\rangle) =< t, \vee >$, and the P system

$$\Pi(\langle n,m\rangle) = (\Gamma(\langle n,m\rangle), H(\langle n,m\rangle), \mu, \mathcal{M}_s, \mathcal{M}_{<t,\vee>}, R(\langle n,m\rangle))$$

is defined as follows:

• Working alphabet, $\Gamma(\langle n,m\rangle)$, is

$$\begin{array}{l}
\Sigma(\langle n,m\rangle) \cup \\
\{z_{j,c} \,|\, j \in \{0,\ldots n\},\, c \in \{\wedge,\vee\}\} \cup \\
\{z_{j,c,l} \,|\, j \in \{0,\ldots,n-1\},\, c \in \{\wedge,\vee\},\, l \in \{t,f\}\} \cup \\
\{x_{i,j,l}, \overline{x}_{i,j,l} \,|\, j \in \{1,\ldots,n\},\, i \in \{1,\ldots,m\},\, l \in \{t,f\}\} \cup \\
\{x_{i,j} \,|\, j \in \{1,\ldots,n\},\, i \in \{1,\ldots,m\}\} \cup \\
\{r_i, r_{i,t}, r_{i,f} \,|\, i \in \{1,\ldots,m\}\} \cup \\
\{d_1,\ldots,d_m, q, t_0,\ldots,t_4, ans_0,\ldots,ans_5\} \cup \\
\{yes, yes_\vee, yes^*, yes_\wedge, \underline{yes}_\wedge, YES\} \cup \\
\{no, no_\vee, no^*, \underline{no}_\vee, no_\wedge, \underline{no}_\wedge, NO\}
\end{array}$$

• The set of labels, $H(\langle n,m\rangle)$, is

$$\{< l,c >:\ l \in \{t,f\}, c \in \{\wedge,\vee\}\} \cup \{a,s,1,\ldots,m\}$$

• Initial membrane structure: $\mu = [\,[\,]_{<t,\vee>}\,]_s$

• Initial multisets: $\mathcal{M}_s = \emptyset$, $\mathcal{M}_{<t,\vee>} = \{z_{0,\wedge,t} z_{0,\wedge,f}\}$

• Input membrane: $[\,]_{<t,\vee>}$

• The set of evolution rules, $R(\langle n,m\rangle)$, consists of the following rules (recall that $\lambda$ denotes the empty string, and if $c$ is $\wedge$ then $\overline{c}$ is $\vee$, and if $c$ is $\vee$ then $\overline{c}$ is $\wedge$):

**1.** $\left.\begin{array}{l} [z_{j,c} \to z_{j,c,t}, z_{j,c,f}]_{<l,\overline{c}>} \\ [z_{j,c,l} \to [z_{j+1,\overline{c}}]_{<l,c>}]_{<l',\overline{c}>} \end{array}\right\}$,

for $l, l' \in \{t,f\}, c \in \{\vee,\wedge\}, j \in \{0,\ldots,n-1\}$.

The goal of these rules is to create one membrane for each assignment to the variables of the formula. Firstly, the object $z_{j,c}$ evolves to two objects, one for the assignment *true* (the object $z_{j,c,t}$), and a second one for the assignment *false* (the object $z_{j,c,f}$). In a second step these objects will create two membranes. The new membrane with $t$ in its label represents the assignment $x_{j+1} = true$; on the other hand, the new membrane with $f$ in its label represents the assignment $x_{j+1} = false$.

**2.** $\left.\begin{array}{l} [x_{i,j} \to x_{i,j,t} x_{i,j,f}]_{<l,c>} \\ [\overline{x}_{i,j} \to \overline{x}_{i,j,t} \overline{x}_{i,j,f}]_{<l,c>} \\ [r_i \to r_{i,t} r_{i,f}]_{<l,c>} \end{array}\right\}$,

for $l \in \{t,f\}, i \in \{1,\ldots,m\}, c \in \{\vee,\wedge\}, j \in \{1,\ldots,n\}$.

These rules duplicate the objects representing the formula so it can be evaluated on the two possible assignments, $x_j = true$ ($x_{i,j,t}, \overline{x}_{i,j,t}$) and $x_j = false$ ($x_{i,j,f}, \overline{x}_{i,j,f}$). The objects $r_i$ are also duplicated ($r_{i,t}, r_{i,f}$) in order to keep track of the clauses that evaluate true on the previous assignments to the variables.

**3.** $\left.\begin{array}{ll} x_{i,1,t}[\,]_{<t,c>} \to [r_i]_{<t,c>} & \overline{x}_{i,1,t}[\,]_{<t,c>} \to [\lambda]_{<t,c>} \\ x_{i,1,f}[\,]_{<f,c>} \to [\lambda]_{<f,c>} & \overline{x}_{i,1,f}[\,]_{<f,c>} \to [r_i]_{<f,c>} \end{array}\right\}$,

for $i \in \{1,\ldots,m\}, c \in \{\vee,\wedge\}$.

According to these rules the formula is evaluated in the two possible assignments for the variable that is being analyzed. The objects $x_{i,1,t}$ (resp. $\overline{x}_{i,1,f}$) get into the membrane with $t$ in its label (resp. $f$) being transformed into the objects $r_i$ representing that the clause number $i$ evaluates true on the assignment $x_{j+1} = true$ (resp. $x_{j+1} = false$). On the other hand, the objects $\overline{x}_{i,1,t}$ (resp. $x_{i,1,t}$) get into the membrane with $f$ in its label (resp. $t$) producing no objects. This represents that these objects do not make the clause true in the assignment $x_{j+1} = true$ (resp. $x_{j+1} = false$).

**4.** $\left.\begin{array}{l} x_{i,j,l}[\,]_{<l,c>} \to [x_{i,j-1}]_{<l,c>} \\ \overline{x}_{i,j,t}[\,]_{<l,c>} \to [\overline{x}_{i,j-1}]_{<l,c>} \\ r_{i,t}[\,]_{<l,c>} \to [r_i]_{<l,c>} \end{array}\right\}$,

for $l \in \{t,f\}, i \in \{1,\ldots,m\}, c \in \{\vee,\wedge\}, j \in \{2,\ldots,n\}$.

In order to analyze the next variable the second subscript of the objects $x_{i,j,l}$ and $\overline{x}_{i,j,l}$ are decreased when they are sent into the corresponding membrane labelled with $l$. Moreover, following the last rule, the objects $r_{i,l}$ get into the new membranes to keep track of the clauses that evaluate true on the previous assignments.

**5.** $[z_{n,c} \to d_1 \ldots d_m q]_{<l,\overline{c}>}$, for $l \in \{t,f\}$ and $c \in \{\vee,\wedge\}$.

At the end of the generation stage the object $z_n$ will produce the objects $d_1,\ldots,d_m$ and $yes_0$, which will take part in the checking stage.

**6.** $\left.\begin{array}{lll} [d_i \to [t_0]_i]_{<l,c>} & & \\ r_{i,t}[\,]_i \to [r_i]_i & [r_i]_i \to \lambda & \\ [t_j \to t_{j+1}]_i & [t_2]_i \to t_3 & \end{array}\right\}$,

for $i \in \{1,\ldots,m\}, j \in \{0,1\}, l \in \{t,f\}, c \in \{\vee,\wedge\}$.

Following these rules each object $d_i$ creates a new membrane with label $i$ where the object $t_0$ is placed; this object will act as a counter. The object $r_i$ gets into the membrane labelled with $i$ and dissolves it preventing the counter $t_i$ from reaching the object $t_2$. The fact that the object $t_2$ appears in a membrane with label $i$ means that there is no object $r_i$, that is, the clause number $i$ does not evaluate true on the assignment associated with the membrane; therefore neither does the formula on the associated assignment.

**7.** $\left.\begin{array}{lll} [q \to [ans_0]_a]_{<l,c>} & & \\ t_3[\,]_a \to [t_4]_a & [t_4]_a \to \lambda & \\ [ans_h \to ans_{h+1}]_a & [ans_5]_a \to yes & \\ [ans_5 \to no]_{<l,c>} & & \end{array}\right\}$,

for $l \in \{t,f\}, c \in \{\vee,\wedge\}, h = 0,\ldots,4$.

The object $q$ creates a membrane with label $a$ where the object $ans_0$ is placed. The object $ans_h$ evolves to the object $ans_{h+1}$; at the same time the objects $t_3$ can get into the membrane labelled with $a$ and dissolve it preventing the object $yes$ from being sent out from this membrane.

$$\mathbf{8}.\ \left.\begin{array}{ll} [yes]_{<l,c>} \rightarrow yes_{\overline{c}} & [no]_{<l,c>} \rightarrow no_{\overline{c}} \\ [yes_{\vee}]_{<l,\vee>} \rightarrow yes^* & [no_{\vee} \rightarrow \underline{no}_{\vee}]_{<l,\vee>} \\ [yes^* \rightarrow yes_{\wedge}]_{<l,\wedge>} & [\underline{no}_{\vee}]_{<l,\vee>} \rightarrow no_{\wedge} \\ [\underline{no}_{\vee} \rightarrow \lambda]_{<l,\wedge>} & [yes_{\vee} \rightarrow \lambda]_{<l,\wedge>} \\ [no_{\wedge}]_{<l,\wedge>} \rightarrow no^* & [yes_{\wedge} \rightarrow \underline{yes}_{\wedge}]_{<l,\wedge>} \\ [no^* \rightarrow no_{\vee}]_{<l,\vee>} & [\underline{yes}_{\wedge}]_{<l,\wedge>} \rightarrow yes_{\vee} \\ [\underline{no}_{\wedge} \rightarrow \lambda]_{<l,\vee>} & [\underline{yes}_{\wedge} \rightarrow \lambda]_{<l,\vee>} \\ [yes^*]_s \rightarrow YES\,[\,]_s & [no_{\wedge}]_s \rightarrow NO\,[\,]_s \end{array}\right\},$$

for $l \in \{t, f\}$.

This set of rules controls the output stage. After the evaluation stage, from each working membrane we obtain an object *yes* or *no* depending on whether the assignment associated with this membrane satisfies or not the formula. Contrary to the SAT problem, in QSAT it is not enough that one assignment satisfies the formula, but the final answer is YES if an appropriate combination of assignments according to the quantifiers $\exists$ and $\forall$ is found.

### A. An overview of the computation

First of all we define a polynomial encoding of the QSAT problem in the family $\mathbf{\Pi}$ constructed in the previous section. Given a boolean formula in conjunctive normal form, $\varphi = C_1 \wedge \ldots \wedge C_m$ such that $Var(\varphi) = \{x_1, \ldots, x_n\}$, and considering $\varphi^*$ the (existential) fully quantified formula associated with it, we define $s(\varphi^*) = \langle n, m \rangle$ (recall the bijection mentioned in the previous section) and $cod(\varphi^*) = \{x_{i,j} : x_j \in C_i\} \cup \{\overline{x}_{i,j} : \neg x_j \in C_i\}$.

Next we describe informally how the recognizer P system with membrane creation $\Pi(s(\varphi^*))$ with input $cod(\varphi^*)$ works.

In the initial configuration we have the input multiset $cod(\varphi^*)$ and the objects $z_{0,\wedge,t}$ and $z_{0,\wedge,f}$ placed in the input membrane (membrane labelled with $< t, \vee >$). In the first step of the computation the object $z_{0,\wedge,t}$ creates a new membrane with label $< t, \vee >$ which represents the assignment $x_1 = true$ and the object $z_{0,\wedge,f}$ creates a new membrane with label $< f, \vee >$ which represents the assignment $x_1 = false$. The second component of the labels, i.e., $\wedge$ and $\vee$ will be used in the output stage.

In these two new membranes the object $z_{1,\vee}$ is placed. At the same time the input multiset representing the formula $\varphi^*$ is duplicated following the two first rules in group 2. In the next step, according to the rules in group 3, the formula is evaluated on the two possible assignments for $x_1$. In the same step the rules in group 4 decrease the second subscript of the objects representing the formula ($x_{i,j,l}, \overline{x}_{i,j,l}$ with $j \geq 2$) in order to analyze the next variable. Moreover, at the same time, the object $z_{1,c}$ produces the object $z_{1,c,t}$ and $z_{1,c,f}$ ($c \in \{\wedge, \vee\}$) and the system is ready to analyze the next variable. The generation and evaluation stage goes on in this way until all the possible assignments to the variables are generated and the formula is evaluated on each of them. Observe that it takes two steps to generate the possible assignments for a variable and evaluate the formula on them; therefore the generation and evaluation stage takes $2n$ steps.

The checking stage starts when the object $z_{n,c}$ produces the objects $d_1, \ldots, d_m$ and the object $q$. In the first step of the checking stage each object $d_i$, for $i = 1, \ldots, m$, creates a new membrane labelled with $i$ where the object $t_0$ is placed, and the object $q$ creates a new membrane with label $a$ placing the object $yes_0$ in it.

The objects $r_{i,t}$, which indicate that the clause number $i$ evaluates true on the assignment associated with the membrane, are sent into the membranes by the last rule in group 4 so the system keeps track of the clauses that are true. The objects $r_{i,t}$ get into the membrane with label $i$ and dissolves it in the following two steps preventing the counter $t_2$ from dissolving the membrane and producing the object $t_3$ according to the last rule in group 6. If for some $i$ there is no object $r_i$, which means that the clause $i$ does not evaluate true on the associated assignment, the object $t_2$ will dissolve the membrane labelled with $i$ producing the object $t_3$ that will get into the membrane with label $a$ where the object $ans_h$ evolves following the rules in group 7. The object $t_4$ dissolves the membrane with label $a$ preventing the production of the object $ans_5$. Therefore the checking stage takes 7 steps.

Finally the output stage takes place according to the rules in group 8. If some object $ans_5$ is present in any membrane with label $< l, c >$, ($l \in \{t, f\}$, $c \in \{\wedge, \vee\}$), this means that there exists at least one clause not satisfied by the assignment associated with the membrane, and by the last rule in group 7 we obtain $no$ in this membrane. Otherwise, the object $ans_5$ will be inside the membrane with label $a$, it will dissolve the membrane, and send $yes$ to the working membrane.

At this point, in each of the $2^n$ working membranes we have an object $yes$ or $no$ depending on if the associated assignment satisfies or not the formula $\varphi$. In the last steps we control the flow of the objects $yes$ and $no$ from the working membranes to the environment. Basically, the process is the following: if there is one object $yes$ inside a membrane with $\vee$ in its label, this object dissolves the membrane and sends out another $yes$; if this does not happen, i.e., if two objects $no$ are inside a membrane with label $\vee$, the membrane is dissolved and $no$ is sent out. Analogously, if there is one object $no$ inside a membrane with $\wedge$ in its label, this object dissolves the membrane and sends out another $no$. Otherwise, if two objects $yes$ are inside a membrane with label $\vee$, the membrane is dissolved and $yes$ is sent out.

If the answer is affirmative, then the system halts in the (4n+8)–step. If, on the contrary, the answer is negative, then the system halts in the (4n+9)–step.

Hence, the family $\mathbf{\Pi}$ of recognizer P systems with membrane creation using dissolution rules solves in polynomial (actually, linear) time QSAT according to Definition 6. So, we have the following result.

*Theorem 2:* $QSAT \in \mathbf{PMC}_{\mathcal{MC}_{+d}}$

*Corollary 1:* $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\mathcal{MC}_{+d}}$

*Proof:* It suffices to remark that the QSAT problem is **PSPACE**–complete, $QSAT \in \mathbf{PMC}_{\mathcal{MC}_{+d}}$, and this complexity class is closed under polynomial time reduction. ■

## VI. Conclusions

It is a very interesting issue to obtain conditions providing a distinction between tractable problems (that is, those that

are solvable by computational devices running in polynomial time) and the problems that are not tractable.

This paper is focused in that direction and it wishes to stress the relevant role played by an apparently innocent operation (as dissolution rules) in order to "separate" the complexity classes **P** and **PSPACE**. Thus, in the framework of recognizer P systems with membrane creation, dissolution rules permit to distinguish the tractability of decision problems, suppossing $\mathbf{P} \neq \mathbf{PSPACE}$.

In our characterization of the class **P**, we have used the concept of *dependency graph*, that initially was defined to help to design strategies looking for short computations of confluent membrane systems. We characterize the accepting computations of recognizer P systems with membrane creation and without dissolution through the reachability of a distinguished node of the graph from other nodes associated with the initial configuration.

## ACKNOWLEDGMENT

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 (University of Seville), in the case of the fourth author.

## REFERENCES

[1] 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 (2005), 673–678.

[2] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: A linear solution of Subset Sum Problem by using Membrane Creation. In *Mechanisms, symbols and models underlying cognition, First International Work-Conference on the interplay between Natural and Artificial Computation, IWINAC 2005* (J. Mira, J.R. Alvarez, eds.), LNCS 3561 (2005), 258–267.

[3] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: Solving SAT with Membrane Creation. In *Computability in Europe 2005, CiE 2005: New Computational Paradigms* (S. Barry Cooper, B. Lowe, L. Torenvliet, eds.), Report ILLC X-2005-01, University of Amsterdam, 82–91.

[4] P.L. Luisi: The chemical implementation of autopoiesis, *Self-Production of Supramolecular Structures* (G.R. Fleishaker et al., eds.), Kluwer, Dordrecht, 1994.

[5] Ch.H. Papadimitriou: *Computational Complexity*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

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

[7] Gh. Păun: *Membrane Computing. An Introduction*, Springer–Verlag, Berlin, 2002.

[8] Gh. Păun: Further open problems in membrane computing. *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, eds.), Report RGNC 01/04, University of Seville, 2004, 354–365.

[9] M.J. Pérez–Jiménez: An approach to computational complexity in Membrane Computing. In *Membrane Computing, 5th International Workshop, WMC5, Revised Selected and Invited Papers* (G. Mauri, Gh. Păun, M. J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, eds.), LNCS 3365 (2005), 85–109.

[10] M.J. Pérez-Jiménez, F.J. Romero-Campero: 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-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, eds.), Report RGNC 01/04, University of Seville, 2004, 414–430.

[11] M.J. Pérez-Jiménez, A. Riscos-Núñez: Solving the Subset-Sum problem by P systems with active membranes, *New Generation Computing*, 23, 4 (2005), 367–384.

[12] M.J. Pérez-Jiménez, A. Riscos-Núñez: A linear–time solution for the Knapsack problem using P systems with active membranes, *Membrane Computing* (C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 2933 (2004), 250–268.

[13] 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. Vaszil, eds.), 2003, 284-294.

[14] ISI web page: *http://esi-topics.com/erf/october2003.html*

[15] P systems web page: *http://psystems.disco.unimib.it/*