A uniform solution to SAT using membrane creation

Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez*, Francisco 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

Abstract

In living cells, new membranes are produced basically through two processes: mitosis and autopoiesis. These two processes have inspired two variants of cell-like membrane systems, namely P systems with active membranes and P systems with membrane creation. In this paper, we provide the first uniform, efficient solution to the SAT problem in the framework of recogniser P systems with membrane creation using dissolution rules. Recently the authors have proved that if the dissolution rules are not allowed to be used, then the polynomial complexity class associated with this variant of P systems is the standard complexity class **P**. This result, together with the main result of this paper, shows the surprising role of the apparently "innocent" operation of membrane dissolution. The use of this type of rule establishes the difference between efficiency and non-efficiency for P systems with membrane creation, and provides a barrier between **P** and **NP** (assuming $\mathbf{P} \neq \mathbf{NP}$).

Keywords: Natural computing; Membrane computing; Cellular complexity classes; SAT problem

1. Introduction

Membrane computing is an emergent branch of natural computing introduced by Păun in [12]. 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 [11] was mentioned in [14] as a highly cited paper in October 2003.

This non-deterministic model of computation starts from the assumption that the processes taking place in the compartmental structures of living cells 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 that evolve according to given rules in a synchronous non-deterministic maximally parallel manner.[1] The representation of data as multisets is an abstraction from the way in which chemical compounds are found in living cells. Membrane computing is a cross-disciplinary field, with contributions by computer scientists,

* Corresponding author. Tel.: +34 954557952; fax: +34 954557952.
E-mail address: marper@us.es (M.J. Pérez-Jiménez).

[1] An introduction can be found in [6], and updated information at [15].

biologists, formal linguists and complexity theoreticians, enriching each others' results and problems, and promising new research lines.

In this paper, we present a contribution from the computational side. We introduce a family of P systems with *membrane creation*, constructed in a *uniform way*, that solves the problem of determining, for a given formula in conjunctive normal form, whether it is *satisfiable* or not (the SAT problem).

The paper is organised as follows: first P systems with membrane creation are introduced in the next section. In Section 3, recogniser P systems (devices that capture the intuitive idea underlying the concept of an algorithm) are presented. The solution in the framework of *membrane creation* to the SAT problem is given in Section 4. Finally, some formal details and conclusions are given.

2. P systems with membrane creation

Polynomial solutions to **NP**-complete problems in membrane computing are produced by trading time for space. This is inspired by the capability of cells to produce an exponential number of new membranes (new workspaces) in polynomial time. Basically, there are two ways of producing new membranes in living cells: *mitosis* (membrane division) and *autopoiesis* (membrane creation, see [5]). 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 **NP**-complete problems, as SAT [10], Subset Sum [7], Knapsack [8], Bin Packing [9] and Partition [2], but as Păun pointed in [13] *"membrane division was much more carefully investigated than membrane creation as a way to obtain tractable solutions to hard problems"*.

In this paper, we investigate the second variant mentioned above. 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 the 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 $m \geq 1$ is the initial degree of the system; O is the alphabet of *objects*, and H is a finite set of *labels* for membranes; μ is a *membrane structure*, consisting of m membranes injectively labelled with elements of H, and $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ; R is a finite set of *rules*, of the 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 *(object evolution rules)* associated with membranes and depending only on the label of the membrane.
(b) $a[\,]_h \to [b]_h$ where $h \in H$, $a, b \in O$ *(send-in communication rules)*. An object is introduced in the membrane, possibly modified.
(c) $[a]_h \to [\,]_h\, b$ where $h \in H$, $a, b \in O$ *(send-out communication rules)*. An object is sent out of the membrane, possibly modified.
(d) $[a]_h \to b$ where $h \in H$, $a, b \in O$ *(dissolution rules)*. A membrane is dissolved in reaction with an object, which 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 *(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 (d) are used as is usual in the framework of membrane computing, i.e., in a maximal parallel way. In one step, each object in a membrane can only be used for one rule (non- deterministically chosen), but any object which can evolve by a rule must do it (with the restrictions indicated below).
- Rules of type (e) are used also in a maximal parallel way. Each object a in a membrane labelled with h_1 produces a new membrane with label h_2, placing in it the multiset of objects described by the string v.
- If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external one. The skin is never dissolved.

- All the elements which are not involved in any of the operations to be applied remain unchanged.
- Rules associated with the label h are used for all membranes with this label, irrespective of whether the membrane is an initial one or whether it was created.
- Several rules can be applied to different objects in the same membrane simultaneously. The exceptions are the rules of type (d), since a membrane can be dissolved only once.

3. Recogniser P systems with membrane creation

Recogniser P systems were introduced in [8], 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 to the language associated with the problem or not.

In the literature, recogniser P systems are associated in a natural way with P systems with *input*. The data related to an instance of the decision problem has to be provided to the P system in order for it to compute the appropriate answer. This is done by codifying each instance as a multiset placed in an *input membrane*. The output of the computation, (*yes* or *no*), is sent to the environment. 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 their non-determinism. The design of a family of recogniser P systems has to consider it, because all possibilities in the non-deterministic computations have to output the same answer. This can be summarised 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 *recogniser 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* and *no*.
(2) All its computations halt.
(3) If 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 C is an accepting computation (respectively, rejecting computation) if the object *yes* (respectively, *no*) appears in the external environment associated with the corresponding halting configuration of C.

In the next section, we present a uniform solution to the SAT problem in linear time in the following sense.

Definition 3. Let F be a class of recogniser P systems. 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 P systems from F, and we denote this by $X \in \mathbf{PMC}_F$, if the following holds:

- 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 exist a pair (cod, s) of polynomial-time computable functions over I_X such that:
  - For each instance $u \in I_X$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$.
  - 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 $u \in I_X$, each computation of $\Pi(s(u))$ with input $cod(u)$ performs at most $p(|u|)$ steps.
  - The family $\mathbf{\Pi}$ is sound with regard to (X, cod, s); i.e., 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 a requirement for 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 the class $\mathbf{PMC}_F$ is closed under polynomial-time reduction and complement, see [10]. In this paper, we will deal with the class $\mathcal{MC}$ of recogniser P systems with membrane creation.

4. Solving SAT in linear time with membrane creation

The SAT problem is the following: *Given a Boolean formula in conjunctive normal form (CNF), to determine whether or not it is satisfiable, that is, whether there exists an assignment to its variables on which it evaluates to true.*

In this section, we describe a family of P systems which solves this problem. We will address the resolution via a brute force algorithm, in the framework of recogniser P systems with membrane creation, which consists in the following stages:

• *Generation and evaluation stage:* Using membrane creation, we will generate all possible assignments associated with the formula and evaluate it for each.

• *Checking stage:* In each membrane, we check whether or not the formula evaluates to true on the assignment associated with it.

• *Output stage:* The systems sends out to the environment the right answer.

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 $\mathbb{N}^2$ onto $\mathbb{N}$). For any given formula in CNF, $\varphi = C_1 \wedge \cdots \wedge C_m$, with n variables and m clauses, we construct a P system $\Pi(\langle n, m\rangle)$ solving it. Therefore the family presented here is

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

For each element of the family, the input membrane is $i(\langle n, m\rangle) = t$, the input alphabet is $\Sigma(\langle n, m\rangle) = \{x_{i,j}, \overline{x}_{i,j} \ : \ 1 \leq i \leq m, 1 \leq j \leq n\}$ and the P system $\Pi(\langle n, m\rangle) = (\Gamma(\langle n, m\rangle), \{a, t, f, 1, \ldots, m\}, \mu, w_a, w_t, R(\langle n, m\rangle))$ is defined as:

• Working alphabet: $\Gamma(\langle n, m\rangle) = \Sigma(\langle n, m\rangle) \cup \{x_{i,j,l}, \overline{x}_{i,j,l}, z_i, z_{i,l}, r_j, r_{j,l}, d_j \ : \ l = t, f, \ 1 \leq i \leq n, \ 1 \leq j \leq m\} \cup \ \{yes, no, yes_i, no_j \ : \ 0 \leq i \leq 9, 0 \leq j \leq 2n + 11\} \cup \ \{q, k_0, k_1, k_2, t_0, t_1, t_2, t_3\}$.

Note that the size of the alphabet is $6nm + 5n + 4m + 32 \in \Theta(nm)$, and recall that the size of an instance of the SAT problem, a formula with n variables and m clauses, is of the order $\Omega(nm)$; therefore the size of the working alphabet is *linear* on the size of the input.

• Initial membrane structure: $\mu = [\,[\,]_a\,]_t$

• Initial Multisets: $w_a = \{no_0\}$ $w_t = \{z_{0,t}, \ z_{0,f}\}$

• The set of evolution rules, $R(\langle n, m\rangle)$, consists of the following (recall that λ denotes the empty string):

1. $[z_{j,t} \to [z_{j+1}\, k_0]_t]_l \quad [z_{j,f} \to [z_{j+1}\, k_0]_f]_l$ for $l = t$, f and $j = 0, \ldots, n - 2$.

The goal of these rules is to create one membrane for each assignment to the variables of the formula. The new membrane with label t, where the object z_{j+1} is placed, represents the assignment $x_{j+1} = true$; on the other hand the new membrane with label f, where the object z_{j+1} is placed represents the assignment $x_{j+1} = false$.

2. $$\left.\begin{array}{ll} [x_{ij} \to x_{i,j,t}x_{i,j,f}]_l & [r_i \to r_{i,t}r_{i,f}]_l \\ [\overline{x}_{i,j} \to \overline{x}_{i,j,t}\overline{x}_{i,j,f}]_l & [z_k \to z_{k,t}\, z_{k,f}]_l \end{array}\right\} \quad \text{for} \quad \begin{array}{l} l = t, f; \ k = 0, \ldots, n-1 \\ i = 1, \ldots, m; \ j = 1, \ldots n. \end{array}$$

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 to true on the previous assignments to the variables. The objects z_k produce the objects $z_{k,t}$ and $z_{k,f}$ which will create the new membranes representing the two possible assignments for the next variable.

3. $$\left.\begin{array}{ll} x_{i,1,t}[\,]_t \to [r_i]_t, & \overline{x}_{i,1,t}[\,]_t \to [\lambda]_t \\ x_{i,1,f}[\,]_f \to [\lambda]_f, & \overline{x}_{i,1,f}[\,]_f \to [r_i]_f \end{array}\right\} \quad \text{for } i = 1, \ldots, m.$$

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

4. $$\left.\begin{array}{ll} x_{i,j,t}[\,]_t \to [x_{i,j-1}]_t, & \overline{x}_{i,j,t}[\,]_t \to [\overline{x}_{i,j-1}]_t \\ x_{i,j,f}[\,]_f \to [x_{i,j-1}]_f, & \overline{x}_{i,j,f}[\,]_f \to [\overline{x}_{i,j-1}]_f \\ r_{i,t}[\,]_t \to [r_i]_t, & r_{i,f}[\,]_f \to [r_i]_f \end{array}\right\} \quad \text{for} \quad \begin{array}{l} i = 1, \ldots, m \\ j = 2, \ldots, n. \end{array}$$

In order to analyse the next variable, the second subscripts 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 to true on the previous assignments.

5. $[k_s \to k_{s+1}]_l \qquad [k_2]_l \to \lambda \qquad \text{for } l = t, f;\ s = 0, 1.$

The objects k_i for $i = 0, 1, 2$ are counters that dissolve membranes when they are not useful any longer during the rest of the computation.

6. $[z_{n-1,t} \to [z_n]_t]_l, \qquad [z_{n-1,f} \to [z_n]_f]_l, \qquad [z_n \to d_1 \dots d_m q]_l \qquad \text{for } l = t, f.$

At the end of the generation stage, the objects $z_{n-1,l}$ create two new membranes where the formula will be evaluated on the two possible assignments for the last variable x_n. The object z_n is placed in both membranes, and will produce the objects $d_1, \dots, d_m, yes_0$, which will take part in the checking stage.

7. $$\left.\begin{array}{lll} [d_i \to [t_0]_i]_l & r_{i,t}[\,]_i \to [r_i]_i, & [r_i]_i \to \lambda \\ [t_s \to t_{s+1}]_i, & [t_2]_i \to t_3 & \end{array}\right\} \quad \text{for } \begin{array}{l} i = 1, \dots, m \\ s = 0, 1. \end{array}$$

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 to true on the assignment associated with the membrane; therefore neither does the formula evaluate to true nor its associated assignment.

8. $$\left.\begin{array}{lll} [q \to [yes_0]_a]_l & t_3[\,]_a \to [t_3]_a & [t_3]_a \to \lambda \\ [yes_h \to yes_{h+1}]_a, & [yes_5]_a \to yes_6 & [yes_6]_l \to yes_7[\,]_l \end{array}\right\} \quad \text{for } \begin{array}{l} l = t, f \\ h = 0, \dots, 4. \end{array}$$

The object q creates a membrane with label a where the object yes_0 is placed. The object yes_h evolves to the object yes_{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_6 from being sent out from this membrane.

9. $$\left.\begin{array}{ll} [no_p \to no_{p+1}]_a, & [no_{2n+10}]_a \to no_{2n+11} \\ yes_7[\,]_a \to [yes_8]_a, & [yes_8]_a \to yes_9 \\ [yes_9]_t \to yes[\,]_t & [no_{2n+11}]_t \to no[\,]_t \end{array}\right\} \quad \text{for } p = 0, \dots, 2n + 9.$$

From the beginning of the computation, the object no_p evolves to the object no_{p+1} inside the membrane labelled with a. If any object yes_7 is produced during the computation, it means that the formula evaluates to true on some assignment to its variables, and it gets into this membrane and dissolves it, producing the object yes_9 that will send out to the environment the object *yes*. On the other hand, if no object yes_7 appears in the skin, the object no_{2n+10} will dissolve the membrane labelled with a, producing the object no_{2n+11} that will send out to the environment the object *no*.

4.1. An overview of the computation

First of all, given a formula in CNF, $\varphi = C_1 \wedge \dots \wedge C_m$ such that $Var(\varphi) = \{x_1, \dots, x_n\}$, we define $s(\varphi) = \langle n, m \rangle$ and $cod(\varphi) = \{x_{i,j} \ : \ x_j \in C_i\} \cup \{\overline{x}_{i,j} \ : \ \neg x_{i,j} \in C_i\}$. Then (cod, s) is a pair of polynomial-time computable functions over I_{SAT} such that $s(\varphi)$ is a natural number and $cod(\varphi)$ is an input multiset over $\Pi(s(u))$. Next, we describe informally how the P system with $\Pi(s(\varphi))$ and with input $cod(\varphi)$ works.

In the initial configuration, we have, on the one hand, the input multiset $cod(\varphi)$ and the objects $z_{0,t}$ and $z_{0,f}$ placed in the skin (membrane labelled with t); and on the other hand, we have in the membrane labelled with a the object no_0. This object evolves during the computation following the first rule in the set 9.

In the first step of the computation, the object $z_{0,t}$ creates a new membrane with label t, which represents the assignment $x_1 = true$ and the object $z_{0,f}$ creates a new membrane with label f, which represents the assignment $x_1 = false$. In these two new membranes, the objects z_1 and k_0 are placed. At the same time, the input multiset representing the formula is duplicated following the two first rules in 2. In the next step, according to the rules in 3, the formula is evaluated on the two possible assignments for x_1. In the same step, the rules in 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 analyse the next variable. Moreover, at the same time, the object z_1 produces the object $z_{1,t}$ and $z_{1,f}$ and the system is ready to analyse the next variable. And so the generation and evaluation stages go until all the possible assignments to the variables are generated, and the formula is evaluated on each one of them. Observe that it takes two steps to generate the possible assignments for a variable and to evaluate the formula on them; therefore the generation and evaluation stages take $2n$

steps. Note that the object k_0 in the rules in 5 is a counter that dissolves the membrane when the object k_2 appears; that is, it dissolves the membrane once the membrane is not useful any longer in the rest of the computation.

The checking stage starts when the object z_n 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, which signify that the clause number i evaluates to true on the assignment associated with the membrane, are sent into the membranes by the last rule in 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 dissolve 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 7. If, for some i, there is no object r_i, which means that the clause i does not evaluate to true on the associated assignment, then the object t_2 will dissolve the membrane labelled with i, thus producing the object t_3 that will get into the membrane with label a where the object yes_h evolves following the rules in 8. The object t_3 dissolves the membrane, preventing the production of the object yes_6. Therefore the checking stage takes 6 steps.

Finally the output stage takes place according to the rules in 9. On the one hand, if some object yes_6 is present in any membrane (which represents that the formula evaluates to true on the assignment associated with this membrane) it is sent out to the skin being transformed into the object yes_7. In the next step, yes_7 gets into the membrane labelled with a, being transformed into yes_8; then it dissolves the membrane, producing the object yes_9. This dissolution prevents the object no_{2n+11} from being produced. And finally the object yes is sent out to the environment. On the other hand, if there is no object yes_6, then the membrane with label a is not dissolved, and thus the object no_{2n+11} is produced, and the object no is sent out to the environment. Observe that the output stage takes 5 steps if the answer is yes, and 6 steps if the answer is no.

5. Some formal details

In the previous section, we have presented a family $\mathbf{\Pi}$ of recogniser P systems which solve the SAT problem. For each Boolean formula, a P system $\Pi(\langle n, m\rangle)$ is constructed, where n is the number of variables and m is the number of clauses. First of all, observe that the evolution rules of $\Pi(s(\varphi))$ are defined in a recursive manner from φ, in particular from n and m. Let us list the necessary resources to construct $\Pi(s(\varphi))$:

- Size of the alphabet: $6nm + 5n + 4m + 32 \in \Theta(nm)$
- Initial number of membranes: $2 \in \Theta(1)$
- Initial number of objects: $3 \in \Theta(1)$
- Sum of the lengths of the rules: $86nm + 84n + 144m + 121 \in \Theta(nm)$.

Therefore a Turing machine working in polynomial time can build $\Pi(s(\varphi))$ from the formula φ.

Finally, we can prove, using a formal description of the computation, that the P system $\Pi(s(\varphi))$ with input $cod(\varphi)$ always halts and sends to the environment the object yes or no in the last step. The number of steps of such a P system is $2n + 11$ if the output is yes and $2n + 12$ if the output is no; therefore there exists a linear bound for the number of steps of the computation.

Hence, the family $\mathbf{\Pi}$ of recogniser P systems with membrane creation using dissolution rules solves the SAT problem in polynomial (actually, linear) time according to Definition 3. So, we have the following result:

Theorem 4. $SAT \in \mathbf{PMC}_{\mathcal{MC}}$.

Corollary 5. $\mathbf{NP} \cup \mathbf{co\text{-}NP} \subseteq \mathbf{PMC}_{\mathcal{MC}}$.

Proof. It suffices to remark that the SAT problem is **NP**-complete, SAT $\in \mathbf{PMC}_{\mathcal{MC}}$, and that this complexity class is closed under polynomial time reduction and under complement. □

Remark 6. Note that in the family of recogniser P systems given in Section 4, membrane creation rules are used to produce an exponential workspace where all possible assignments to the variables of the formula are generated, whereas the process of checking whether or not the formula evaluates to true on any of them is done using dissolution rules.

If we denote by $\mathcal{MC}_{+d}$ (respectively, $\mathcal{MC}_{-d}$) the class of recogniser P systems with membrane creation and with dissolution rules (respectively, without dissolution rules), then we have just proved that $\mathbf{NP} \cup \mathbf{co\text{-}NP} \subseteq \mathbf{PMC}_{\mathcal{MC}_{+d}}$.

In [3], the authors showed that the generation in polynomial time of an exponential workspace (number of membranes) using membrane creation rules, is not enough to solve **NP**-complete problems in polynomial time (unless $\mathbf{P} = \mathbf{NP}$). More precisely, the authors proved that $\mathbf{P} = \mathbf{PMC}_{\mathcal{MC}_{-d}}$.

Summing up, in the framework of recogniser P systems with membrane creation, we have proved the following: (a) the class of problems which can be solved in a polynomial time by a family of such P systems *without dissolution* is equal to class **P**; and (b) the class of problems which can be solved in a polynomial time by a family of such P systems *with dissolution* contains the class $\mathbf{NP} \cup \mathbf{co\text{-}NP}$.

6. Conclusions

Membrane computing is a new cross-disciplinary field of natural computing, which has reached an important success in its short life.

This paper deals with the study of *efficient solutions* to well-known computationally hard 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. It exploits membrane creation (a poorly studied variant) to solve **NP**-complete problems, giving the first uniform solution to **SAT** in polynomial time (recently, we have proved that this variant is **PSPACE** powerful [4]).

We stress the relevant role played by the rules of dissolution in the framework of recogniser P systems using membrane creation, in order to "separate" their tractability from the (presumable) intractability of decision problems, putting a barrier between the complexity classes **P** and **NP** (assuming $\mathbf{P} \neq \mathbf{NP}$).

This paper can be considered as a contribution to the interesting problem of characterising the tractability of hard decision problems in terms of the descriptional resources required in membrane systems.

Acknowledgements

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] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, Towards a programming language in cellular computing, Electronic Notes in Theoretical Computer Science 123 (2005) 93–110.

[2] 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.

[3] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero, Characterizing tractability with membrane creation, in: 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2005. Workshop on Theory and Applications of P Systems, Timisoara, Romania, 2005, IEEE Computer Society, 2005, pp. 448–457.

[4] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero, A linear time solution for QSAT with membrane creation, in: R. Freund, G. Lojka, M. Oswald, Gh. Paun (Eds.), Pre-Proceedings of the Sixth International Workshop on Membrane Computing, WMC6, Vienna University of Technology, Vienna, Austria, 2005, pp. 395–409.

[5] P.L. Luisi, The chemical implementation of autopoiesis, in: G.R. Fleishaker, S. Colonna, P.L. Luisi (Eds.), Self-Production of Supramolecular Structures, Kluwer, Dordrecht, 1994.

[6] Gh. Păun, Membrane Computing. An Introduction, Springer-Verlag, Berlín, 2002.

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

[8] M.J. Pérez-Jiménez, A. Riscos-Núñez, A linear solution for the Knapsack problem using active membranes, Lecture Notes in Computer Science 2933 (2004) 250–268.

[9] M.J. Pérez-Jiménez, F.J. Romero-Campero, Solving the BIN PACKING problem by recognizer P systems with active membranes, in: Gh. Păun, A. Riscos, A. Romero, F. Sancho (Eds.), Proceedings of the Second Brainstorming Week on Membrane Computing, Report RGNC 01/04, University of Seville, Seville, Spain, 2004, pp. 414–430.

[10] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, A polynomial complexity class in P systems using membrane division, in: E. Csuhaj-Varjú, C. Kintala, D. Wotschke, Gy. Vaszyl (Eds.), Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003, 2003, pp. 284–294.

[11] A. Păun, Gh. Păun, The power of communication: P systems with symport/antiport, New Generation Computing 20 (3) (2002) 295–305.
[12] Gh. Păun, Computing with membranes, Journal of Computer and System Sciences 61 (1) (2000) 108–143.
[13] Gh. Păun, Further open problems in membrane computing, in: Gh. Păun, A. Riscos, A. Romero, F. Sancho (Eds.), Proceedings of the Second Brainstorming Week on Membrane Computing, Report RGNC 01/04, University of Seville, Seville, Spain, 2004, pp. 354–365.
[14] ISI web page. http://esi-topics.com/erf/october2003.html.
[15] P systems web page. http://psystems.disco.unimib.it/.