Computational efficiency of dissolution rules in membrane systems

MIGUEL A. GUTIÉRREZ-NARANJO, MARIO J. PÉREZ-JIMÉNEZ*, AGUSTÍN RISCOS-NÚÑEZ and FRANCISCO J. ROMERO-CAMPERO

Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Seville, Avda. Reina Mercedes s/n, 41012 Seville, Spain

Trading (in polynomial time) space for time in the framework of membrane systems is not sufficient to efficiently solve computationally hard problems. On the one hand, an exponential number of objects generated in polynomial time is not sufficient to solve **NP**-complete problems in polynomial time. On the other hand, when an exponential number of membranes is created and used as workspace, the situation is very different. Two operations in P systems (membrane division and membrane creation) capable of constructing an exponential number of membranes in linear time are studied in this paper. **NP**-complete problems can be solved in polynomial time using P systems with active membranes and with polarizations, but when electrical charges are not used, then dissolution rules turn out to be very important. We show that in the framework of P systems with active membranes but without polarizations and in the framework of P systems with membrane creation, dissolution rules play a crucial role from the computational efficiency point of view.

Keywords: Computational efficiency; Dissolution rules; Membrane systems

1. Introduction

Computational Complexity theory is a central area of research within computer science. One of the main goals of this theory is to study the efficiency of algorithms and their data structures through the analysis of the lower bounds for the amount of resources required for every algorithm that solves the problem. This theory describes a borderline between problems that can be efficiently solved and those that can only be solved with an unrealistic amount of resources, and it provides a classification of *abstract problems* allowing us to detect their inherent complexity from the algorithmic point of view. Such a classification requires a precise and formal definition of the concept of *abstract problem* and the model to be considered. In order to specify a *complexity class* within a general computational framework, the following parameters are required: (a) the *model* of computation, D (in our case, recognizer P systems); (b) the *mode* of computation, M (in our case, non-deterministic and parallel); (c) the resource or complexity measure, r, that we wish to bound (usually time and/or space); and (d) an upper *bound* of the resources, f (a total recursive function from natural numbers to natural numbers).

Then, a complexity class is defined as the set of all languages decided by the device D operating in mode M and such that for any input string u, D expends at most $f(|u|)$ units of a resource r, to accept or reject u.

Membrane Computing is inspired by the structure and functioning of living cells, and it provides a new non-deterministic model of computation which 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*, and they consist 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.

In this paper we work with P systems capable of constructing an exponential workspace (expressed by the number of membranes) in polynomial (often linear) time. These models abstract the way of obtaining new membranes through the processes of *mitosis* (membrane division) and *autopoiesis* (membrane creation).

P systems with active membranes have been successfully used to design (uniform) solutions to well-known **NP**-complete problems, such as SAT [1], *Subset Sum* [2], *Knapsack* [3], *Bin Packing* [4], *Partition* [5], and the *Common Algorithmic Problem* [6].

Recently, the first (*uniform*) results related to computational efficiency using membrane creation have arisen; for instance, giving polynomial time solutions to the following **NP**-complete problems: SAT [7], *Subset Sum* [8], and QSAT [9].

The present paper is a contribution to the problem of describing a borderline between tractability and intractability in terms of descriptional resources required in (recognizer) membrane systems.

The paper is organized as follows. In the next section, some preliminary ideas about recognizer membrane systems and polynomial complexity classes are introduced. Next, we study the computational efficiency of recognizer P systems with active membranes and without polarizations (in section 3), and the computational efficiency of recognizer P systems with membrane creation (in section 4), stressing the crucial role played by dissolution rules. Conclusions and some final remarks are given in section 5.

2. Recognizer membrane systems

Membrane Computing is a vivid research area initiated in 1998 by Păun [10] and nominated by the Institute for Scientific Information as a *Fast Emerging Research Front* in Computer Science, in October 2003 [11]. The initial goal was to abstract computing models from the structure and functioning of a cell, as well as from the organization of cells into tissues, organs and other higher-order structures.

A large variety of cell-like computing models, called P systems, were considered in this framework, based on the fundamental concept of biological membranes; the respective models are distributed (compartmentalized) parallel computing devices, processing multisets of abstract objects by means of various types of evolution rules. Parallelism, communication, non-determinism, synchronization, dynamic architecture of the model, etc. are central concepts of the theory, with biological, mathematical, and computer science sources of inspiration.

The main *syntactic* ingredients of a P system are a *membrane structure* (consisting of several membranes arranged hierarchically inside a main membrane, the *skin*, and delimiting *regions*), *multisets* of objects (corresponding to chemical substances present in the compartments of a cell), and *evolution rules* (corresponding to chemical reactions that can take place inside the cell).

The *semantics* of P systems is defined through a non-deterministic and synchronous model (a global clock is assumed). A *configuration* of a P 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. We get *transitions* from one configuration of the system to the next by applying the evolution rules to the objects placed inside the regions, in a non-deterministic, maximally parallel manner (in each region, all objects that can evolve must do so). 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 that reaches a configuration where no more rules can be applied to the existing objects and membranes 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.

In this paper we use membrane computing as a framework to address the resolution of decision problems, that is, problems that require either a *yes* or a *no* answer. More precisely, a decision problem X is a pair (I_X, θ_X) where I_X is a (countable) language over a finite alphabet (the elements are called *instances*) and θ_X is a predicate (a total boolean function) over I_X.

There exists a natural correspondence between languages and decision problems in such a way that to *solve* a decision problem is equivalent to *recognizing* the associated language: each language L over an alphabet Σ has a decision problem X_L associated with it as follows: $I_{X_L} = \Sigma^*$ and $\theta_{X_L} = \{(x, 1) : x \in L\} \cup \{(x, 0) : x \notin L\}$; reciprocally, given a decision problem $X = (I_X, \theta_X)$, the language L_X over the alphabet of I_X corresponding to it is defined as $L_X = \{a \in I_X : \theta_X(a) = 1\}$.

We consider P systems as *recognizer language* devices.

DEFINITION 2.1 *A recognizer P system is a P system with external output* (*that is*, *the results of halting computations are encoded in the environment*) *such that*:

(i) *the working alphabet contains two distinguished elements* `yes` *and* `no`;
(ii) *all computations halt*; *and*
(iii) *if 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 is an *accepting computation* (respectively, *rejecting computation*) if the object `yes` (respectively, `no`) appears in the environment associated with the corresponding halting configuration.

In order to ensure that a family of recognizer P systems solves a decision problem, two main properties must be proved. For each instance of the problem: (a) if *there exists an* accepting computation of the membrane system processing it answering *yes*, then the problem also answers *yes* for that instance (*soundness*); and (b) if the problem answers *yes*, then *any* computation of the system processing that instance answers *yes* (*completeness*).

If we require the family of membrane systems to be sound and complete, then the following *confluence* condition is satisfied: every computation of a system has the same output.

2.1 *Solving problems by recognizer P systems without input*

We formalize the previous ideas in the framework of recognizer P systems without input.

DEFINITION 2.2 *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(u))_{u \in I_X}$ be a (countable) family of recognizer membrane systems without input.*

- *We say that the family $\mathbf{\Pi}$ is sound with regard to X if for each instance of the problem $u \in I_X$ such that there exists an accepting computation of $\Pi(u)$, we have $\theta_X(u) = 1$.*
- *We say that the family $\mathbf{\Pi}$ is complete with regard to X if for each instance of the problem $u \in I_X$ such that $\theta_X(u) = 1$, every computation of $\Pi(u)$ is an accepting computation.*

The soundness property means that if we obtain an *acceptance response* of the system (associated with an instance) through *some* computation, then the answer to the problem (for that instance) is *yes*. The completeness property means that if we obtain an *affirmative* response to the problem, then *any* computation of the system must be an accepting one.

The first results concerning the *solvability* of **NP**-complete problems in polynomial time (even linear) by membrane systems were given by Păun [12, 13], Zandron *et al.* [14], Krishna and Rama [15], and Obtulowicz [16], in the framework of P systems without an input membrane. Thus, the constructive proofs of such results need to design *one* system for *each* instance of the problem. We say that these solutions are *semi-uniform solutions*, and this can be formally defined as follows.

DEFINITION 2.3 *Let $X = (I_X, \theta_X)$ be a decision problem. We say that X is solvable in polynomial time by a (countable) family of recognizer membrane systems without input $\mathbf{\Pi} = (\Pi(u))_{u \in I_X}$, and we denote it by $X \in \mathbf{PMC}^*_{\mathcal{R}}$ if the following are true:*

- *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 $\Pi(u)$ from the instance $u \in I_X$;*
- *the family $\mathbf{\Pi}$ is polynomially bounded; that is, there exists a polynomial function $p(n)$ such that, for each $u \in I_X$, all computations of $\Pi(u)$ halt in, at most, $p(|u|)$ steps; and*
- *the family $\mathbf{\Pi}$ is sound and complete with regard to X.*

We say that the family $\mathbf{\Pi}$ is a *semi-uniform solution* of the problem X.

Note that in order to decide if a decision problem X belongs to the complexity class $\mathbf{PMC}^*_{\mathcal{R}}$ two different tasks are considered: the first is the construction of the family, which we require to be done in polynomial time (sequential time by deterministic Turing machines). The second is the execution of the systems of the family, in which we imposed that the total number of steps performed by their computations are bounded by the function g (parallel time by non-deterministic membrane systems).

A direct consequence of working with recognizer membrane systems is that the complexity classes $\mathbf{PMC}^*_{\mathcal{R}}$ are closed under complement. Moreover, these complexity classes are closed under polynomial time reduction, in the classical sense [17].

2.2 *Solving problems by recognizer P systems with input*

In this section we deal with recognizer membrane systems *with input membrane* solving decision problems in a *uniform* way in the following sense: all instances of a decision problem with the same *size* (according to a previously fixed polynomial time computable criterion) are processed by the same system, on which an appropriate input, representing the specific instance, is supplied.

DEFINITION 2.4 *A P system with input membrane is a tuple (Π, Σ, i_Π), where: (a) Π is a P system with working alphabet Γ, with p membranes labelled with $1, \ldots, p$, and initial* *multisets* $\mathcal{M}_1, \ldots, \mathcal{M}_p$ *associated with them*; (b) Σ *is an* (*input*) *alphabet strictly contained in* Γ *and the initial multisets are over* $\Gamma - \Sigma$; *and* (c) i_Π *is the label of a distinguished* (*input*) *membrane.*

The computations of a P system with input membrane over a multiset are defined in a natural way, the only difference is that the initial configuration of (Π, Σ, i_Π) must be the initial configuration of Π in which the input multiset is added in the membrane labelled by i_Π. More formally,

Definition 2.5 *Let* (Π, Σ, i_Π) *be a* P *system with input membrane. Let* Γ *be the working alphabet of* Π, μ *the membrane structure, and* $\mathcal{M}_1, \ldots, \mathcal{M}_p$ *the initial multisets of* Π. *Let* m *be a multiset over the input alphabet* Σ. *The initial configuration of* (Π, Σ, i_Π) *with input* m *is* $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots, \mathcal{M}_p)$.

Definition 2.6 *Let* L *be a language, and* $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ *a family of* P *systems. A polynomial encoding from* L *to* $\mathbf{\Pi}$ *is a pair* (cod, s) *of polynomial-time computable functions whose domain is* L, *and for each* $u \in L$, $s(u)$ *is a natural number and* $cod(u)$ *is an input multiset of* $\Pi(s(u))$.

Polynomial encodings are stable under polynomial time reductions [1]. More precisely, the following proposition holds.

Proposition 2.7 *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* $\mathbf{\Pi}$.

The concepts of soundness and completeness can be extended in a natural way to families of recognizer P systems with input membrane via polynomial encodings.

Definition 2.8 *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 for each instance of the problem* $u \in I_X$ *such that there exists an accepting computation of* $\Pi(s(u))$ *with input* $cod(u)$, *we have* $\theta_X(u) = 1$.
- *We say that the family* $\mathbf{\Pi}$ *is complete with regard to* (X, cod, s) *if for each instance of the problem* $u \in I_X$ *such that* $\theta_X(u) = 1$, *we have that every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting one.*

Next, we solve a decision problem through a family of recognizer P systems constructed in polynomial time by a Turing machine, such that each element of the family processes all the instances of *same size*, in some sense. We say that these solutions are *uniform solutions*.

Definition 2.9 *Let* $X = (I_X, \theta_X)$ *be a decision problem. We say that* X *is solvable in polynomial time by a family of recognizer membrane systems with input membrane* $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$, *and we denote it by* $X \in \mathbf{PMC}_{\mathcal{R}}$ *if the following is true.*

- *The family* $\mathbf{\Pi}$ *is polynomially uniform by Turing machines*; *that is, there exists a deterministic Turing machine that constructs in polynomial time the system* $\Pi(n)$ *from* $n \in \mathbf{N}$.
- *There exists a polynomial encoding* (cod, s) *of* X *in* $\mathbf{\Pi}$ *such that*:
  - *the family* $\mathbf{\Pi}$ *is polynomially bounded with regard to* (X, cod, s); *that is, there exists a polynomial function* $p(n)$ *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*; *and*

– *the family* $\mathbf{\Pi}$ *is sound and complete with regard to* (X, cod, s).

We say that the family $\mathbf{\Pi}$ is a *uniform solution* to the problem X.

Note that in order to decide if a decision problem X belongs to the complexity class $\mathbf{PMC}_{\mathcal{R}}$, for each instance u of X, first of all the natural number $s(u)$ and the input multiset $cod(u)$ must be computed, and, finally, $\Pi(s(u))$ is constructed. This is properly a *precomputation stage*, running in polynomial time expressed by a number of *sequential steps* in the classical framework of Turing machines. We then execute the system $\Pi(s(u))$ with input $cod(u)$. This is properly the *computation stage*, also running in polynomial time, but now it is described by a number of *parallel steps* in the framework of membrane computing.

The complexity classes $\mathbf{PMC}_{\mathcal{R}}$ are closed under complement and closed under polynomial time reduction in the classical sense [17].

Remark 1 Let us note that if $\mathbf{\Pi}$ is a family of recognizer P systems solving a decision problem X in polynomial time and in a *uniform* way, then it provides a polynomial time solution of X in a *semi-uniform* way. That is, we have $\mathbf{PMC}_{\mathcal{R}} \subseteq \mathbf{PMC}^{*}_{\mathcal{R}}$.

3. Recognizer P systems with active membranes and without polarizations

A particularly interesting class of membrane systems are the systems with active membranes and with three electrical charges [12], where membrane division can be used in order to solve computationally hard problems in polynomial or even linear time by a space–time trade-off.

The first efficient *semi-uniform solution* to SAT was given by Păun [12] using division for non-elementary membranes. This result was improved by Păun *et al.* [18] using only division for elementary membranes (in that paper, a semi-uniform solution to *HPP* using membrane creation is also presented).

Sosik [19] provides an efficient *semi-uniform solution* to *QSAT* (quantified satisfiability problem), a well-known **PSPACE**-complete problem, in the framework of P systems with active membranes but using cell division rules for non-elementary membranes.

Different efficient *uniform solutions* have been obtained in the framework of recognizer P systems with active membranes, with polarizations and using division rules for elementary membranes: (a) some weakly **NP**-complete problems solvable in polynomial time in that framework are the following: *Knapsack* [3], *Subset Sum* [2], *Partition* [5]; and (b) some strongly **NP**-complete problems solvable in polynomial time are the following: SAT [20], *Clique* [21], *Bin Packing* [4], *Common Algorithmic Problem* [6].

The polynomial complexity class associated with recognizer P systems with active membranes and with three electrical charges does not seems precise enough to describe classical complexity classes below **NP**. Therefore, it is challenging to investigate weaker variants of cell-like membrane systems able to characterize classical complexity classes.

In this paper we work with a variant of these membrane systems that does not use polarizations.

Definition 3.1 *A* P *system with active membranes and without polarizations is a* P *system with* Γ *as working alphabet, with* H *as the finite set of labels for membranes, and where the rules are of the following forms*:

(a) $[\,a \rightarrow u\,]_h$ *for* $h \in H$, $a \in \Gamma$, $u \in \Gamma^*$. *These are object evolution rules. An object* $a \in \Gamma$ *belonging to a membrane labelled by* h *evolves to a string* $u \in \Gamma^*$.

(b) $a\,[\;]_h \rightarrow [\,b\,]_h$ *for* $h \in H,\ a, b \in \Gamma$. *These are send-in communication rules. An object from the region immediately outside a membrane labelled by h is introduced in this membrane, possibly transformed into another object.*
(c) $[\,a\,]_h \rightarrow b\,[\;]_h$ *for* $h \in H,\ a, b \in \Gamma$. *These are send-out communication rules. An object is sent out from the membrane labelled by h to the region immediately outside, possibly transformed into another object.*
(d) $[\,a\,]_h \rightarrow b$ *for* $h \in H,\ a, b \in \Gamma$. *These are dissolution rules. A membrane labelled by h is dissolved in reaction with an object. The skin is never dissolved.*
(e) $[\,a\,]_h \rightarrow [\,b\,]_h\,[\,c\,]_h$ *for* $h \in H,\ a, b, c \in \Gamma$. *These are division rules for elementary membranes. An elementary membrane can be divided into two membranes with the same label, possibly transforming some objects.*

These rules are applied according to the following principles.

- All the rules are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non-deterministic way), but any object which can evolve by one rule of any form must evolve.
- If at the same time a membrane labelled with h is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then we assume that first the evolution rules of type (a) are used, and then the division is produced. Of course, this process takes only one step.
- The rules associated with membranes labelled with h are used for all copies of this membrane. At one step, a membrane can be the subject of *only one* rule of types (b)–(e).

Instead of rules of type (e) we can consider *division rules for non-elementary membranes*, that is, rules of the form $[\,[\;]_{h_1}[\;]_{h_2}\,]_{h_0} \rightarrow [\,[\;]_{h_1}\,]_{h_0}\,[\,[\;]_{h_2}\,]_{h_0}$, where h_0, h_1, h_2 are labels: if the membrane with label h_0 contains other membranes than those with labels h_1, h_2, these membranes and their contents are duplicated and placed in both new copies of the membrane h_0; all membranes and objects placed inside membranes h_1, h_2, as well as the objects from membrane h_0 placed outside membranes h_1 and h_2, are reproduced in the new copies of membrane h_0.

We denote by $\mathcal{AM}^0(\alpha, \beta)$, where $\alpha \in \{-d, +d\}$ and $\beta \in \{-ne, +ne\}$, the class of all recognizer P systems with active membranes without polarization such that:

- if $\alpha = +d$ (resp. $\alpha = -d$), then dissolution rules are permitted (resp. forbidden);
- if $\beta = +ne$ (resp. $\beta = -ne$), then division rules for elementary and non-elementary (resp. only division rules for elementary) membranes are permitted.

Proposition 3.2 *For each* $\alpha \in \{-d, +d\}$ *and* $\beta \in \{-ne, +ne\}$ *we have*:

- $\mathbf{PMC}_{\mathcal{AM}^0(\alpha,\beta)} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^0(\alpha,\beta)}$;
- $\mathbf{PMC}_{\mathcal{AM}^0(\alpha,-ne)} \subseteq \mathbf{PMC}_{\mathcal{AM}^0(\alpha,+ne)}$;
- $\mathbf{PMC}^*_{\mathcal{AM}^0(\alpha,-ne)} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^0(\alpha,+ne)}$;
- $\mathbf{PMC}_{\mathcal{AM}^0(-d,\beta)} \subseteq \mathbf{PMC}_{\mathcal{AM}^0(+d,\beta)}$;
- $\mathbf{PMC}^*_{\mathcal{AM}^0(-d,\beta)} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^0(+d,\beta)}$.

Proof Every (polynomial time) uniform solution to a decision problem provides a (polynomial time) semi-uniform solution to the problem. ■

3.1 *Forbidding dissolution rules*

Let $\Pi \in \mathcal{AM}^0(-d, \beta)$ be a recognizer P system with active membranes without polarizations and not using dissolution rules, with $\beta \in \{-ne, +ne\}$. Each rule of Π can be considered as a *dependency relation* between the object triggering the rule and the object or objects produced by its application.

We can consider a general pattern for rules of types (a), (b), (c) and (e) in the form $(a, h) \rightarrow (a_1, h')(a_2, h') \ldots (a_s, h')$, where:

- the rules of type (a) correspond to the case $h = h'$ and $s \geq 1$;
- the rules of type (b) correspond to the case $h = f(h')$ and $s = 1$;
- the rules of type (c) correspond to the case $h' = f(h)$ and $s = 1$;
- the rules of type (e) correspond to the case $h = h'$ and $s = 2$.

If h is the label of a membrane, then $f(h)$ denotes the label of the father of the membrane labelled with h. We adopt the convention that the father of the skin membrane is the environment (and we denote by *env* the label associated with the environment of the system).

For example, let us consider a general rule $(a, h) \rightarrow (a_1, h')(a_2, h') \ldots (a_s, h')$. Then we can interpret that as: from the object a in the membrane labelled by h we can *reach* the objects $a_1, \ldots, a_s$ in the membrane labelled by h'.

The *dependency graph associated with* Π is the directed graph $G_\Pi = (V_\Pi, E_\Pi)$ whose nodes are the pairs $(a, h) \in \Gamma \times (H \cup \{env\})$, Γ being the working alphabet of Π and H the set of labels of Π, such that the object a in the membrane labelled by h either triggers a rule or it is produced by a rule, and $((a, h), (a', h'))$ is an arc if there exists a rule r of Π such that the object a in the membrane labelled by h produces the object a' in the membrane labelled by h' by the application of rule r. More precisely, the set of vertices of G_Π is $V_\Pi = VL_\Pi \cup VR_\Pi$, where

$$\begin{aligned}
VL_\Pi = \{(a, h) \in \Gamma \times H :& \exists u \in \Gamma^*([a \rightarrow u]_h \in R) \vee \\
& \exists b \in \Gamma([a]_h \rightarrow [\ ]_h b \in R) \vee \\
& \exists b \in \Gamma \exists h' = ch(h)(a[\ ]_{h'} \rightarrow [b]_{h'} \in R) \vee \\
& \exists b, c \in \Gamma([a]_h \rightarrow [b]_h[c]_h \in R)\}, \\
VR_\Pi = \{(b, h) \in \Gamma \times H :& \exists a \in \Gamma \exists u \in \Gamma^*([a \rightarrow u]_h \in R \wedge b \in \text{alph}(u)) \vee \\
& \exists a \in \Gamma \exists h' = ch(h)([a]_{h'} \rightarrow [\ ]_{h'} b \in R) \vee \\
& \exists a \in \Gamma(a[\ ]_h \rightarrow [b]_h \in R) \vee \\
& \exists a, c \in \Gamma([a]_h \rightarrow [b]_h[c]_h \in R)\}.
\end{aligned}$$

The set of arcs of G_Π is

$$\begin{aligned}
E_\Pi = \{((a, h), (b, h')) :& \exists u \in \Gamma^*([a \rightarrow u]_h \in R \wedge b \in \text{alph}(u) \wedge h = h') \vee \\
& ([a]_h \rightarrow [\ ]_h b \in R \wedge\ h' = f(h)) \vee \\
& (a[\ ]_{h'} \rightarrow [b]_{h'} \in R \wedge h = f(h')) \vee \\
& \exists c \in \Gamma([a]_h \rightarrow [b]_h[c]_h \in R \wedge h = h')\}.
\end{aligned}$$

It is easy to prove that there exists a deterministic Turing machine that constructs the dependency graph G_Π associated with Π 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 [22].

Let Δ_Π be the set whose elements are the pairs $(a, h) \in \Gamma \times (H \cup \{env\})$ such that there exists a path (within the dependency graph) from (a, h) to $(\texttt{yes}, env)\}$. Having in mind that the *reachability problem* (see chapter 1 of [23]) can be solved by a search algorithm running in polynomial (quadratic) time, there exists a deterministic Turing machine that constructs the set Δ_Π in polynomial time [22].

Given a family $\{\Pi(n) : n \in \mathbf{N}\}$ of recognizer P systems with input membranes (not using dissolution rules) solving a decision problem (in a uniform way), an instance u of the problem will be accepted by the system if and only if there is an object in a membrane of the initial configuration of the system $\Pi(s(u))$ with input $cod(u)$ such that there exists a path on the associated dependency graph reaching the object `yes` in the environment.

THEOREM 3.3 *For each $\beta \in \{-ne, +ne\}$ we have $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0(-d,\beta)}$.*

Proof Let us assume that $\beta = -ne$ (the proof in the case $\beta = +ne$ is similar because division rules for non-elementary membranes do not provide any arc in the dependency graph).

Taking into account that the class $\mathbf{PMC}_{\mathcal{AM}^0(-d,-ne)}$ is closed under polynomial time reduction, it suffices to prove that $\mathbf{PMC}_{\mathcal{AM}^0(-d,-ne)} \subseteq \mathbf{P}$. Let $X \in \mathbf{PMC}_{\mathcal{AM}^0(-d,-ne)}$ and let $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$ be a family of recognizer P systems in $\mathcal{AM}^0(-d, -ne)$ solving X according to Definition 2.9. Let (cod, s) be the polynomial encoding associated with that solution.

Given an instance u of X, let

$$\Pi(s(u)) = (\Gamma(s(u)), H(s(u)), \mu, \mathcal{M}_1, \ldots, \mathcal{M}_{p(u)}, R(s(u)))$$

be the P system processing u whose input membrane is labelled by $i_{\Pi(s(u))}$. Then, we denote $(cod(u))^* = \{(a, i_{\Pi(s(u))}) : a \in \text{alph}(cod(u))\}$, and for each j $(1 \leq j \leq p(u))$ we denote $\mathcal{M}_j^* = \{(a, j) : a \in \mathcal{M}_j\}$.

We consider the following deterministic algorithm:

```
 Input: an instance $u$ of $X$
- Construct the system $\Pi(s(u))$ with input multiset $cod(u)$
- Construct the dependency graph $G_{\Pi(s(u))}$ associated with $\Pi(s(u))$
- Construct the set $\Delta_{\Pi(s(u))}$ associated with $\Pi(s(u))$
 $answer \leftarrow$ no; $j \leftarrow 1$
  while $j \leq p(u) \wedge answer =$ no do
     if $\Delta_{\Pi(s(u))} \cap \mathcal{M}_j^* \neq \emptyset$ then
       $answer \leftarrow$ yes
    $j \leftarrow j + 1$
  endwhile
  if $\Delta_{\Pi(s(u))} \cap (cod(u))^* \neq \emptyset$ then
    $answer \leftarrow$ yes
```

The algorithm described above solves the problem X. Indeed, on the one hand, the answer of this algorithm is `yes` if and only if there exists a pair $(a, h) \in \Delta_{\Pi(s(u))}$ such that the symbol a appears in the membrane labelled with h in the initial configuration (associated with the input multiset $cod(u)$). On the other hand, a pair (a, h) belongs to $\Delta_{\Pi(s(u))}$ if and only if there exists a path from (a, h) to $(\texttt{yes}, environment)$, that is, if and only if we can obtain an accepting computation of $\Pi(s(u))$ with input $cod(u)$.

The cost to determine whether or not $\Delta_{\Pi(s(u))} \cap \mathcal{M}_j^* \neq \emptyset$, for each j $(1 \leq j \leq p(u))$, or $\Delta_{\Pi(s(u))} \cap (cod(u))^* \neq \emptyset$, is of the order $O(|\Gamma(s(u))|^2 \cdot |H(s(u))|^2)$. Hence, the running time of this algorithm is bounded by

$$f(|u|) + O(|R(s(u))| \cdot q) + O(p(u) \cdot |\Gamma(s(u))|^2 \cdot |H(s(u))|^2),$$

where f is the (total) cost of a polynomial encoding from X to $\mathbf{\Pi}$ and $q = \max\{length(r) : r \in R(s(u))\}$. That is, the algorithm is polynomial in the size $|u|$ of the input. ■

Similar characterizations of **P** can be obtained when we deal with semi-uniform solutions in the framework of recognizer P systems with active membranes without polarizations and where dissolution rules are forbidden. The proofs are similar; it is sufficient to consider the system $\Pi(u)$ for each instance u of X, instead of the system $\Pi(s(u))$ with input the multiset $cod(u)$.

THEOREM 3.4 *For each $\beta \in \{-ne, +ne\}$ we have $\mathbf{P} = \mathbf{PMC}^*_{\mathcal{AM}^0(-d,\beta)}$.*

3.2 *Permitting dissolution rules*

In this section we show that the class of decision problems solvable in polynomial time in a semi-uniform way by families of recognizer P systems with active membranes, without polarization, where dissolution rules are permitted, and using division rules for elementary and non-elementary membranes, contains the standard complexity class **NP**. For that, we describe a family of such recognizer membrane systems which solves the Subset Sum problem in linear time and in a semi-uniform way. The Subset Sum problem is the following: *Given a finite set A, a weight function, $w : A \to \mathbf{N}$, and a constant $k \in \mathbf{N}$, determine whether or not there exists a subset $B \subseteq A$ such that $w(B) = k$.* We propose here a solution to this problem based on a brute force algorithm implemented in the framework of P systems with active membranes, without polarizations, with dissolution rules, and using division for elementary and non-elementary membranes.

The idea of the design is better understood if we divide the solution to the problem into several stages.

- *Generation stage.* For every subset of A, a membrane is generated via membrane division.
- *Weight calculation stage.* In each membrane the weight of the associated subset is calculated. This stage will take place in parallel with the previous one.
- *Checking stage.* For each membrane it is checked whether or not the weight of its associated subset is exactly k. This stage cannot start before the previous ones have been completed.
- *Output stage.* When the previous stage has been completed in all membranes, the system sends out the answer to the environment.

We will use a tuple $u = (n, (w_1, \dots, w_n), k)$ to represent an instance of the problem, where n stands for the size of $A = \{a_1, \dots, a_n\}$, $w_i = w(a_i)$, and k is the constant given as input for the problem. Associated with u, we consider the P system with active membranes, without polarization, without input membrane $\Pi(u) = (\Gamma(u), H(u), \mu, \mathcal{M}_0, \mathcal{M}_1, \dots, \mathcal{M}_{k+3}, R(u))$ defined as follows.

- The working alphabet $\Gamma(u)$ is the set

$$\{d_0, \dots, d_{2n+1}, a_1, \dots, a_n, e_1, \dots, e_n\} \cup \{b, s, c, \underline{c}, z_0, \dots, z_{2n+k+5}, \mathtt{yes}, \mathtt{no}\}.$$

- The set of labels $H(u)$ is $\{0, 1, 2, 3, \dots, k+3\}$.
- The initial membrane structure is $\mu = [\,[\,[\,[\,[\dots[\,[\,]_0\,]_1 \dots]_k\,]_{k+1}\,]_{k+2}\,]_{k+3}$.
- The initial multisets are $\mathcal{M}_0 = d_0$, $\mathcal{M}_{k+2} = z_0$ and $\mathcal{M}_i = \emptyset$, if $i \in \{1, \dots, k, k+1, k+3\}$.
- The set of evolution rules, $R(u)$, consists of the following:

(a)

$$\begin{aligned} &[d_{2i} \to a_{i+1}d_{2i+1}]_0, && \text{for } i \in \{0, \ldots, n\},\\ &[d_{2i+1} \to d_{2i+2}]_0, && \text{for } i \in \{0, \ldots, n-1\}. \end{aligned}$$

The goal of the counter d_i is to control the appearance of the object a_j only in the odd steps. The importance of these objects will be explained in the next set of rules.

(b)

$$\left.\begin{aligned} [a_i]_0 \to [e_i]_0\,[b]_0,\\ [e_i \to s^{w_i}]_0, \end{aligned}\right\} \quad \text{for } i \in \{1, \ldots, n\}.$$

The object a_i triggers the rule for division of elementary membranes. After the division, in one membrane is placed the object e_i, and in the other the object b. The object b remains inactive, whereas the object e_i evolves in the next step to as many objects s as the weight w_i.

(c)

$$[\,[\,]_i\,[\,]_i\,]_{i+1} \longrightarrow [\,[\,]_i\,]_{i+1}\,[\,[\,]_i\,]_{i+1}, \quad \text{for } i \in \{1, \ldots, k\}.$$

This is the set of rules for the division of non-elementary membranes. These three first sets of rules produce a membrane structure with 2^n branches. On each of the leaves of the tree we have a membrane with as many objects s as the weight of a possible subset, S, of A.

(d)

$$\begin{aligned} &[d_{2n+1}]_0 \to b,\\ &[s]_i \to c, \quad \text{for } i \in \{1, \ldots, k+1\}. \end{aligned}$$

When the generation stage has finished, the object d_{2n+1} dissolves the membrane with label 0. At this point, the elements s start to dissolve membranes. If there are enough objects s, all the membranes with labels $1, \ldots, k+1$ are dissolved. Otherwise, the branch remains inactive.

(e)

$$[c \to \underline{c}]_{k+1}.$$

This is a waiting step and the key of the computation. If in a branch the codified weight of the subset w_S is less than k, the membrane remains inactive. Otherwise, all the membranes of the branch are dissolved until reaching the membrane with label $k+1$. If $w_S = k$ in this membrane, there are no objects s that dissolve it, and the object $\underline{c}$ remains in the membrane. On the contrary, if $w_S > k$, the membrane is dissolved in the same step in which $\underline{c}$ is produced, and $\underline{c}$ goes to the membrane with label $k+2$.

(f)

$$\begin{aligned} &[z_i \to z_{i+1}]_{k+2}, \quad \text{for } i \in \{0, \ldots, 2n+k+4\},\\ &[\underline{c}]_{k+1} \to \texttt{yes},\\ &[yes]_{k+2} \to \texttt{yes},\\ &[z_{2n+k+5}]_{k+2} \to \texttt{no}. \end{aligned}$$

If one of the subsets of A has weight k, then an object $\underline{c}$ appears in a membrane with label $k+1$. This object dissolves the membrane and sends an object `yes` to the membrane with label $k+2$. In this membrane we keep a counter z_i along the computation. If an object $\underline{c}$ has sent an object yes to this membrane, this object will dissolve the membrane in the next step preventing the object z_{2n+k+5} remaining in the membrane. Otherwise, if the object $\underline{c}$ is never produced, we get an object z_{2n+k+5} in the membrane with label $k+2$. In the following step, this membrane is dissolved and an element no is sent to the membrane with label $k+3$.

(g)

$$[\texttt{no}]_{k+3} \longrightarrow \texttt{no}[\,]_{k+3},$$
$$[\texttt{yes}]_{k+3} \longrightarrow \texttt{yes}[\,]_{k+3}.$$

From the above, we know that the membrane with label $k+3$ is reached by one and only one of the objects yes and no. These rules send that object to the environment in the last step of the computation.

Then we have the following result.

PROPOSITION 3.5 *SubsetSum* $\in \mathbf{PMC}^*_{\mathcal{AM}^0(+d,+ne)}$.

COROLLARY 3.6 $\mathbf{NP} \cup \mathbf{co}\text{-}\mathbf{NP} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^0(+d,+ne)}$.

Proof It suffices to make the following observations: the Subset Sum problem is **NP**-complete, belonging to the class $\mathbf{PMC}^*_{\mathcal{AM}^0(+d,+ne)}$, and this class is closed under polynomial-time reduction and closed under complement. ■

4. P systems with membrane creation

One of the roles of biological membranes is to keep the molecules of a region close to each other, in order to facilitate their reactions. So, when a region becomes too large, new membranes can appear inside it during biological evolution.

In this section we consider a variant of P systems (slightly different from those originally introduced by Păun [24]), where new membranes are produced under the influence of existing objects.

DEFINITION 4.1 *A P system with membrane creation is a tuple of the form* $\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_p, R)$, *where*:

(i) $p \geq 1$ *is the initial degree of the system*; Γ *is the alphabet of objects and* H *is a finite set of labels for membranes*;
(ii) μ *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* Γ, *describing the multisets of objects placed in the* p *regions of* μ;
(iii) 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* Γ *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* Γ *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 with it 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 so.
- 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, irrespective of whether or not the membrane is an initial one or 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 where dissolution rules are forbidden (respectively, permitted).

PROPOSITION 4.2 *For each* $\alpha \in \{-d, +d\}$, $\mathbf{PMC}_{\mathcal{MC}(\alpha)} \subseteq \mathbf{PMC}^*_{\mathcal{MC}(\alpha)}$.

Proof Every (polynomial time) uniform solution to a decision problem provides a (polynomial time) semi-uniform solution to the problem. ■

4.1 *Forbidding dissolution rules*

Let Π be a recognizer P system with membrane creation where dissolution rules are forbidden. In a similar way as for membrane systems with active membranes, we can define the dependency graph of such a P system.

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

- 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 $s = 1$; and
- 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 $h, h' \in H$ the cost of determining whether or not $h' \in F(h)$ is of the order $O(|R| + p)$, p being 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 $(a, h) \rightarrow (a_1, h') \dots (a_s, h')$. This states that from object a in the membrane labelled by h we can *reach* the objects $a_1, \dots, a_s$ in the membrane labelled by h'.

The formal definition of the dependency graph of such P systems is similar to that given for P systems with active membranes without polarizations and not using dissolution rules.

It is easy to prove that there exists a deterministic Turing machine that constructs the dependency graph G_Π associated with Π in polynomial time [25].

The set Δ_Π whose elements are the pairs $(a, h) \in \Gamma \times (H \cup \{env\})$ such that there exists a path (within the dependency graph) from (a, h) to $(\mathtt{yes}, env)$ can be constructed by a deterministic Turing machine in polynomial time [25].

We show that, in the framework of recognizer P systems with membrane creation where dissolution rules are forbidden, if $\mathbf{P} \neq \mathbf{NP}$, then constructing an exponential workspace (number of membranes) in polynomial time is not sufficient to solve **NP**-complete problems in polynomial time.

THEOREM 4.3 $\mathbf{P} = \mathbf{PMC}_{\mathcal{MC}(-d)}$.

Proof Since the class $\mathbf{PMC}_{\mathcal{MC}(-d)}$ is closed under polynomial time reduction, it suffices to prove 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, in a uniform way. Let (cod, s) be the polynomial encoding associated with that solution.

Then, the algorithm described in the proof of theorem 3.3 solves problem X in polynomial time. ■

Similar characterizations of **P** can be obtained when we deal with semi-uniform solutions in the framework of recognizer P systems with membrane creation and without dissolution rules. The proofs are similar, and it is sufficient to consider the system $\Pi(u)$ for each instance u, instead of the system $\Pi(s(u))$ with input the multiset $cod(u)$. So, we have the following result.

THEOREM 4.4 $\mathbf{P} = \mathbf{PMC}^*_{\mathcal{MC}(-d)}$.

4.2 *Permitting dissolution rules*

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

Given a boolean formula $\varphi(x_1, \dots, x_n)$ in conjunctive normal form, with boolean variables $x_1, \dots, x_n$, the sentence $\varphi^* = \exists x_1 \forall x_2 \dots Q_n x_n \varphi(x_1, \dots, x_n)$ (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 $\varphi(x_1, \dots, x_n)$. Recall that a sentence is a boolean formula in which every variable is in the scope of a quantifier.

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

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

It is well known that QSAT is a **PSPACE**-complete problem [23].

The solution proposed in this section follows a brute force approach, in the framework of recognizer P systems with membrane creation where dissolution rules are permitted, and consists of the following stages.

- *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 has gates AND, OR. In this stage, the values of the formula corresponding to each assignment are 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 system sends out to the environment the right answer according to the result of the previous stage.

Let us consider the pair function from $\mathbf{N}^2$ onto $\mathbf{N}$ defined by

$$\langle n, m\rangle = \left(\frac{(n+m)(n+m+1)}{2}\right) + n.$$

This function is polynomial time computable and bijective.

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 φ^* associated with φ (when an appropriate input is supplied, $cod(\varphi^*) = \{x_{i,j} : x_j \in C_i\} \cup \{\bar{x}_{i,j} : \neg x_j \in C_i\}$).

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

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

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

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

defined as follows.

- The working alphabet, $\Gamma(\langle n, m\rangle)$, is

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

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

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

- The initial membrane structure is $\mu = [\,[\,]_{<t,\vee>}\,]_s$.
- The initial multisets are $\mathcal{M}_s = \emptyset$, $\mathcal{M}_{<t,\vee>} = \{z_{0,\wedge,t} z_{0,\wedge,f}\}$.
- The input membrane is $[\,]_{<t,\vee>}$.
- The set of evolution rules, $R(\langle n, m\rangle)$, consists of the following rules (recall that λ denotes the empty string, and if c is $\wedge$ then $\bar{c}$ is $\vee$, and if c is $\vee$ then $\bar{c}$ is $\wedge$):

1.

$$\left.\begin{array}{l} [z_{j,c} \rightarrow z_{j,c,t}, z_{j,c,f}]_{<l,\bar{c}>}, \\ [z_{j,c,l} \rightarrow [z_{j+1,\bar{c}}]_{<l,c>}]_{<l',\bar{c}>}, \end{array}\right\} \quad \begin{array}{l} \text{for } l, l' \in \{t, f\}, c \in \{\vee, \wedge\}, \\ j \in \{0, \dots, n-1\}. \end{array}$$

The goal of these rules is to create one membrane for each assignment to the variables of the formula. First, the object $z_{j,c}$ evolves to two objects, one for the assignment *true* (the object $z_{j,c,t}$), and the second 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.

$$\left.\begin{array}{l} [x_{i,j} \rightarrow x_{i,j,t} x_{i,j,f}]_{<l,c>}, \\ [\bar{x}_{i,j} \rightarrow \bar{x}_{i,j,t} \bar{x}_{i,j,f}]_{<l,c>}, \\ [r_i \rightarrow r_{i,t} r_{i,f}]_{<l,c>}, \end{array}\right\} \quad \begin{array}{l} \text{for } l \in \{t, f\}, i \in \{1, \dots, m\}, c \in \{\vee, \wedge\}, \\ j \in \{1, \dots, n\}. \end{array}$$

These rules duplicate the objects representing the formula so it can be evaluated on the two possible assignments, $x_j =$ *true* ($x_{i,j,t}$, $\bar{x}_{i,j,t}$) and $x_j =$*false* ($x_{i,j,f}$, $\bar{x}_{i,j,f}$). 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.

$$\left.\begin{array}{l} x_{i,1,t}[\,]_{<t,c>} \rightarrow [r_i]_{<t,c>} \bar{x}_{i,1,t}[\,]_{<t,c>} \rightarrow [\lambda]_{<t,c>}, \\ x_{i,1,f}[\,]_{<f,c>} \rightarrow [\lambda]_{<f,c>} \bar{x}_{i,1,f}[\,]_{<f,c>} \rightarrow [r_i]_{<f,c>}, \end{array}\right\} \quad \begin{array}{l} \text{for } i \in \{1, \dots, m\}, \\ c \in \{\vee, \wedge\}. \end{array}$$

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

4.

$$\left.\begin{array}{l} x_{i,j,l}[\,]_{<l,c>} \rightarrow [x_{i,j-1}]_{<l,c>}, \\ \bar{x}_{i,j,t}[\,]_{<l,c>} \rightarrow [\bar{x}_{i,j-1}]_{<l,c>}, \\ r_{i,t}[\,]_{<l,c>} \rightarrow [r_i]_{<l,c>}, \end{array}\right\} \quad \begin{array}{l} \text{for } l \in \{t, f\}, i \in \{1, \dots, m\}, c \in \{\vee, \wedge\}, \\ j \in \{2, \dots, n\}. \end{array}$$

In order to analyse the next variable, the second subscript of the objects $x_{i,j,l}$ and $\bar{x}_{i,j,l}$ 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.

$$[z_{n,c} \to d_1 \dots d_m q]_{<l,\bar{c}>}, \qquad \text{for } l \in \{t, f\}, c \in \{\vee, \wedge\}.$$

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

6.

$$\left.\begin{array}{l} [d_i \to [t_0]_i]_{<l,c>}, \\ r_{i,t}[\,]_i \to [r_i]_i [r_i]_i \to \lambda, \\ [t_j \to t_{j+1}]_i [t_2]_i \to t_3, \end{array}\right\} \quad \begin{array}{l} \text{for } i \in \{1, \dots, m\}, j \in \{0, 1\}, l \in \{t, f\}, \\ c \in \{\vee, \wedge\}. \end{array}$$

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 true on the assignment associated with the membrane; therefore, neither does the formula on the associated assignment.

7.

$$\left.\begin{array}{l} [q \to [ans_0]_a]_{<l,c>}, \\ t_3[\,]_a \to [t_4]_a [t_4]_a \to \lambda, \\ [ans_h \to ans_{h+1}]_a [ans_5]_a \to yes, \\ [ans_5 \to no]_{<l,c>}, \end{array}\right\} \quad \begin{array}{l} \text{for } l \in \{t, f\}, c \in \{\vee, \wedge\}, \\ h = 0, \dots, 4. \end{array}$$

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.

8.

$$\left.\begin{array}{l} [yes]_{<l,c>} \to yes_{\bar{c}} [no]_{<l,c>} \to no_{\bar{c}}, \\ [yes_\vee]_{<l,\vee>} \to yes^* \qquad [no_\vee \to \underline{no}_\vee]_{<l,\vee>}, \\ [yes^* \to yes_\wedge]_{<l,\wedge>} [\underline{no}_\vee]_{<l,\vee>} \to no_\wedge, \\ [\underline{no}_\vee \to \lambda]_{<l,\wedge>} [yes_\vee \to \lambda]_{<l,\wedge>}, \\ [no_\wedge]_{<l,\wedge>} \to no^* [yes_\wedge \to \underline{yes}_\wedge]_{<l,\wedge>}, \\ [no^* \to no_\vee]_{<l,\vee>} [\underline{yes}_\wedge]_{<l,\wedge>} \to yes_\vee, \\ [\underline{no}_\wedge \to \lambda]_{<l,\vee>} [\underline{yes}_\wedge \to \lambda]_{<l,\vee>}, \\ [yes^*]_s \to YES\,[\,]_s [no_\wedge]_s \to NO\,[\,]_s, \end{array}\right\} \quad \text{for } l \in \{t, f\}.$$

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 or not the assignment associated with this membrane satisfies the formula. Contrary to the SAT problem, in QSAT it is not sufficient 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.

This family $\mathbf{\Pi}$ of recognizer P systems with membrane creation and using dissolution rules solves in polynomial (actually, linear) time the QSAT problem in a uniform way. So, we have the following result.

THEOREM 4.5 $QSAT \in \mathbf{PMC}_{\mathcal{MC}(+d)}$.

COROLLARY 4.6 $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\mathcal{MC}(+d)}$.

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. ■

5. Conclusions

Assuming that $\mathbf{P} \neq \mathbf{NP}$, many interesting problems of the real world are intractable and hence it is not possible to execute algorithmic solutions in an electronic computer when we use instances of those problems whose size is huge. Nevertheless, we can trade space for time in order to provide polynomial time solutions to computationally hard problems.

In the framework of membrane systems it is possible to create an exponential workspace in polynomial (often linear) time in a natural manner. For instance, evolution rules of the form $[u \to v]$, with the length of v strictly greater than the length of u, allow the construction of an exponential number of objects in linear time, but it is well known that this condition is not sufficient to solve **NP**-complete problems. What happens when an exponential number of membranes is constructed and used as workspace?

Membrane division and membrane creation are two operations in P systems capable of obtaining an exponential number of membranes in linear time. In the case of P systems with active membranes, if we use three (or two) electrical charges, then **NP**-complete problems can be solved in polynomial time, but what happens when polarizations are not used?

In this paper we show that in both the framework of P systems with active membranes but without polarizations and in the framework of P systems with membrane creation, dissolution rules play a crucial role from the computational efficiency point of view. Specifically, if dissolution rules are forbidden, then it is not possible to solve **NP**-complete problems in polynomial time (unless $\mathbf{P} = \mathbf{NP}$). Nevertheless, if dissolution rules are permitted, then it is possible to efficiently solve computationally hard problems.

A conjecture known in the membrane computing area under the name of the P-conjecture affirms that $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0}$, where $\mathcal{AM}^0$ is the class of all recognizer P systems with active membranes and without polarizations.

In this paper we give a partial affirmative answer to the P-conjecture when dissolution rules are forbidden, and a partial negative answer to the P-conjecture is also given when we use semi-uniform solutions, dissolution rules, and membrane division rules for elementary and non-elementary membranes. For P systems with membrane creation, similar results are obtained.

Therefore, we show that dissolution rules provide a boundary between what can and cannot be solved algorithmically with realistic resource requirements, in the framework of P systems with active membranes without polarizations and in the framework of P systems with membrane creation.

Acknowledgements

The authors wish to acknowledge the support of the project TIN2005-09345-C04-01 of the Ministerio de Educación y Ciencia of Spain, co-financed by FEDER funds, and by a FPU fellowship to the fourth author from the Ministerio de Educación y Ciencia of Spain.

References

[1] Pérez-Jiménez, M.J., Romero-Jiménez, A. and Sancho-Caparrini, F., 2003, In: E. Csuhaj-Varjú, C. Kintala, D. Wotschke and G. Vaszil (Eds) *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003* (Budapest: Computer and Automation Research Institute of the Hungarian Academy of Sciences).

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

[3] Pérez-Jiménez, M.J. and Riscos-Núñez, A., 2004, A linear-time solution to the Knapsack problem using P systems with active membranes. In: C. Martín-Vide, Gh. Păun, G. Rozenberg and A. Salomaa (Eds) *Membrane Computing*. Lecture Notes in Computer Science 2933 (Berlin: Springer).

[4] Pérez-Jiménez, M.J. and Romero-Campero, F.J., 2004, In: Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez and F. Sancho-Caparrini (Eds) *Proceedings of the Second Brainstorming Week on Membrane Computing*. Report RGNC 01/04 (Seville: University of Seville).

[5] Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J. and Riscos-Núñez, A., 2005, A fast P system for finding a balanced 2-partition. *Soft Computing*, **9**, 673–678.

[6] Pérez-Jiménez, M.J. and Romero-Campero, F.J., 2005, In: M. Margenstern (Ed.) *Machines, Computations and Universality*. Lecture Notes in Computer Science 3354 (Berlin: Springer).

[7] Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J. and Romero-Campero, F.J., 2005, In: S. Barry Cooper, B. Lowe and L. Torenvliet (Eds) *CiE 2005: New Computational Paradigms, Amsterdam*. Report ILLC X-2005-01 (Amsterdam: University of Amsterdam).

[8] Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J. and Romero-Campero, F.J., 2005, In: J. Mira and J.R. Alvarez (Eds) *Mechanisms, Symbols and Models Underlying Cognition*. Lecture Notes in Computer Science 3561 (Berlin: Springer).

[9] Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J. and Romero-Campero, F.J., 2006, A linear time solution for QSAT with membrane creation. In: R. Freund, Gh. Păun, G. Rozenburg, and A. Salomaa (Eds) *Membrane computing*. Lecture Notes in Computer Science 3850 (Berlin: Springer).

[10] Păun, Gh., 2000, Computing with membranes. *Journal of Computer and System Sciences*, **61**, 108–143.

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

[12] Păun, Gh., 2001, P systems with active membranes: attacking **NP**-complete problems. *Journal of Automata, Languages and Combinatorics*, **6**, 75–90.

[13] Păun, Gh., 2000, In: I. Antoniou, C. Calude and M.J. Dinneen (Eds) *Unconventional Models of Computation* (London: Springer).

[14] Zandron, C., Ferreti, C. and Mauri, G., 2000, In: I. Antoniou, C. Calude and M.J. Dinneen (Eds) *Unconventional Models of Computation* (London: Springer).

[15] Krishna, S.N. and Rama, R., 1999, A variant of P systems with active membranes: solving NP-complete problems. *Romanian Journal of Information Science and Technology*, **2**, 357–367.

[16] Obtulowicz, A., 2001, Deterministic P systems for solving SAT problem. *Romanian Journal of Information Science and Technology*, **4**, 551–558.

[17] Pérez-Jiménez, M.J., 2005, In: G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg and A. Salomaa (Eds) *Membrane Computing*. Lecture Notes in Computer Science 3365 (Berlin: Springer).

[18] Păun, Gh., Suzuki, Y., Tanaka, H. and Yokomori, T., 2004, On the power of membrane division in P systems. *Theoretical Computer Science*, **324**, 61–85.

[19] Sosik, P., 2003, The computational power of cell division. *Natural Computing*, **2**, 287–298.

[20] Pérez-Jiménez, M.J., Romero-Jiménez, A. and Sancho-Caparrini, F., 2003, Complexity classes in cellular computing with membranes. *Natural Computing*, **2**, 265–285.

[21] Alhazov, A., Martín-Vide, C. and Pan, L., 2004, In: N. Jonoska, Gh. Păun and G. Rozenberg (Eds) *Aspects of Molecular Computing*. Lecture Notes in Computer Science 2950 (Berlin: Springer).

[22] Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A. and Romero-Campero, F.J., 2006, On the power of dissolution in P systems with active membranes. In: R. Freund, Gh. Păun, G. Rozenburg, and A. Salomaa (Eds) *Membrane computing*. Lecture Notes in Computer Science 3850 (Berlin: Springer).

[23] Papadimitriou, C.H., 1995, *Computational Complexity* (New York: Addison-Wesley).

[24] Păun, Gh., 2002, *Membrane Computing. An Introduction* (Berlin: Springer).

[25] Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A. and Romero-Campero, F.J., 2005, In: G. Ciobanu and Gh. Păun (Eds) *Proceedings of the First International Workshop on Theory and Applications of P Systems* (Timisoara, Romania: Research Institute e-Austria).