An apparently innocent problem in Membrane Computing

David Orellana-Martín, Luis Valencia-Cabrera,
Agustín Riscos-Núñez, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: {dorellana, lvalencia, ariscosn, marper}@us.es

Summary. The search for efficient solutions of computationally hard problems by means of families of membrane systems has lead to a wide and prosperous field of research. The study of computational complexity theory in *Membrane Computing* is mainly based on the look for frontiers of efficiency between different classes of membrane systems. Every frontier provides a powerful tool for tackling the **P** *versus* **NP** problem in the following way. Given two classes of recognizer membrane systems $\mathcal{R}_1$ and $\mathcal{R}_2$, being systems from $\mathcal{R}_1$ non-efficient (that is, capable of solving only problems from the class **P**) and systems from $\mathcal{R}_2$ presumably efficient (that is, capable of solving **NP**-complete problems), and $\mathcal{R}_2$ the same class that $\mathcal{R}_1$ with some ingredients added, passing from $\mathcal{R}_1$ to $\mathcal{R}_2$ is comparable to passing from the non efficiency to the presumed efficiency. In order to prove that $\mathbf{P} = \mathbf{NP}$, it would be enough to, given a solution of an **NP**-complete problem by means of a family of recognizer membrane systems from $\mathcal{R}_2$, try to remove the added ingredients to $\mathcal{R}_2$ from $\mathcal{R}_1$. In this paper, we study if it is possible to solve SAT by means of a family of recognizer P systems from $\mathcal{AM}^0(-d, +n)$, whose non-efficiency was demonstrated already.

Key words: Membrane Computing, polarizationless P systems with active membranes, cooperative rules, the **P** versus **NP** problem, SAT problem.

1 Introduction

Membrane Computing is a bio-inspired computing model based on the behavior and the structure of living cells. Introduced by Gh. Păun in 1998 [4], it has been used in a wide range of applications, and several variants have been developed depending on the field of study. From the beginning, the research of computational complexity issues from the perspective of membrane systems has been a prosperous field of study, with several papers written and interesting results found. The

first "solution" to an **NP**-complete problem, the SAT problem, in linear time is presented in [5]. We say "solution" since there was no definition of *solving a problem* by means of membrane systems. There was no definition until [8] where *recognizer* membrane systems (called *decision* membrane systems in the paper, and *accepting* membrane systems in a later paper), for a family of membrane systems of a certain type capable of *solving* a computational problem.

In the following years, for demonstrating the non-efficiency of membrane systems (that is, the capability of only solving problems from the class **P**), some tools were developed, as the *simulation technique* [7], the *algorithmic technique* [] and the *dependency graph technique* [1]. By using the former, in [1] it was demonstrated that P systems from $\mathbf{PMC}_{\mathcal{AM}^0(-d,+n)}$ were capable of solving only problems from the class **P**. As complexity classes $\mathbf{PMC}_{\mathcal{R}}$, being $\mathcal{R}$ a class of recognizer membrane systems, was demonstrated to be closed under polynomial reduction [], finding an efficient solution to *any* **NP**-complete problem by means of a family of P systems from $\mathcal{AM}^0(-d,+n)$ would lead to a negative answer to the $\mathbf{P} \neq \mathbf{NP}$ conjecture; that is, it solving an **NP**-complete problem in this framework leads to $\mathbf{P} = \mathbf{NP}$. It seems interesting then trying to find a solution based on the most common techniques while solving a computationally hard problem.

From here, the paper is organized as follows: in the next section, a brief view to the general structure of techniques to solve **NP**-complete problems by membrane systems is given. Section 3 is devoted to present a "solution" to the SAT problem, such that it depends of the existence of some *special machines*. These machines are detailed in the following two sections, explaining the structure in Section 4 and the behavior in Section 5. After that, in Section 6, the three kinds of special machines introduced are *reduced* to a single one, capable of solving each of the problems of the previous machines. Last, the main result is presented in Section 7. The paper ends with some conclusions and interesting open research lines.

2 Solutions to NP-complete problems

In the framework of Membrane Computing, several efficient solutions to computationally hard problems have been provided by means of a family of membrane systems; that is, they are solutions that run in polynomial time with respect of the size of the input. Usually, this is done by interchanging time and space, in the sense that we need to create an exponential workspace in terms of membranes or cells in the computation in order to obtain all the possible alternatives to solve the instance, and taking advantage of the inherent parallelism of membrane systems to check them at the same time. For this purpose, a family of membrane systems must be defined, each of its systems solving a subset of all the instances of the problem. Usually, the protocol to solve computationally hard problems is the following one:

1. *Generation stage*: In this stage, using division rules [6], separation rules [3] or membrane creation rules [2], among others, we can obtain an exponential workspace in terms of membranes or cells in polynomial (or even linear time).
2. *Checking stage*: In this stage, the presumed solutions in the previous stage are checked in order to know if any of them is a real solution of the instance.
3. *Output stage*: This stage consists in sending an object `yes` or an object `no` to the environment depending on the solvability or not of the instance.

In this sense, an interesting work for the reader is [9], where solutions are analyzed by decomposing the solutions in subroutines.

3 A "solution" to the SAT problem without using dissolution

Here we provide a solution to the `SAT` problem by means of a family of recognizer P systems with active membranes $\mathbf{\Pi} = \{\Pi(t) \mid t \in \mathbb{N}\}$ from $\mathcal{AM}^0(-d, +n)$ with a special mechanism whose behavior will be explained later. Given a Boolean formula φ in CNF and simplified with n variables and p clauses, the system $\Pi(s(\varphi)) + cod(\varphi)$ processes it, being $s(\varphi) = \langle n, p\rangle = \frac{(n+p)(n+p+1)}{2} + n$ and $cod(\varphi) = \{x_{i,j,0} \mid x_i \in C_j\} \cup \{\overline{x}_{i,j,0} \mid \neg x_i \in C_j\}$.

For each $n, p \in \mathbb{N}$, we consider the recognizer P system

$$\Pi(\langle n, p\rangle) = (\Gamma, \Sigma, \mu, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, multisets(M_{i,j})(1 \leq i \leq n, 1 \leq j \leq p),$$
$$multisets(M_{i,j,l})(1 \leq i \leq \lceil \tfrac{n}{2^l} \rceil, 1 \leq j \leq p, 1 \leq l \leq \lceil \log_2 n \rceil),$$
$$multisets(M_j)(1 \leq j \leq p), multisets(M_{d,l})(1 \leq l \leq \lceil \log_2 n \rceil),$$
$$multisets(M_r), \mathcal{R}, i_{in}, i_{out}),$$

from $\mathcal{AM}^0(-d, +n)$ where:

1. Working alphabet Γ:
   $\{\mathtt{yes}, \mathtt{no}, a, a'\} \cup \{a_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq 2i-1\} \cup$
   $\{t_{i,j}, f_{i,j} \mid 1 \leq i \leq n, 2i \leq j \leq 2n-1\} \cup$
   $\{T_{i,j}, F_{i,j} \mid 1 \leq i \leq n, 0 \leq j \leq n-1\} \cup \{T_i, F_i, t_i, f_i \mid 1 \leq i \leq n\} \cup$
   $\{x_{i,j,k}, \overline{x}_{i,j,k} \mid 1 \leq i \leq n, 1 \leq j \leq p, 1 \leq k \leq 3n\} \cup$
   $\{c_{i,j,l} \mid 1 \leq i \leq \frac{2n}{2^l}, 1 \leq j \leq p, 1 \leq l \leq \lceil \log_2 n \rceil\} \cup$
   $\{c_{j,l} \mid 1 \leq j \leq p, 1 \leq l \leq (j-1)(k+2)+1\} \cup \{d_j \mid 1 \leq j \leq p\} \cup$
   $\{d_{p,l} \mid 1 \leq l \leq \lceil \log_2 n \rceil + 1\} \cup alphabet(M_{i,j})(1 \leq i \leq n, 1 \leq j \leq p) \cup$
   $alphabet(M_{i,j,l})(1 \leq i \leq \lceil \frac{n}{2^l} \rceil, 1 \leq j \leq p, 1 \leq l \leq \lceil \log_2 n \rceil) \cup$
   $alphabet(M_j)(1 \leq j \leq p) \cup alphabet(M_{d,l})(1 \leq l \leq \lceil \log_2 n \rceil) \cup alphabet(M_r)$
2. Input alphabet Σ:
   $\{x_{i,j,0}, \overline{x}_{i,j,0} \mid 1 \leq i \leq n, 1 \leq j \leq p\}$
3. Initial multisets:
   $\mathcal{M}_1 = \emptyset, \mathcal{M}_2 = \emptyset, \mathcal{M}_3 = \{a_{i,1} \mid 1 \leq i \leq n\}$
4. The rule set $\mathcal{R}$ consists on the following rules:

1.1 Rules to create p copies of every possible truth assignment in each of the 2^n membranes labelled by 2.

$$[\,a_{i,j} \to a_{i,j+1}\,]_3 \quad \text{for } 2 \le i \le n, 1 \le j \le 2i-2$$

$$[\,a_{i,2i-1}\,]_3 \to [\,t_{i,2i}\,]_3\,[\,f_{i,2i}\,]_3 \quad \text{for } 1 \le i \le n$$

$$[\,a_{n,2n-1}\,]_3 \to [\,T_{n,n+1}\,]_3\,[\,F_{n,n+1}\,]_3$$

$$[\,[\quad]_3\,[\quad]_3\,]_2 \to [\,[\quad]_3\,]_2\,[\,[\quad]_3\,]_2$$

$$\left.\begin{array}{l}[\,t_{i,j} \to t_{i,j+1}\,]_3\\ [\,f_{i,j} \to f_{i,j+1}\,]_3\end{array}\right\} \text{for } 1 \le i \le n-1, 2i \le j \le 2n-2$$

$$\left.\begin{array}{l}[\,t_{i,2n-1} \to T_{i,i+1}\,]_3\\ [\,f_{i,2n-1} \to F_{i,i+1}\,]_3\end{array}\right\} \text{for } 1 \le i \le n-1$$

$$\left.\begin{array}{l}[\,T_{i,j} \to T_{i,j-1}\,]_3\\ [\,F_{i,j} \to F_{i,j-1}\,]_3\end{array}\right\} \text{for } 1 \le i \le n, 1 \le j \le i+1$$

$$\left.\begin{array}{l}[\,T_{i,0}\,]_3 \to T_{i,i}\,[\quad]_3\\ [\,F_{i,0}\,]_3 \to F_{i,i}\,[\quad]_3\end{array}\right\} \text{for } 1 \le i \le n$$

$$[\,T_{n,0}\,]_3 \to T_n\,[\quad]_3$$

$$[\,F_{n,0}\,]_3 \to F_n\,[\quad]_3$$

$$\left.\begin{array}{l}[\,T_{i,j} \to T_{i,j+1}\,]_2\\ [\,F_{i,j} \to F_{i,j+1}\,]_2\end{array}\right\} \text{for } 1 \le i \le n-2, i \le j \le n-2$$

$$\left.\begin{array}{l}[\,T_{i,n-1} \to T_i\,]_2\\ [\,F_{i,n-1} \to F_i\,]_2\end{array}\right\} \text{for } 1 \le i \le n-1$$

$$\left.\begin{array}{l}[\,T_i \to t_i^p\,]_2\\ [\,F_i \to f_i^p\,]_2\end{array}\right\} \text{for } 1 \le i \le n-1$$

2.1 Rules to check which clauses are satisfied by the truth assignments.

$$\left.\begin{array}{l}[\,x_{i,j,k} \to x_{i,j,k+1}\,]_2\\ [\,\overline{x}_{i,j,k} \to \overline{x}_{i,j,k+1}\,]_2\end{array}\right\} \text{for } 1 \le i \le n, 1 \le j \le p, 0 \le k \le 3n-1$$

$$\left.\begin{array}{l}x_{i,j,3n}\,[\quad]_{M_{i,j}} \to [\,a'\,]_{M_{i,j}}\\ \overline{x}_{i,j,3n}\,[\quad]_{M_{i,j}} \to [\,a'\,]_{M_{i,j}}\end{array}\right\} \text{for } 1 \le i \le n, 1 \le j \le p$$

$$[\,a' \to a\,]_{M_{i,j}} \quad \text{for } 1 \le i \le n, 1 \le j \le p$$

$$\left.\begin{array}{l}t_i\,[\quad]_{M_{i,j}} \to [\,a\,]_{M_{i,j}}\\ f_i\,[\quad]_{M_{i,j}} \to [\,a\,]_{M_{i,j}}\end{array}\right\} \text{for } 1 \le i \le n, 1 \le j \le p$$

$$M_{i,j}(3n+2) : a^2 \rightsquigarrow c_{i,j,1} \quad \text{in } k \text{ steps for } 1 \le i \le n, 1 \le j \le p$$

2.2 Rules to obtain only one copy of each object c_j, if possible.

$$\left.\begin{array}{l} c_{2i-1,j,l}\,[\quad]_{M_{i,j,l}} \to [\,a'\,]_{M_{i,j,l}} \\ c_{2i,j,l}\,[\quad]_{M_{i,j,l}} \to [\,a\,]_{M_{i,j,l}} \end{array}\right\} \text{ for } \begin{array}{l} 1 \le i \le \lceil \frac{n}{2^l} \rceil, \\ 1 \le j \le p, 1 \le l \le \lceil \log_2 n \rceil \end{array}$$

$$[\,a' \to a\,]_{M_{i,j,l}} \quad \text{for } \begin{array}{l} 1 \le i \le \lceil \frac{n}{2^l} \rceil, \\ 1 \le j \le p, 1 \le l \le \lceil \log_2 n \rceil \end{array}$$

2^{l-1} membranes of each type $M_{i,j,l}$ are present in the step $3n+1$

$$M_{i,j,l}((3n+3)+2l+k(l-1)) : \left.\begin{array}{l} a^2 \rightsquigarrow c_{i,j,l+1} \\ a \rightsquigarrow c_{i,j,l+1} \end{array}\right\} \text{ in } k \text{ steps for } \begin{array}{l} 1 \le i \le \lceil \frac{n}{2^l} \rceil, \\ 1 \le j \le p, \\ 1 \le l \le \lceil \log_2 n \rceil - 1 \end{array}$$

$$M_{1,j,\lceil \log_2 n \rceil}((3n+3)+(2+k)\lceil \log_2 n \rceil - 1) : \left.\begin{array}{l} a^2 \rightsquigarrow c_{j,1} \\ a \rightsquigarrow c_{j,1} \end{array}\right\} \text{ in } k \text{ steps for } 1 \le j \le p$$

3.1 Rules to check if all the clauses are satisfied by a truth assignment.

$$\left.\begin{array}{l} d_{j-1}\,[\quad]_{M_j} \to [\,a'\,]_{M_j} \\ c_{j,(j-1)(k+2)+1}\,[\quad]_{M_j} \to [\,a\,]_{M_j} \end{array}\right\} \text{ for } 1 \le j \le p$$

$$[\,c_{j,l} \to c_{j,l+1}\,]_2 \quad \text{for } 1 \le j \le p, 1 \le j \le (j-1)(k+2)$$

$$M_j((3n+3)+(2+k)\lceil \log_2 n \rceil + 2j + k(j-1)) : a^2 \rightsquigarrow d_j \text{ in } k \text{ steps} \quad \text{for } 1 \le j \le p$$

4.1 Rules to obtain only one copy of the object d_p, if possible.

$$[\,d_p\,]_2 \to d_{p,1}[\quad]_2$$

$$d_{p,l}\,[\quad]_{M_{d,l}} \to [\,a\,]_{M_{d,l}} \quad \text{for } 1 \le l \le \lceil \log_2 n \rceil$$

2^{l-1} membranes of each type $M_{d,l}$ are present in the step $(3n+3)+(2+k)(\lceil \log_2 n \rceil + p)$

$$M_{d,l}((3n+3)+(2+k)(\lceil \log_2 n \rceil + p)+2l+k(l-1)) : \left.\begin{array}{l} a^2 \rightsquigarrow d_{p,l+1} \\ a \rightsquigarrow d_{p,l+1} \end{array}\right\} \text{ for } 1 \le l \le \lceil \log_2 n \rceil$$

4.2 Rules to return the correct answer.

$$d_{p,\lceil \log_2 n \rceil + 1}\,[\quad]_{M_r} \to [\,a\,]_{M_r}$$

$$M_r((3n+3)+(2+k)(2\lceil \log_2 n \rceil + p + 1)) : \left.\begin{array}{l} a \rightsquigarrow \texttt{yes} \\ a^0 \rightsquigarrow \texttt{no} \end{array}\right\} \text{ in } k \text{ steps}$$

$$[\,\texttt{yes}\,]_1 \to \texttt{yes}\,[\quad]_1$$

$$[\,\texttt{no}\,]_1 \to \texttt{no}\,[\quad]_1$$

5. The input membrane is the membrane labelled by 2 ($i_{in} = 2$) and the output zone is the environment ($i_{out} = env$).

The proposed solution follows a brute force algorithm in the framework of recognizer P systems with active membranes without dissolution rules and division rules for elementary and non-elementary membranes, and it consists on the following stages:

1. *Generation stage*: Using rules from **1.1**, 2^n membranes labelled by 2 and 3 will be produced. Each of the membranes labelled by 2 will contain a different truth assignment for the n variables. This stage takes $3n + 1$ time steps.
2. *First checking stage*: In this stage, an object $c_{j,1}$ will be produced in the membranes labelled by 2 whose truth assignment makes true the clause C_j. First of all, with rules from **2.1**, multiple copies of objects $c_{i,j,1}$ will be produced, and later with the rules from **2.2** will lead to a single copy of objects $c_{j,1}$ if there was at least one object $c_{i,j,1}$. This stage takes $(2 + k)\lceil \log_2 n \rceil$.
3. *Second checking stage*: In this stage, an object d_p will be produced in a membrane labelled by 2 if the truth assignment associated with it makes true the whole formula φ, by using rules from **3.1**. This stage takes $p(2+k)$ time steps.
4. *Output stage*: Finally, by using rules from **4.1**, a single object $\mathbf{d}_{p,\lceil \log_2 n \rceil}$ will be produced in the skin membrane if there exists at least one truth assignment that makes true the formula φ. If such an object exists, the system will send an object `yes` to the environment. Otherwise, it will return an object `no`. This is done by using the rules from **4.2**. This stage takes $\lceil \log_2 n \rceil + 4$ time steps.

4 Details of the special machines

In the previous solution, new syntax with respect to the classical of membrane systems appears. Let us define some kind of "machines" that follows a not-so-much special behavior. A machine M_i is no more than a P system from the corresponding family, in this case, from $\mathcal{AM}^0(-d, +n)$. Given a machine M_i, we say that $multisets(M_i)$ will be the multisets of objects placed initially in the membranes within the structure of the machine M_i, $alphabet(M_i)$ will be the working alphabet of the machine and if there is a rule of the kind $M_i(t) : rules$, the machine M_i will execute the corresponding rule associated to the number of objects placed in the system at configuration $\mathcal{C}_t$. In k steps, the machine M_i will return the corresponding answer to the parent membrane. Of course, not all the machines will spend the same amount of time steps, since in the end they are P systems and different machines can spend different times, but for the sake of simplicity, we say that they spend the same number of time steps. Later, we will look for a simple machine that must spend exactly k time steps, and that machine will be a "sub-routine" used for the ones used in the solution.

As opposed to oracles, that in this sense they can be though as machines that start working when they receive some input, these machines are "running" from the first configuration; that is, they are P systems that work as a P system of its family, so it cannot accomplish tasks that are impossible for P systems of its own

family.

We can think that the "machine" can wait until step t by using subscripts and evolution rules. Since division rules are allowed for both elementary and non-elementary rules, we can ensure that we can obtain enough number of machines of each kind. For this purpose, if a machine M_i needs to be replicated into 2^k copies, a single copy of this machine is present at the beginning of the computation. In the skin membrane of this machine, there is a *leaf* membrane labelled by d, and such that it contains an object: a_1. Let us define the following rules:

$[\, a_{2i} \to a_{2i+1} \,]_d$ for $1 \leq i \leq k-1$

$[\, a_{2i-1} \,]_d \to [\, a_{2i} \,]_d [\, a_{2i} \,]_d$ for $1 \leq i \leq k$

$[\, [\quad]_d [\quad]_d \,]_{skin_{M_i}} \to [\, [\quad]_d \,]_{skin_{M_i}} [\, [\quad]_d \,]_{skin_{M_i}}$

By using these rules, we can obtain 2^k exact copies of the machine M_i in $2k-1$ steps. The label d is a label such that it is not used in the whole machine M_i. Since the rest of objects are supposed to be evolving within their corresponding membranes, this process does not affect in this task.

5 Duties of the special machines

Five different machines are described in the previous solution. As discussed in the previous section, a machine M_i such that it has a rules defined as $M_i(t) : rule$, then when the configuration $\mathcal{C}_t$ is reached, the rule will be executed from the next transition. In k time steps, it will return to its parent membrane the corresponding answer.

- $M_{i,j}$: If there are 2 copies of the object a, then it returns an object $c_{i,j,1}$. Otherwise, it returns nothing.
- $M_{i,j,l}$: If there are 1 or 2 copies of the object a, then it returns an object $c_{i,j,l+1}$. Otherwise, it returns nothing.
- M_j: If there are 2 copies of the object a, then it returns an object d_j. Otherwise, it returns nothing.
- $M_{d,l}$: If there are 1 or 2 copies of the object a, then it returns an object $d_{p,l+1}$. Otherwise, it returns nothing.
- M_r: If there are 1 copy of the object a, then it returns an object `yes`. If there are no copies of the object a, it returns `no`. Otherwise, it returns nothing.

As we can observe, three different behaviors are required here. For instance, the answers of $M_{i,j}$ and M_j are similar. Since they do not return the same object, we can think that we have a single type of machine M_i, and it is within a membrane labelled by $M'_{i,j}$ (respectively, M'_j). The behavior of this machine is simple: If there are 2 copies of the object a, then it returns `yes`. Otherwise, it returns `no`. The corresponding answer would be sent at the $(k-1)$-th time step to the membrane

$M'_{i,j}$ (resp., M'_j). Then, in the k-th time step, the membrane $M'_{i,j}$ (resp., M'_j) would send to the parent membrane an object $c_{i,j,1}$ (resp., d_j) if an object `yes` appeared in the previous configuration, and it would not send anything if an object `no` had appeared.

1. $M_{i,j}, M_j$: Differentiate between 1 and 2 copies of object a.
2. $M_{i,j,l}, M_{d,l}$: Differentiate between 1 or 2 and none copies of object a.
3. M_r: Differentiate between 1 and 0 copies of object a.

In fact, taking into account that we are *deciding* if the number of objects a is equal to a number, then we can generalize these three cases into a single one, in order to have a single problem to be solved.

6 Covering all the cases

What we are trying to solve here is deciding if the number of objects of a certain type corresponds to a particular number.

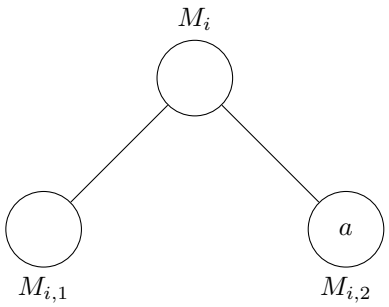
Then, the following question could generalize the previous problems:

> *Can we differentiate the existence of a single object from the non-existence or the multiple existence of the object?*

Or, more formally, if there is a single instance of an object, then return `yes`. Otherwise, return `no`.

We are going to give a proof on how these three cases can be reduced to this question.

1. The first case is reduced as follows: let $M_{i,1}$, $M_{i,2}$ and $M_{i,3}$ be three machines that solve the current problem. Then, we can think that the machine M_i is formed as follows:



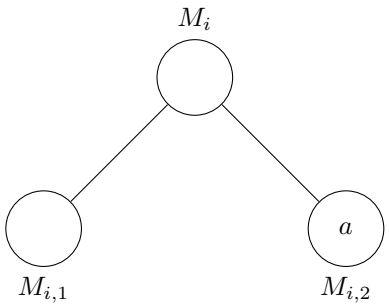
For sending objects a to each machine $M_{i,j}$, they can be replicated by using object evolution rules and sent to them by using send-in rules (for the order, subscripts can be used). $M_{i,1}$ will solve the problem normally; that is, if there is only one object a, then it will send an object $\texttt{yes}_1$ to M_i in k steps, otherwise, it will return an object $\texttt{no}_1$. At the same time, the problem will be solved in $M_{i,2}$, but as there is an object a present in the system, it will return an object $\texttt{yes}_2$ if there are no objects a in M_i, otherwise it will return an object $\texttt{no}_2$.

If $M_{i,1}$ returns an object $\texttt{yes}_1$, it means that there is only one copy of the object a. Therefore, M_i will not send anything to its parent membrane. If $M_{i,1}$ returns $\texttt{no}_1$, there are two possibilities: On the one hand, there can be no objects a, then $M_{i,2}$ will return an object $\texttt{yes}_2$. On the other hand, there can be two copies of object a, then $M_{i,2}$ will return $M_{i,2}$ will return an object $\texttt{no}_2$. The following table represent the desired output for each possibility:

Input	$M_{i,1}$	$M_{i,2}$	Output
$n = 0$	$\texttt{no}_1$	$\texttt{yes}_2$	no
$n = 1$	$\texttt{yes}_1$	$\texttt{no}_2$	no
$n = 2$	$\texttt{no}_1$	$\texttt{no}_2$	yes

Since these three cases are the only possible ones, it is easy to see that if only an object no is present, it must return no (that will not be sent to the parent membrane, since in these situations, there is only one or none copies of object a). The other possible situation is that two objects no appear. In that case, we should return an object yes (that later will be sent to the parent membrane as the corresponding object). It is easy to do it by changing objects $\texttt{no}_i$ to objects a. If there is a single copy, the machine M_i will return nothing. Otherwise, it will send to the parent membrane the corresponding object.

2. The second case is similar to the first one. In this case, the objective of M_i is to return an object when there are 1 or 2 copies of the object a. In the other case; that is, when there are no copies of the object a, it will return nothing. As in the previous case, we will have the following structure:



If $M_{i,1}$ and $M_{i,2}$ work as in the previous case, while there are 2 copies of the object a, it will return the correct answer. But in the case that there is a single copy of a, then it will return nothing, and this is not the behavior that we expect. But, with a simple trick, we can transform this answer to the correct one just by flipping the answer of $M_{i,2}$; that is, if there is a single object a, then return $\texttt{no}_2$. Otherwise, it returns $\texttt{yes}_2$. Remember that it is possible since it can have an external membrane that has object evolution rules similar to $\texttt{yes} \rightarrow \texttt{no}_2$ and $\texttt{no} \rightarrow \texttt{yes}_2$. Therefore, the following table represents the desired output taking into account the behavior of $M_{i,1}$ and $M_{i,2}$:

Input	$M_{i,1}$	$M_{i,2}$	Output
$n = 0$	$\texttt{no}_1$	$\texttt{yes}_2$	`no`
$n = 1$	$\texttt{yes}_1$	$\texttt{no}_2$	`yes`
$n = 2$	$\texttt{no}_1$	$\texttt{no}_2$	`yes`

By looking at the table, we can clearly see that if an object $\texttt{no}_2$ comes out from the machine $M_{i,2}$, then it will return `yes` (that will be sent to the parent membrane as the corresponding object). Then, we can transform object $\texttt{no}_2$ into an object a. Objects $\texttt{no}_1$, $\texttt{yes}_1$ and $\texttt{yes}_2$ will not evolve into an object a. Therefore, if objects $\texttt{no}_1$ and $\texttt{yes}_2$ are the output of machines $M_{i,1}$ and $M_{i,2}$ there will not be any object a in M_i, thus it will return a `no` (that will not be sent to the parent membrane). If object $\texttt{no}_2$ is the output of $M_{i,2}$, no matter the output of $M_{i,1}$, since it will not produce another object a will produce a `yes` as an output (and it will be sent to the parent membrane as the corresponding object).

3. The third case is trivial since only two scenarios can occur: On the one hand, if there is no object a it will return an object `no`. On the other hand, if there is an object a present in the machine, then it will return a `yes`.

The three previous problems have been reduced to the previously stated question. Therefore, having a P system from $\mathcal{AM}^0(-d, +n)$ capable of solving this

problem, would complete the solution. This machine must be totally independent from the input, and this machine will spend exactly k steps.

7 Reduction of the problem

In order to prove $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{R}}$, an efficient solution to an **NP**-complete problem by means of a family of recognizer P systems from $\mathcal{R}$ must be provided. In this paper, a "solution" to the SAT problem has been provided by means of a family of recognizer membrane systems from $\mathcal{AM}^0(-d, +n)$. This solution depends on the ability of these P systems to solve the proposed problem. Thus, the existence of a single membrane system of this family capable of solving this question would lead to $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}^0(-d,+n)}$.

In [1], by using the dependency graph technique, it was proved that $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0(-d,+n)}$. Therefore, a single membrane system from this family solving the cited problem would not exist unless $\mathbf{P} = \mathbf{NP}$.

Thus, a new tool to tackle the **P** *versus* **NP** problem has been stablished: If there exists a single membrane from $\mathcal{AM}^0(-d, +n)$ solving the problem of differentiating a single appearance of a certain object from its non-existence or the multiple existence, then $\mathbf{P} = \mathbf{NP}$.

8 Conclusions and future work

In this paper, a "solution" to the SAT problem in the framework of recognizer P systems from $\mathcal{AM}^0(-d, +n)$ has been given. This is not a real solution since it needs of special "machines" that execute tasks whose possible execution in this framework has not been demonstrated. Since a positive solution of this problem yields $\mathbf{P} = \mathbf{NP}$, a very powerful tool to tackle this problem has been raised.

Two interesting research lines open up: On the one hand, solve this problem as it would lead to a very powerful result (in fact, if it ends up being a question with an affirmative answer, an efficient mechanism to solve **NP**-complete problems would raise from the solution). On the other hand, to explore the existence of more conjectures of this type, since they can be helpful to solve, in an affirmative or in a negative way, the **P** *versus* **NP** problem.

References

1. M.Á. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero. On the power of dissolution in P systems with active membranes. In

R. Freund, Gh. Păun, Gr. Rozenberg, A. Salomaa (eds.). *Membrane Computing, 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers, Lecture Notes in Computer Science*, **3850** (2006), 224-240.

2. M.Á. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero. A uniform solution to SAT using membrane creation. *Theoretical Computer Science*, **371**, 1–2 (2007), 54–61.
3. L. Pan, T-.O.Ishdorj. P systems with active membranes and separation rules. *Journal of Universal Computer Science*, **10**, 5 (2004), 630–649.
4. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143, and *Turku Center for CS-TUCS Report* No. 208, 1998.
5. Gh. Păun. P systems with active membranes: attacking **NP**–complete problems, *Journal of Automata, Languages and Combinatorics*, **6**, 1 (2001), 75-90.
6. M.J. Pérez-Jiménez, A. Riscos-Núñez, Á. Romero-Jiménez, D. Woods. Complexity: Membrane division, membrane creation. In: Păun et al. (eds.), The Oxford Handbook of Membrane Computing, Oxford University Press, 2010, chap. 12, 302–336.
7. M.J. Pérez-Jiménez, Á. Romero Jiménez. Simulating Turing Machines by P systems with External Output. *Fundamenta Informaticae*, Annales Societatis Mathematicae Polonae, Series IV, IOS Press, Amsterdam, **49**, 1-3 (2002), 273-287
8. M.J. Pérez-Jiménez, Á. Romero-Jiménez, F. Sancho-Caparrini. Decision P systems and the $\mathbf{P} \neq \mathbf{NP}$ conjecture. In Gh. Păun, Gr. Rozenberg, A. Salomaa, C. Zandron (eds.) Membrane Computing 2002. *Lecture Notes in Computer Science*, **2597** (2003), 388-399. A preliminary version in Gh. Păun, C. Zandron (eds.) *Pre-proceedings of Workshop on Membrane Computing 2002*, MolCoNet project-IST-2001-32008, Publication No. 1, Curtea de Arges, Romanian, August 19-23, 2002, pp. 345-354.
9. Á. Romero-Jiménez, D. Orellana-Martín. Design Patterns for Efficient to NP-Complete Problems in Membrane Computing. In C. Graciani et al. (eds.), Enjoying Natural Computing: Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday, Springer, 2018, chap. 19, 237–255.