Counting Membrane Systems

Luis Valencia-Cabrera, David Orellana-Martín,
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: {`lvalencia, dorellana, ariscosn, marper`}`@us.es`

Abstract. A decision problem is one that has a `yes`/`no` answer, while a counting problem asks how many possible solutions exist associated with each instance. Every decision problem X has associated a counting problem, denoted by $\#X$, in a natural way by replacing the question "*is there a solution*?" with "*how many solutions are there*?". Counting problems are very attractive from a computational complexity point of view: if X is an **NP**-complete problem then the counting version $\#X$ is **NP**-hard, but the counting version of some problems in class **P** can also be **NP**-hard.
In this paper, a new class of membrane systems is presented in order to provide a natural framework to solve counting problems. The class is inspired by a special kind of non-deterministic Turing machines, called counting Turing machines, introduced by L. Valiant. A polynomial-time and uniform solution to the counting version of the `SAT` problem (a well-known #**P**-complete problem) is also provided, by using a family of counting polarizationless P systems with active membranes, without dissolution rules and division rules for non-elementary membranes but where only very restrictive cooperation (minimal cooperation and minimal production) in object evolution rules is allowed.

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 computational paradigm inspired by the structure and functioning of the living cells as well as from the cooperation of cells in tissues, organs, and organisms. This paradigm provides distributed parallel and non-deterministic computing models. All of them share the main syntactical ingredients: a finite alphabet (the *working* alphabet whose elements are called *objects*), a finite set of processor units delimiting *compartments* (called *membranes*, *cells* or *neurons*) interconnected by a *graph-structure* in such manner that initially each processor contains a multiset of objects, a finite set of *evolution rules* which provides the dynamic of the system, and an *environment*.

According to the inspiration they come from, there are basically three approaches: *cell-like* P systems, where the compartments are arranged like in a living cell (that is, in a hierarchical structure) [7]; *tissue-like* P systems, whose inspiration comes from the living tissues, where cells bump into each other and communicate through pores or other membrane mechanisms [20]; and *neural-like* P systems, which mimic the way that neurons communicate with each other by means of short electrical impulses, identical in shape (voltage), but emitted at precise moments of time [10]. The term *membrane system* is used to refer to *cell-like* P systems, *tissue-like* P systems or *neural-like* P systems indistinctly.

In cell-like and neural-like P systems the environment plays a "passive" role in the sense that it only receives objects, but cannot contribute objects to the system. However, in tissue-like P systems the role played by the environment is "active"in the sense that it can receive objects from the system and also send objects inside the system, and the objects initially placed in the environment have an arbitrarily large number of copies.

Decision problems (those having a `yes`/`no` answer) have associated a language in a natural way, in such manner that solving a decision problem is expressed in terms of the recognition of the language associated with it. In this context, recognizer membrane systems was introduced in order to define what solving a decision problem means in the framework of Membrane Computing [12]. P systems with active membranes was introduced in [8, 9]. These cell-like models make use of electrical charges associated with membranes and division rules. They have the ability to provide efficient solutions to computationally hard problems, by making use of an exponential workspace created in a polynomial time. The class of decision problems which can be solved by families of P systems with active membranes with dissolution rules and which use division for elementary and non-elementary membranes is equal to **PSPACE** [14]. However, if electrical charges are removed from the usual framework of P systems with active membranes, then dissolution rules come to play a relevant role (without them, only problems in class **P** can be solved in an efficient way, even in the case that division for non-elementary membranes are permitted [5]). P systems with active membranes and without polarizations were initially studied in [1, 2] by replacing electrical charges by the ability to change the label of the membranes.

Counting problems (those asking how many possible solutions exist associated with each instance) have a natural number, instead of a `yes`/`no`, as an answer. Each decision problem has associated a counting problem in a natural way by replacing "*there exists a solution*" with "*how many solutions*". For instance, the counting problem associated with the `SAT` problem (denoted by #`SAT`) is the following: given a Boolean formula φ in conjunctive normal form, how many truth assignments make true φ? It is worth pointing out that the counting problem associated with a decision problem may be harder than the decision problem, from a complexity point of view.

The main goal of this paper is twofold. On the one hand, to provide a formal framework in Membrane Computing to solve counting problems by introducing *counting membrane systems*. This approach was first initiated by A. Alhazov et al. [3] and now, the computing model is formally defined inspired by *counting Turing machines* introduced by L. Valiant in 1979: "*a standard nondeterministic TM with an auxiliary output device that (magically) prints in binary notation on a special tape the number of accepting computations induced by the input*". L. Valiant also introduced the complexity class #**P** of functions that can be computed by *counting Turing machines* running in polynomial time [19]. The concept of #**P**-complete problems is defined in a natural way by considering *parsimonious reduction*, that is reduction which preserves the number of solutions.

On the other hand, following the works initiated in [15–17], a uniform and polynomial-time solution to the #`SAT` problem, a well-known #**P**-complete problem, is provided by means of a family of counting membrane systems from $\mathcal{DAM}^0_{\mathtt{c}}(mcmp, +c, -d, -n)$ whose elements are (counting) polarizationless P systems with active membranes where labels of membranes keep unchanged by the application of rules, but where dissolution rules and division rules for non-elementary membranes are forbidden and some kind of very restrictive cooperation in object evolution rules is allowed.

The paper is structured as follows. Next, we shortly recall some preliminary basic definitions related to abstract problems. Section 3 introduces counting membrane systems, and the concept of uniform polynomial-time solvability of counting problems by means of families of counting membrane systems is presented (specifically, the class $\mathcal{DAM}^0_{\mathtt{c}}(mcmp, +c, -d, -n)$ is defined). Section 4 is devoted to showing a uniform and polynomial-time solution of the #`SAT` by using a family of counting polarizationless P systems with active membranes, without dissolution rules and with division only for elementary membranes where minimal cooperation and minimal production is allowed in object evolution rules. The paper ends with some conclusions and final remarks.

2 Abstract problems

Roughly speaking, an *abstract problem* is a "general question to be answered, usually possessing several parameters whose values are left unspecified" [4]. Solving an abstract problem consists of answering the question associated with it. Thus, an abstract problem consists of a (finite or infinite) set of *concrete problems*, called *instances*, obtained by specifying particular values for all parameters. Each instance has an associated set (eventually empty) of possible *solutions* and the answer to the general question of the problem is related to that set.

A *search problem* (or *function problem*) is an abstract problem such that the question is to identify/find *one* solution from the set of possible solutions associated with each instance. For example, given a Boolean formula φ in conjunctive normal form to find any truth assignment which makes it true, or if there is no such truth assignment, answer "`no`". That is, in this problem a "function" must be computed in such manner that for every input formula φ, this "function" may have many possible outcomes (any satisfying truth assignments) or none.

A *decision problem* is a particular case of search problem. Specifically, a decision problem can be viewed as an abstract problem that has a `yes` or `no` answer. This kind of problems can be formulated by specifying a generic instance of the problem and by stating a `yes`/`no` question concerning to the generic instance [4]. For example, the `SAT` problem is the following decision problem: given a Boolean formula in conjunctive normal form, is there a truth assignment that makes the formula true?

Informally, a *counting problem* is an abstract problem such that one asks how many possible solutions exist associated with each generic instance, that is, in this kind of problems the output is a natural number rather than just `yes` or `no` as in a decision problem. For example, the #`SAT` problem previously defined is a particular case of a counting problem.

In *optimization problems* we seek to find a *best solution* associated with each instance among a collection of feasible solutions, according to a concept of optimality given by an objective function associated with the problem. For example, the `MAXSAT` problem is the following optimization problem: given a Boolean formula in conjunctive normal form and a natural number k, is there a truth assignment that makes true at least k of the clauses?

Next, we formally define the previous concepts. A *search problem* (or *function problem*) X is a tuple (Σ_X, I_X, S_X) such that: (a) Σ_X is a finite alphabet; (b) I_X is a language over Σ_X whose elements are called *instances* of X; and (c) S_X is a function whose domain is I_X and for each $u \in I_X$, $S_X(u)$ is a set whose elements are called *solutions* for u. To *solve a search problem* X means the following: for each instance $u \in I_X$ return one element of $S_X(u)$ in the case that $S_X(u) \neq \emptyset$; otherwise, return "`no`". Each search problem $X = (\Sigma_X, I_X, S_X)$ has an associated binary relation Q_X defined as follows: $Q_X = \{(u, z) \mid u \in I_X \wedge z \in S_X(u)\}$. Then, solving the search problem X can be interpreted as follows: given an instance $u \in X$, find one element z such that $(u, z) \in Q_X$. We say that a deterministic Turing machine M solves a search problem X if, given as input any instance $u \in I_X$, the machine M with input u returns some element belonging to $S_X(u)$ (M accepts u) in the case that $S_X(u) \neq \emptyset$; otherwise, it returns "`no`" (M rejects u). That is, the Turing machine M computes a multivalued function F on I_X: this function may have many possible outcomes or none.

An *optimization problem* X is a tuple $(\Sigma_X, I_X, S_X, O_X)$ such that:

– (Σ_X, I_X, S_X) is a search problem.
– O_X is a function whose domain is I_X and for each instance $u \in I_X$ and for each possible solution $a \in S_X(u)$ associated with u, $O_X(u, a)$ is a positive rational number.
– For each instance $u \in I_X$ there exists a solution $a \in S_X(u)$ such that either $\forall b\, (b \in S_X(u) \Rightarrow O_X(u, b) \leq O_X(u, a))$ (we say that a is a *maximal solution* to instance u), or $\forall b\, (b \in S_X(u) \Rightarrow O_X(u, b) \geq O_X(u, a))$ (we say that a is a *minimal solution* to instance u).

To *solve an optimization problem* X means the following: for each instance $u \in I_X$, return a *maximal solution* or a *minimal solution.* We say that a deterministic Turing machine M solves an optimization problem X if, given an instance $u \in I_X$, the machine M with input u returns one *optimal* (maximal or minimal) solution associated with that instance.

A *decision problem* X is a search problem (Σ_X, I_X, S_X) such that for each instance $u \in I_X$, $S_X(u) = \{0\}$ or $S_X(u) = \{1\}$. In the case $S_X(u) = \{0\}$ we say that the answer of the decision problem is *negative* (`no`) for instance u. In the case $S_X(u) = \{1\}$ we say that the answer of the decision problem is *affirmative* (`yes`) for instance u. To *solve a decision problem* X means the following: for each instance $u \in I_X$, return `yes` in the case $S_X(u) = \{1\}$, otherwise, return `no`. Let us notice that a decision problem $X = (\Sigma_X, I_X, S_X)$ can be viewed as an optimization problem $(\Sigma_X, I_X, S_X, O_X)$ where $O_X(u, a)$ is constant, always equal to 1 (recall that for each instance $u \in I_X$ the set of possible solutions $S_X(u)$ is a singleton, either $\{0\}$ or $\{1\}$). Each decision problem $X = (\Sigma_X, I_X, S_X)$ has an associated language L_X defined as follows: $L_X = \{u \in \Sigma_X^* \mid S_X(u) = \{1\}\}$. Conversely, each language L over an alphabet Γ has an associated decision problem $X_L = (\Sigma_{X_L}, I_{X_L}, S_{X_L})$ defined as follows: $\Sigma_{X_L} = \Gamma$, $I_{X_L} = \Gamma^*$ and $S_{X_L}(u) = \{1\}$, for each $u \in L$, and $S_{X_L}(u) = \{0\}$, for each $u \notin L$. According to these definitions, for each decision problem X we have $X_{L_X} = X$ and for each language L we have $L_{X_L} = L$. A deterministic Turing machine M is said to solve a decision problem X if machine M *recognizes* or *decides* the language L_X associated with the problem X, that is, for any string u over Σ_X, if $u \in L_X$, then the answer of M on input u is *yes* (that is, M accepts u), and the answer is *no* otherwise (that is, M rejects u). A *non-deterministic* Turing machine M is said to solve a decision problem X if machine M *recognizes* L_X, that is, for any string u over Σ_X, $u \in L_X$ if and only if there exists <u>at least one</u> computation of M with input u such that the answer is *yes*.

A *counting problem* X is a tuple $(\Sigma_X, I_X, S_X, F_X)$ such that (Σ_X, I_X, S_X) is a search problem and F_X is the function whose domain is I_X, defined as follows: $F_X(u) = |S_X(u)|$, where $|S_X(u)|$ denotes the number of elements of the set $S_X(u)$, for each instance $u \in I_X$. A counting problem X can be considered as a particular case of a search problem expressed as follows: given an instance $u \in I_X$, how many z are there such that $(u, z) \in Q_X$? (where Q_X is the binary relation associated with the search problem). A counting Turing machine M is said to solve a counting problem X if, given an instance $u \in I_X$, the number of the accepting computations of M with input u is equal to the number of elements of the set $S_X(u)$, that is, the number of possible solutions associated with u.

3 Counting membrane systems

The main purpose of computational complexity theory is to provide bounds on the amount of computational resources necessary for any mechanical procedure that solves an abstract problem. Usually, this theory deals with languages encoding/representing decision problems. The solvability of decision problems is expressed in terms of recognize/decide the languages associated with them. In order to formally define what it means to solve decision problems in Membrane

Computing, a new variant called *recognizer membrane systems* was introduced in [12] (so-called *accepting* P systems) for cell-like P systems, in [11] for tissue-like P systems, and in [6] for neural-like P systems (so-called *accepting* spiking neural P systems). Next, a new class of membrane systems, called *counting membrane systems*, is introduced as a framework where counting problems can be solved in a natural way. These systems are inspired from *counting Turing machines* introduced by L. Valiant [19] and from recognizer membrane systems where the Boolean answer of these systems is replaced by an answer encoded by a natural number expressed in a binary notation (placed in the environment associated with the halting configuration).

Definition 1. *A counting membrane system Π is a membrane system such that:*

- *There exist two distinguished disjoint alphabets Σ (input alphabet) and Φ (final alphabet) both of them strictly contained in the working alphabet Γ of Π. Furthermore, a total order in the final alphabet $\Phi = \{a_0, a_1, \ldots, a_n\}$ is considered.*
- *The membrane system has an input compartment labelled by i_{in}.*
- *All computations of the system halt.*
- *For each computation of the system, the environment associated with the corresponding halting configuration, may contain objects from Φ, but each of them with multiplicity at most one.*

According to Definition 1, the *result of any computation $\mathcal{C}$* of a counting P system is a natural number whose binary expression is encoded by the objects from the final alphabet placed in the environment associated with its halting configuration, according to the following criterion: (a) if the set of objects in Φ placed in the environment of the corresponding halting configuration is $\{a_{i_1}, \ldots, a_{i_r}\}$, then the answer of $\mathcal{C}$ is the natural number $2^{i_1} + \ldots + 2^{i_r}$; and (b) if that set is the empty set then the answer of $\mathcal{C}$ is 0.

Many different classes of counting membrane systems depending on the kind of rules can be considered. For example, transition counting P systems, polarizationless counting P systems with active membranes, tissue counting P systems with symport/antiport rules can be defined in a natural way. Then, we will use a subscript $\mathbf{c}$ to emphasize that we are dealing with some kind of counting membrane system. For instance, $\mathcal{DAM}^0_{\mathbf{c}}(+e, +c, -d, -n)$ denotes the class of all polarizationless counting P systems with active membranes which use object evolution rules, communication rules and division rules only for elementary membranes, but dissolution rules are not allowed.

It is worth pointing out that any recognizer membrane system Π can be considered as a "particular case" of counting membrane system, where the final alphabet Φ is a singleton alphabet $\{a_0\}$ and the rules of the counting system are obtained from the rules of the recognizer system replacing `yes` with a_0 and replacing object `no` with a garbage object $\natural$ different from a_0.

3.1 Polynomial complexity classes for counting membrane systems

The concept of polynomial encoding in recognizer membrane systems was introduced in [13] and polynomial encodings are stable under polynomial-time reductions. This concept can be translated to counting membrane systems in a natural way.

Definition 2. *Let X be a counting problem whose set of instances is I_X. Let $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ be a family of counting membrane systems. A* polynomial encoding *of X in $\mathbf{\Pi}$ is a pair (cod, s) of polynomial-time computable functions over I_X such that $s(u)$ is a natural number (obtained by means of a* reasonable *encoding scheme) and $cod(u)$ is a multiset over the input alphabet of $\Pi(s(u))$, for each instance $u \in I_X$.*

Definition 3. *A counting problem $X = (\Sigma_X, I_X, S_X, F_X)$ is* solvable in polynomial time and in a uniform way *by a family of counting membrane systems $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ from a class $\mathcal{R}_c$, denoted by $X \in \mathbf{PCMS}_{\mathcal{R}_c}$, if the following holds:*

– *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$ (in unary).*
– *There exists a polynomial encoding (cod, s) of X in $\mathbf{\Pi}$ such that:*
  - *The family $\mathbf{\Pi}$ is polynomially bounded with respect to (X, cod, s), that is, there exists a natural number $k \in \mathbb{N}$ such that for each instance $u \in I_X$, every computation of the system $\Pi(s(u))$ with input $cod(u)$ performs at most $|u|^k$ steps.*
  - *For each instance $u \in I_X$ and for each computation $\mathcal{C}$ of $\Pi(s(u))$ with input $cod(u)$ we have the result of $\mathcal{C}$ is $F_X(u)$.*

Having in mind that any recognizer membrane system Π can be considered as a "particular case" of counting membrane system, we have $\mathbf{PMC}_{\mathcal{R}} \subseteq \mathbf{PCMS}_{\mathcal{R}}$, for any class of recognizer systems $\mathcal{R}$.

3.2 Counting membrane systems from $\mathcal{DAM}^0_c(mcmp, +c, -d, -n)$

Let us recall that $\mathcal{DAM}^0(+e, +c, -d, -n)$ denotes the class of all recognizer polarizationless P systems with active membranes (μ denotes the membrane structure, Γ denotes the working alphabet and H denotes the set of labels) such that the set of rules is of the following forms:

⋆ $[\, a \to u \,]_h$ for $h \in H$, $a \in \Gamma$, u is a finite multiset over Γ (*object evolution rules*).

⋆ $a\,[\ \ ]_h \to [\, b \,]_h$ for $h \in H$, $a, b \in \Gamma$ and h is not the label of the root of μ (*send-in communication rules*).

⋆ $[\, a \,]_h \to b\,[\ \ ]_h$ for $h \in H$, $a, b \in \Gamma$ (*send-out communication rules*).

- ⋆ $[\,a\,]_h \to [\,b\,]_h\;[\,c\,]_h$ for $h \in H$, $a, b, c \in \Gamma$ and h is the label of an elementary membrane different of the root of μ (*division rules for elementary membranes*).

It is well known [5] that only problems in class **P** can be solved in polynomial time (and in a uniform way) by means of families from $\mathcal{DAM}^0(+e, +c, -d, -n)$. Moreover, this holds even in the case that division rules for elementary and non-elementary membranes are permitted.

By incorporating a restricted cooperation in object evolution rules, a uniform polynomial-time solution to the SAT problem, a well-known **NP**-complete problem [4], has been provided [17]. Specifically, *minimal cooperation and minimal production* (**mcmp**) in object evolution rules has been considered, that is, rules of the forms $[\,a \to b\,]_h$ or $[\,a\,b \to c\,]_h$, where $a, b, c \in \Gamma$, but at least one object evolution rule is of the second type. The corresponding class of recognizer P systems was denoted by $\mathcal{DAM}^0(mcmp, +c, -d, -n)$. Then we denote by $\mathcal{DAM}^0_{\mathbf{c}}(mcmp, +c, -d, -n)$ the class of all counting polarizationless P systems with active membranes, with minimal cooperation and minimal production in object evolution rules, with communication rules and division rules only for elementary membranes, but without dissolution rules.

4 A solution to #SAT in $\mathcal{DAM}^0_{\mathbf{c}}(mcmp, +c, -d, -n)$

In this section a uniform and polynomial-time solution to the counting problem #SAT problem, a well-known #**P**-complete problem, is provided by means of a family of counting membrane systems from $\mathcal{DAM}^0_{\mathbf{c}}(mcmp, +c, -d, -n)$. For that, the solution to the SAT problem given in [17] by using a family of membrane systems from $\mathcal{DAM}^0(mcmp, +c, -d, -n)$ is adapted, basically, in what concerns to the output stage.

Let us recall that the polynomial-time computable function (the *Cantor pair function*) $\langle n, p\rangle = ((n+p)(n+p+1)/2)+n$ is a primitive recursive and bijective function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. The family $\mathbf{\Pi} = \{\Pi(t) \mid \; t \in \mathbb{N}\}$ is defined in such a manner that system $\Pi(t)$ will process any Boolean formula φ in conjunctive normal form (CNF) with n variables and p clauses, where $t = \langle n, p\rangle$, provided that the appropriate input multiset $cod(\varphi)$ is supplied to the system (through the corresponding input membrane), and will answer how many truth assignments make true the input formula φ.

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

$$\Pi(\langle n, p\rangle) = (\Gamma, \Sigma, \Phi, H, \mu, \mathcal{M}_1, \mathcal{M}_2, \mathcal{R}, i_{in})$$

from $\mathcal{DAM}^0(mcmp, +c, -d, -n)$, defined as follows:

(1) Working alphabet $\Gamma = \{\beta\,,\,\natural\} \cup \{\alpha_i \mid 0 \le i \le 2n+2p+1\} \cup$
$\{a_{i,j}\,,\mid 0 \le i \le n-1,\, 0 \le j \le i\} \cup \{a_i\,,\,\gamma_i \mid 0 \le i \le n-1\} \cup$
$\{b_{i,k} \mid 1 \le i \le n, 1 \le k \le i\} \cup \{c_j \mid 1 \le j \le p\} \cup$
$\{d_j \mid 2 \le j \le p\} \cup \{t_{i,k}, f_{i,k} \mid 1 \le i \le n, i \le k \le n+p-1\} \cup$
$\{T_{i,k}, F_{i,k} \mid 1 \le i \le n, 0 \le k \le n-1\} \cup \{T_i, F_i \mid 1 \le i \le n\} \cup$
$\{x_{i,j,k}, \overline{x}_{i,j,k}, x^*_{i,j,k} \mid 1 \le i \le n, 1 \le j \le p, 0 \le k \le n+p\}$.

(2) Input alphabet $\Sigma = \{x_{i,j,0}, \overline{x}_{i,j,0}, x^*_{i,j,0} \mid 1 \leq i \leq n, 1 \leq j \leq p\}$.
(3) Final alphabet $\Phi = \{a_i \mid 0 \leq i \leq n-1\}$.
(4) $H = \{1, 2\}$.
(5) Membrane structure: $\mu = [\ [\ \ ]_2\ ]_1$, that is, $\mu = (V, E)$ where $V = \{1, 2\}$ and $E = \{(1, 2)\}$.
(6) Initial multisets: $\mathcal{M}_1 = \{\alpha_0^n\}$, $\mathcal{M}_2 = \{\beta, b_{i,1}, T^p_{i,0}, F^p_{i,0} \mid 1 \leq i \leq n\}$.
(7) The set of rules $\mathcal{R}$ consists of the following rules:

7.1 Rules for a general counter.

$[\ \alpha_k \longrightarrow \alpha_{k+1}\ ]_1$, for $0 \leq k \leq 2n + 2p$

7.2 Rules to generate all truth assignments.

$[\ b_{i,i}\ ]_2 \longrightarrow [\ t_{i,i}\ ]_2\ [\ f_{i,i}\ ]_2$, for $1 \leq i \leq n$

$[b_{i,k} \longrightarrow b_{i,k+1}\ ]_2$, for $2 \leq i \leq n \wedge 1 \leq k \leq i-1$

7.3 Rules to generate suitable objects in order to start the next stage.

$$\left.\begin{array}{l} [t_{i,k} \longrightarrow t_{i,k+1}\ ]_2 \\ [f_{i,k} \longrightarrow f_{i,k+1}\ ]_2 \end{array}\right\} 1 \leq i \leq n-1 \wedge i \leq k \leq n-1$$

$$\left.\begin{array}{l} [T_{i,k} \longrightarrow T_{i,k+1}\ ]_2 \\ [F_{i,k} \longrightarrow F_{i,k+1}\ ]_2 \end{array}\right\} 1 \leq i \leq n, 0 \leq k \leq n-2$$

$$\left.\begin{array}{l} [T_{i,n-1} \longrightarrow T_i\ ]_2 \\ [F_{i,n-1} \longrightarrow F_i\ ]_2 \end{array}\right\} 1 \leq i \leq n$$

7.4 Rules to filter out either T_i or F_i, according to each truth assignment.

$$\left.\begin{array}{l} [t_{i,k}\ F_i \longrightarrow t_{i,k+1}\ ]_2 \\ [f_{i,k}\ T_i \longrightarrow f_{i,k+1}\ ]_2 \end{array}\right\} 1 \leq i \leq n \wedge n \leq k \leq n+p-2$$

$$\left.\begin{array}{l} [t_{i,n+p-1}\ F_i \longrightarrow \natural\ ]_2 \\ [f_{i,n+p-1}\ T_i \longrightarrow \natural\ ]_2 \end{array}\right\} 1 \leq i \leq n$$

7.5 Rules to prepare the input formula for check clauses.

$$\left.\begin{array}{l} [\ x_{i,j,k} \longrightarrow x_{i,j,k+1}\ ]_2 \\ [\ \overline{x}_{i,j,k} \longrightarrow \overline{x}_{i,j,k+1}\ ]_2 \\ [\ x^*_{i,j,k} \longrightarrow x^*_{i,j,k+1}\ ]_2 \end{array}\right\} 1 \leq i \leq n,\ 1 \leq j \leq p,\ 0 \leq k \leq n+p-1$$

7.6 Rules for the first checking stage.

$$\left.\begin{array}{l} [T_i\ x_{i,j,n+p} \longrightarrow c_j\ ]_2 \\ [T_i\ \overline{x}_{i,j,n+p} \longrightarrow \natural\ ]_2 \\ [T_i\ x^*_{i,j,n+p} \longrightarrow \natural\ ]_2 \\ [F_i\ x_{i,j,n+p} \longrightarrow \natural\ ]_2 \\ [F_i\ \overline{x}_{i,j,n+p} \longrightarrow c_j\ ]_2 \\ [F_i\ x^*_{i,j,n+p} \longrightarrow \natural\ ]_2 \end{array}\right\} 1 \leq i \leq n \wedge 1 \leq j \leq p$$

7.7 Rules for the second checking stage.

$[c_1\ c_2 \longrightarrow d_2\ ]_2$

$[d_j\ c_{j+1} \longrightarrow d_{j+1}\ ]_2$, for $2 \leq j \leq p-1$

7.8 Rules to prepare objects in the skin membrane.

$[\ \beta\ d_p \longrightarrow \gamma_0\ ]_2$

$[\ \gamma_0\ ]_2 \longrightarrow \gamma_0\ [\ ]_2$, for $0 \leq i \leq n-1$

7.9 Rules to prepare objects encoding the binary output.

$[\ \gamma_i^2 \longrightarrow \gamma_{i+1}\ ]_1$, for $0 \leq i \leq n-2$

$[\ \alpha_{2n+2p+1}\ \gamma_i \longrightarrow a_{i,0}\ ]_1$, for $0 \leq i \leq n-1$

$[\ a_{i,j} \longrightarrow a_{i,j+1}\ ]_1$, for $1 \leq i \leq n-1, 0 \leq j \leq i-1$

7.10 Rules to produce the output.
$[\ a_{i,i}\ ]_1 \longrightarrow\ a_i\ [\ ]_1\,,$ for $0 \le i \le n-1$

(8) The input membrane is the membrane labelled by 2 ($i_{in} = 2$) and the output region is the environment.

4.1 An overview of the computation

It is easy to check that each P system $\Pi(\langle n, p\rangle)$ previously defined is deterministic.

We consider the polynomial encoding (cod, s) from #`SAT` in $\mathbf{\Pi}$ defined as follows: let φ be a Boolean formula in conjunctive normal form. Let $Var(\varphi) = \{x_1, \cdots, x_n\}$ be the set of propositional variables and $\{C_1, \cdots, C_p\}$ the set of clauses of φ. Let us assume that the number of variables and the number of clauses of the input formula φ, are greater than or equal to 2. Then, we define $s(\varphi) = \langle n, p\rangle$ and

$$cod(\varphi) = \{x_{i,j,0} \mid\ x_i \in C_j\} \,\cup\, \{\overline{x}_{i,j,0} \mid \neg x_i \in C_j\} \,\cup\, \{x^*_{i,j,0} \mid x_i \notin C_j, \neg x_i \notin C_j\}$$

Notice that we can represent this multiset as a matrix, in such a way that the j-th row ($1 \le j \le p$) encodes the j-th clause C_j of φ, and the columns ($1 \le i \le n$) are associated with variables. We denote by $cod_k(\varphi)$ the multiset $cod(\varphi)$ when the third index of all objects is equal to k.

The Boolean formula φ will be processed by the system $\Pi(s(\varphi))$ with input multiset $cod(\varphi)$. Next, we informally describe how that system works.

The solution proposed is inspired by the solution provided to the `SAT` problem in [17], consisting of the following stages:

- *Generation stage*: by applying division rules from **7.2**, all truth assignments for the variables $\{x_1, \ldots, x_n\}$ associated with φ are produced. This stage takes exactly n computation steps and at the i-th step, $1 \le i \le n$, of this stage, division rule is triggered by object $b_{i,i}$, producing two new membranes with all its remaining contents replicated in the new membranes labelled by 2. Simultaneously to these divisions, objects $t_{i,k}, f_{i,k}, T_{i,k}, F_{i,k}$ (by applying rules from **7.3**) and objects $x_{i,j,k}, \overline{x}_{i,j,k}, x{*}_{i,j,k}$ (by applying rules from **7.5**) evolve during this stage in such manner that at configuration $\mathcal{C}_n$:
  - (a) There is a membrane labelled by 1 which contains n copies of object α_n.
  - (b) There are 2^n membranes labelled by 2 such that each of them contains: a copy of object β, the set $cod_n(\varphi)$, the multiset $\{T_i^p, F_i^p \mid 1 \le i \le n\}$; and a different subset $\{r_{1,n}, \ldots, r_{n,n}\}$, being $r \in \{t, f\}$.
- *Preparation of enough copies for each truth assignment*: in this stage p copies (p is the number of clauses of φ) of each truth assignment are prepared, in order to allow the checking of the literal associated with each variable in each clause. This is done by means of a filtering process applied over the objects T_i and F_i (remember that we have p copies of both of them available on each of the membranes after having applied rules from **7.3** over the initial multiset). By using minimal cooperation and minimal production (applying

rules from **7.4**), objects $t_{i,k}$ (respectively, object $f_{i,k}$) are used to remove all copies of F_i (respectively, T_i). This stage takes exactly p steps, and at configuration $\mathcal{C}_{n+p}$:
(a) The root membrane (labelled by 1) contains n copies of object α_{n+p}.
(b) There are 2^n membranes labelled by 2 such that each of them contains: a copy of object β, n copies of the garbage object $\natural$, the set $cod_{n+p}(\varphi)$, and a different multiset $\{R_1^p, \ldots, R_n^p\}$, being $R \in \{T, F\}$, which corresponds to the truth assignment associated to this membrane.

– *First Checking stage*: by applying rules from **7.6**, we check whether or not each clause of the input formula φ is satisfied by the truth assignments prepared in the previous stage, encoded by each membrane labelled by 2. This stage takes exactly one computation step and at configuration $\mathcal{C}_{n+p+1}$:
(a) The root membrane (labelled by 1) contains n copies of object α_{n+p+1}.
(b) There are 2^n membranes labelled by 2 such that each of them contains: a copy of object β, many copies of the garbage object $\natural$ (which they will not evolve in the rest of the computation), and copies of objects c_j whose presence means that clause C_j is true for the truth assignment encoded by that membrane.
– *Second Checking stage*: by applying rules from **7.7**, we check whether or not all clauses of the input formula φ are satisfied by some truth assignment encoded by a membrane labelled by 2. This stage takes exactly $p-1$ steps and at configuration $\mathcal{C}_{n+2p}$:
(a) The root membrane (labelled by 1) contains n copies of object α_{n+2p}.
(b) There are 2^n membranes labelled by 2 such that each of them contains: a copy of object β, many copies of the garbage object $\natural$ (which they will not evolve in the rest of the computation), and copies of objects d_j and c_j, in such manner that the truth assignment encoded by such membrane makes true φ if and only if contains some object d_p.
– *Output stage. Negative answer:* if the input formula is not satisfiable, then any rule from **7.8** is not applicable and from $\mathcal{C}_{n+2p}$ on, no rules are applied in the system except those from **7.1** until reaching a halting configuration at $\mathcal{C}_{2n+2p+1}$. Therefore, in this case, the system answers 0.
– *Output stage. Affirmative answer:* if the input formula is satisfiable, by applying rules from **7.8** some objects γ_0 are produced at membrane labelled by 1, Due to the semantics of these membrane systems, this stage takes exactly two steps. Thus, at configuration $\mathcal{C}_{n+2p+2}$ the multiplicity of γ_0 in the skin membrane equals to the number of truth assignment of variables $\{x_1, \ldots, x_n\}$ that makes true φ. Next, by applying rules from **7.9**, some objects γ_i, $0 \leq i \leq n-1$, with multiplicity 1 will be generated after, at most, $n-1$ computation steps. Then, at configuration $\mathcal{C}_{(n+2p+2)+n-1} = \mathcal{C}_{2n+2p+1}$ at the skin membrane we have n copies of object $\alpha_{2n+2p+1}$ and some objects γ_i, $0 \leq i \leq n-1$, with multiplicity 1. By applying the second rule from **7.9**, some objects $a_{i,0}$ with multiplicity 1 are produced at that membrane. In order to make deterministic the system, objects $a_{i,0}$ evolves until $a_{i,i}$ by applying the third rules from **7.9**. Finally, the system sends to the environment the right answer according to the results of the previous stage, by applying

rules from **7.10**, for instance, object $a_{i,i}$ is released to the environment as object a_i. This stage takes, at most, n computation steps. Specifically, if object $a_{i,0}$ appears in membrane 1 at configuration $\mathcal{C}_{2n+2p+2}$ then object a_i is sent out to the environment at $i+1$-th step of this stage.

5 Main results

Theorem 1. $\#\texttt{SAT} \in \mathbf{PCMS}_{\mathcal{DAM}^0_c(mcmp,+c,-d,-n)}$.

Proof. The family of P systems previously constructed verifies the following:

(a) Every system of the family $\mathbf{\Pi}$ belongs to $\mathcal{DAM}^0_c(mcmp,+c,-d,-n)$.
(b) The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines because for each $n,p \in \mathbb{N}$, the amount of resources needed to build $\Pi(\langle n,p \rangle)$ is of a polynomial order in n and p:
  - Size of the alphabet: is of the order $O(n^2 \cdot p^2)$.
  - Initial number of membranes: $2 \in \Theta(1)$.
  - Initial number of objects in membranes: $2np+2n+1 \in \Theta(n \cdot p)$.
  - Number of rules: is of the order $O(n^2 \cdot p^2)$.
  - Maximal number of objects involved in any rule: $3 \in \Theta(1)$.
(c) The pair (cod, s) of polynomial-time computable functions defined fulfill the following: for each input formula φ of the $\#\texttt{SAT}$ problem, $s(\varphi)$ is a natural number, $cod(\varphi)$ is an input multiset of the system $\Pi(s(\varphi))$, and for each $n \in \mathbb{N}$, $s^{-1}(n)$ is a finite set.
(d) The family $\mathbf{\Pi}$ is polynomially bounded: indeed, for each input formula φ of the $\#\texttt{SAT}$ problem, the P system $\Pi(s(\varphi)) + cod(\varphi)$ takes at most $n+2p$ computation steps in the case of the input formula is not satisfiable and, on the contrary, takes at most $3n+2p+2$ steps, n being the number of variables of φ and p the number of clauses.
(e) The family $\mathbf{\Pi}$ is sound and complete with regard to (X, cod, s): indeed, it is informally deduced from the overview of the computations previously described.

Therefore, the family $\mathbf{\Pi}$ of P systems previously constructed solves the $\#\texttt{SAT}$ problem in polynomial time in a uniform way.

Corollary 1. $\#\mathbf{P} \subseteq \mathbf{PCMS}_{\mathcal{DAM}^0_c(mcmp,+c,-d,-n)}$.

Proof. It suffices to note that the $\#\texttt{SAT}$ problem is a $\#\mathbf{P}$-complete problem, $\#\texttt{SAT} \in \mathbf{PCMS}_{\mathcal{DAM}^0_c(mcmp,+c,-d,-n)}$, and class $\mathbf{PCMS}_{\mathcal{DAM}^0_c(mcmp,+c,-d,-n)}$ is closed under polynomial-time reduction and under complement.

6 Conclusions

In order to provide a natural framework to solve counting problems in the context of Membrane Computing, a new class of membrane systems, called *counting* *membrane systems*, is presented in this paper. The new kind of models is inspired from counting Turing machines [19] and from recognizer membrane systems [12].

The computational efficiency of the new variant has been explored. Specifically, a polynomial-time and uniform solution to the #`SAT` problem, a well-known #**P**-complete problem, is provided by using a family of counting polarizationless P systems with active membranes, without dissolution rules and division rules for non-elementary membranes but where very restrictive cooperation (minimal cooperation and minimal production) in object evolution rules is allowed.

As future works we suggest to analyze the computational efficiency of counting membrane systems from the previous class but where minimal cooperation and minimal production only is considered for communication rules (maybe only send-in rules or only send-out rules) instead of object evolution rules, following the work initiated in [18]. Besides, it would be interesting, from a computational complexity point of view, to explore the ability to use separation rules (*distribution of objects*) instead of division rules (*replication of objects*) in counting membrane systems, from a computational complexity point of view.

References

1. A. Alhazov, L. Pan. Polarizationless P systems with active membranes. *Grammars*, **7** (2004), 141-159.
2. A. Alhazov, L. Pan, Gh. Păun. Trading polarizations for labels in P systems with active membranes. *Acta Informatica*, **41**, 2-3 (2004), 111-144.
3. A. Alhazov, L. Burtseva, S. Cojocaru, Y. Rogozhin. Solving **PP**-Complete and #**P**-Complete Problems by P Systems with Active Membranes. In D. Wolfe Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa (eds.). *Membrane Computing 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers. Lecture Notes in Computer Science*, **5391** (2009), 108-117.
4. M.R. Garey, D.S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.
5. M.A. 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.
6. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M.J. Pérez-Jiménez. Uniform solutions to SAT and Subset Sum by spiking neural P systems. *Natural Computing*, **8**, 4 (2009), 681-702.
7. Gh. Păun. Computing with membranes, *Journal of Computer and Systems Science*, **61**, 1 (2000), 108-143.
8. Gh. Păun. Attacking **NP**-complete problems. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou, C. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, 94-115.
9. Gh. Păun. P systems with active membranes: attacking **NP**-complete problems, *Journal of Automata, Languages and Combinatorics*, **6** (2001), 75-90.

10. Gh. Păun, M.J. Pérez-Jiménez, Gr. Rozenberg. Spike trains in spiking neural P systems. *International Journal of Foundations of Computer Science*, **17**, 4 (2006), 975-1002.
11. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez. Tissue P systems with cell division. *International Journal of Computers, Communications & Control*, Vol. **III**, 3 (2008), 295-303.
12. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, **2**, 3 (2003), 265-285.
13. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini. A polynomial complexity class in P systems using membrane division, *Journal of Automata, Languages and Combinatorics*, **11**, 4 (2006) 423-434.
14. P. Sosík, A. Rodríguez-Patón. Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, **73** (2007), 137–152.
15. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Polarizationless P systems with active membranes: Computational complexity aspects. *Journal of Automata, Languages and Combinatorics*, **21**, 1-2 (2016), 107–123.
16. L. Valencia-Cabrera, D. Orellana-Martín, A. Riscos-Núñez, M.J. Pérez-Jiménez. Minimal cooperation in polarizationless P systems with active membranes. In C. Graciani, Gh. Păun, D. Orellana-Martín, A. Riscos-Núñez, L. Valencia-Cabrera (eds.) *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*, 1-5 February, 2016, Sevilla, Spain, Fénix Editora, pp. 327-356.
17. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Reaching efficiency through collaboration in membrane systems: dissolution, polarization and cooperation. *Theoretical Computer Science*, in press, 2017.
18. L. Valencia-Cabrera, D. Orellana-Martín, M.A. Martínez-del-Amor, A. Riscos-Núñez, M.J. Pérez-Jiménez. Cooperation in transport of chemical substances: A complexity approach. *Fundamenta Informaticae*, in press, 2017.
19. L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, **8**, 2 (1979), 189-201.
20. G. Zhang, M.J. Pérez-Jiménez, M. Gheorghe. *Real-Life Modelling with Membrane Computing*. Series: Emergence, Complexity and Computation, Volume 25. Springer International Publishing, 2017, X + 367 pages.