Design Patterns for Efficient Solutions to NP-Complete Problems in Membrane Computing

Álvaro Romero-Jiménez and David Orellana-Martín

Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence, Universidad de Sevilla,
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
{romero.alvaro,dorellana}@us.es

Abstract. Many variants of P systems have the ability to generate an exponential number of membranes in linear time. This feature has been exploited to elaborate (theoretical) efficient solutions to **NP**-complete, or even harder, problems. A thorough review of the existent solutions shows the utilization of common techniques and procedures. The abstraction of the latter into design patterns can serve to ease and accelerate the construction of efficient solutions to new hard problems.

Keywords: Membrane computing · NP-complete problems
Efficient solutions · Design patterns · SAT problem

1 Introduction

All of us have seen ourselves, at one time or another of our lives, in the need to solve a "hard problem", whatever this may mean. Computational complexity is a branch of the theory of computation that makes this concept precise by defining in mathematical terms the notions of (decision) problems and of the amount of resources required for any mechanical procedure (algorithm) to solve them. For example, **NP**-complete problems are most of the time regarded to be hard (at least, for those who believe that $\mathbf{P} \neq \mathbf{NP}$). A historical review of many of the ideas from this theory can be found in [1].

Membrane computing is a branch of the natural computing field aiming to abstract computing models, called P systems, from the structure and the functioning of the living cell. It was initiated by Gh. Păun in 1998 [2] and it has quickly become a vigorous scientific discipline, as [3,4] testify.

P systems provide highly parallel and distributed devices. As **NP**-complete problems are intuitively those problems for which checking, but not finding, a solution can be done in feasible time, it was soon realized that (theoretical) efficient solutions to them could be accomplished by several variants of P systems [5]. The main feature of these variants is their ability to construct an exponential space in linear time. This is usually exploited to implement a brute

force search in which first all possible solutions to the problem are generated, and then all of them are verified in parallel at the same time.

The flourish of efficient solutions to **NP**-complete, or even harder, problems has led to the development of a rich computational complexity theory for membrane computing [6,7]. The purpose of this theory is to rigorously define what a solution to a decision problem is in the membrane computing domain, how to measure the resources spent by those solutions and when they can be said to be efficient.

In this theory the following key concepts are defined:

1. Solutions are families of recognizer P systems. These are systems verifying the following properties: there are two special objects, `yes` and `no`, in the working alphabet; all the computations must supply their result by sending out to the output region only one of these two objects; at the precise moment that this happens, the computation must stop.
2. The solutions can be uniform or semi-uniform. In the former case, the systems of the family own an input membrane, and each of them is able to solve all the instances of the problem of a specific "size" (as determined by a polynomial-time computable function s) when an encoding of the instance (computed by a polynomial-time computable function *cod*) is introduced in its input membrane. In the latter case, each instance of the problem is associated with a system of the family, that solves it without requiring additional information because the instance is directly encoded within its objects and rules.
3. A decision problem is said to be solvable in polynomial time by a family of recognizer P systems if the family: can be constructed in polynomial time (by a Turing machine); is polynomially bounded, meaning that there exists a polynomial on the size of the instances of the problem that bounds the number of steps performed by any computation of any of the systems of the family; is sound and complete with respect to the problem, meaning that for each instance of the problem an object `yes` is provided as the result of a computation if and only if the answer to the instance is positive.

The task of setting up an efficient solution to a hard problem is not an easy one. A judicious way to tackle it is to base ourselves on what others have already proved to work. In fact, in software engineering it is recommended practice to make use of the so-called design patterns to improve and speed up software development. Quoting Wikipedia:[1]

> *A software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.*

[1] Software design pattern, https://en.wikipedia.org/w/index.php?title=Software_design_pattern\&oldid=834346932 (last visited May 10, 2018).

A detailed analysis of many existent efficient solutions to hard problems that can be found in the membrane computing literature shows indeed that similar techniques and constructions have been utilized. This paper is devoted to describe some of those common patterns, with the aim of serving as a starting point for the non-experts to develope their own solutions.

The paper is organized as follows: in Sect. 2 some design patterns for membrane computing are described, namely the *exponential space* (Subsect. 2.1), the *all present* (Subsect. 2.2) and the *no if not yes* (Subsect. 2.3) patterns. It is also demonstrated how to implement them with different ingredients. Section 3 exemplifies the use of these design patterns by constructing distinct solutions to the SAT problem, namely one using polarizations and membrane division (Subsect. 3.1), one using dissolution and membrane creation (Subsect. 3.2), one using polarizations and membrane separation (Subsect. 3.3) and one using membrane division and minimal cooperation (Subsect. 3.4). The paper ends with some concluding remarks.

2 Design Patterns for Membrane Computing

One of the peculiarities of many of the variants of P systems considered so far in the literature is their ability to trade time for space. By this we mean that they are able to build an exponential space in polynomial, often linear, time. Although for objects it is pretty obvious, since a simple rewriting rule of the form $a \to a^2$ allows to double the number of objects a in one step, we are more interested in obtaining an exponential number of membranes. Exploiting this feature and the inherent parallelism of P systems, many **NP**-complete problems have been solved in polynomial time following a brute force approach: first, generate an exponential number of membranes each one containing a bunch of objects representing a potential solution to the problem; next, check in a parallel way if any of those is indeed a real solution; finally, provide an answer according to the results obtained.

The analysis of this scheme for the solutions discovers the use of these three patterns: the *exponential space* pattern, to generate an exponential number of membranes; the *all present* pattern, to test if all of the objects from a specific set are present in a membrane; and the *no if not yes* pattern, to be able to provide a negative answer when only the positive one can be checked.

In the next subsections, we describe each of these design patterns and show how to implement them with several features that can be found in P systems with active membranes. We will make use of the following types of rules:

1. Evolution rules $[a \to u]_l^c$, for changing the object a into the multiset u in a membrane with label l and, optionally, a charge c.
2. Communication rules $a_1[\ ]_l^{c_1} \to [a_2]_l^{c_2}$ and $[a_1]_l^{c_1} \to [\ ]_l^{c_2} a_2$, to send an object a_1 into or out from a membrane with label l. The object is rewritten to a_2 and the possible charge c_1 of the membrane changed to c_2.

3. Dissolution rules $[a_1]_l^c \to a_2$, to dissolve a membrane with label l that contains an object a_1 and, optionally, has a charge c. All the objects of the membrane goes to its parent membrane, object a_1 rewritten to a_2.
4. Membrane division rules $[a_1]_l^{c_1} \to [a_2]_l^{c_2}[a_3]_l^{c_3}$, to divide a membrane with label l and that contains an object a_1 and, optionally, has a charge c_1. The membrane divides in two new membranes each of them containing a copy of all its objects, with a_1 rewritten to a_2 and a_3, respectively, and the charge changed to c_2 and c_3, respectively.
5. Membrane creation rules $[a \to [u]_{l_2}^{c_2}]_{l_1}^{c_1}$, to create a membrane with label l_2, contents u and possible charge c_2 from an object a in a membrane with label l_1 and possible charge c_1. The objects from the latter membrane are not replicated into the membrane being created. To simplify the description of the patterns, we will omit the surrounding brackets when the membrane they are referring to is clear from the context.
6. Membrane separation rules $[a]_l^{c_1} \to [\Gamma_0]_l^{c_2}[\Gamma_1]_l^{c_3}$, to separate in two a membrane with label l and possible charge c_1. This separation is triggered by object a and sends all the other objects to one of the new membranes if they belong to Γ_0 and to the other membrane if they belong to Γ_1 (Γ_0 and Γ_1 must be a partition of the working alphabet). The new membranes have, optionally, charges c_2 and c_3, respectively.

2.1 The *Exponential Space* Pattern

The *exponential space* pattern allows to generate an exponential number of membranes in a linear number of steps. If we focus specifically on P systems with active membranes, the usual mechanism is to make use of a collection of objects, each of them triggering a rule that duplicates a membrane. If we arrange those objects in such a way that they appear sequentially in the membranes being created, the number of those membranes is doubled in each step. The final result is that after n steps we get 2^n membranes.

The *Exponential Space* Pattern with Membrane Division. To implement the *exponential space* pattern for P systems with active membranes we will make use of a sequence of objects to repeatedly divide the membranes in two. Each time that any of these objects has performed its task, it is changed into the next object in the sequence and put within both new membranes.

In order to write the pattern rules in an homogeneous way, it is customary to identify with subscripts the objects in the sequence. Thus, using objects $a_0, \ldots, a_n$, we can produce 2^n membranes with label l in n steps as follows:

Initial setup:	$[a_0]_l$
Rules:	$[a_i]_l \to [a_{i+1}]_l[a_{i+1}]_l, \quad 0 \leq i < n$

It is important to note that the semantics itself of the division rule imposes that all the objects other than the one triggering the rule have to be replicated into the new membranes. This eases the design of the subsequent operations,

which are usually required to be performed in parallel within each membrane. Compare it with the pattern for membrane separation, where we do not have this facility.

On the other hand, the fact that rules of the rewriting type are allowed to be run alongside division rules enables us to intercalate other operations with that of constructing the exponential space. This can even be boosted with the use of polarizations for the membranes, since they provide a mechanism to hold up the divisions of the membranes until the intercalated task has finished.

The *Exponential Space* Pattern with Membrane Creation. When dealing with P systems with membrane creation, in order to obtain an exponential space in linear time we face with the difficulty of being able to create only one membrane from the object triggering the rule. To circumvent this obstacle an exponential number of objects from which to create the membranes could be generated (for example, by rewriting rules of the form $a \to a^2$). But this solution has the disadvantage of making harder to set up the scenarios for other operations.

A better alternative is to exploit the semantics of membrane creation rules. Since when creating a membrane any number of objects can be placed within it, the number of objects can be doubled and distributed at the same time. Thus, with the help of objects $a_0, a'_0, \ldots, a_n, a'_n$, we can create 2^n membranes with label l in n steps as follows:

Initial setup:	$a_0 a'_0$
Rules:	$a_i \to [a_{i+1} a'_{i+1}]_l, \quad 0 \leq i < n$
	$a'_i \to [a_{i+1} a'_{i+1}]_l, \quad 0 \leq i < n$

Note that the result of the previous pattern is the creation, inside of the wrapping membrane that contains the initial setup, of a nested membrane structure whose depth is increased and whose number of most inner membranes is doubled with each application of the rules. If we are allowed to dissolve membranes, it is possible to obtain a flat membrane structure in the following manner:

Initial setup:	$a_0 a'_0$
Rules:	$a_i \to [a_{i+1} a'_{i+1} d]_l, \quad 0 \leq i < n-1$
	$a'_i \to [a_{i+1} a'_{i+1} d]_l, \quad 0 \leq i < n-1$
	$[d]_l \to \lambda$
	$a_{n-1} \to [a_n a'_n]_l$
	$a'_{n-1} \to [a_n a'_n]_l$

Again, by means of rewriting rules other operations can be performed at the same time as the exponential space is created. Also, the creation of the new membranes can be delayed with the aid of different techniques, such as the use of charges or the rotation of objects (for example, $a_{i,0} \to a_{i,1} \to \cdots \to a_{i,m} \to [a_{i+1,0} a'_{i+1,0}]_l$).

Finally, the objects required by the succeeding operations to be within the membranes with label l can be effortlessly put inside them by the last creation rules applied (the pattern above puts the objects a_n and a'_n, but this is only for simplicity of the description).

The *Exponential Space* Pattern with Membrane Separation. Separation rules, like division rules, provide two membranes from an initial one. This is the essential property allowing an efficient exponential space generation. However, the key difference is that, unlike with division rules, with separation rules no new objects are created, but the already existing ones are distributed into the new membranes. This means that the number of membranes that could be obtained is limited by the number of objects present in the P system at the beginning.

To overcome this restriction, membrane separation has to be combined with another mechanism, such as evolution rules, providing a way to generate an exponential number of objects in linear time. Also, to accommodate the separation, the pattern uses pairs of objects with identical role. Thus, we need objects $a_0, a'_0, \tilde{a}_0, \tilde{a}'_0, \ldots, a_n, a'_n, \tilde{a}_n, \tilde{a}'_n$ to create 2^n membranes with label l in n steps:

Initial setup:	$[a_0\tilde{a}_0]_l$
Rules:	$[\tilde{a}_i \to a_{i+1}a'_{i+1}\tilde{a}_{i+1}\tilde{a}'_{i+1}]_l, \quad 0 \leq i < n$
	$[\tilde{a}'_i \to a_{i+1}a'_{i+1}\tilde{a}_{i+1}\tilde{a}'_{i+1}]_l, \quad 0 \leq i < n$
	$[a_i]_l \to [\Gamma_0]_l[\Gamma_1]_l, \quad 0 \leq i < n$
	$[a'_i]_l \to [\Gamma_0]_l[\Gamma_1]_l, \quad 0 \leq i < n$

where $a_i, \tilde{a}_i \in \Gamma_0$ and $a'_i, \tilde{a}'_i \in \Gamma_1$.

If we would like a specific object, say for example b, to remain within all the created membranes, the same duplication technique as for objects a_i could be employed:

Initial setup:	$[a_0\tilde{a}_0 b]_l$
Rules:	$[\tilde{a}_i \to a_{i+1}a'_{i+1}\tilde{a}_{i+1}\tilde{a}'_{i+1}]_l, \quad 0 \leq i < n$
	$[\tilde{a}'_i \to a_{i+1}a'_{i+1}\tilde{a}_{i+1}\tilde{a}'_{i+1}]_l, \quad 0 \leq i < n$
	$[b \to bb']_l$
	$[b' \to bb']_l$
	$[a_i]_l \to [\Gamma_0]_l[\Gamma_1]_l, \quad 0 \leq i < n$
	$[a'_i]_l \to [\Gamma_0]_l[\Gamma_1]_l, \quad 0 \leq i < n$

where $a_i, \tilde{a}_i, b \in \Gamma_0$ and $a'_i, \tilde{a}'_i, b' \in \Gamma_1$.

2.2 The *All Present* Pattern

Almost all of the efficient solutions to hard problems found in the membrane computing bibliography follow a brute force approach. They first generate all the possible solutions to the problem and then check if any of them is indeed a solution. For this it is usually demanded that certain requirements are met.

The *all present* pattern arises in this context to confirm the fulfillment of all of these requirements. For that it assumes that the accomplishment of the i-th requirement is signaled by the presence of an object r_i in a certain membrane. It then takes care of verifying that for all of the requirements there is at least one associated object in that membrane.

The *All Present* Pattern with Polarizations. Let us suppose that we want to confirm if in a certain membrane with label l there is at least one copy of all the objects $r_1, \ldots, r_n$. When membrane charges are available, the alternation of two of them provides a control mechanism to iterate a two step process: when the membrane is, for example, positively charged, the presence of the object r_1 is checked; conversely, when the membrane is negatively charged, the subscripts of the objects r_i are decreased by one and a certain counter is increased by one. The absence of the object r_1 prevents the charge of the membrane to change to negative and so the counter gets stalled.

By representing the counter by the objects $c_0, \ldots, c_n$, the pattern takes the following form:

Initial setup:	$[c_0 r_{i_1} \ldots r_{i_k}]_l^+$
Rules:	$[r_1]_l^+ \to [\,]_l^- r_1$
	$[r_i \to r_{i-1}]_l^-, \quad 1 \leq i \leq n$
	$[c_i \to c_{i+1}]_l^-, \quad 0 \leq i < n$
	$r_1[\,]_l^- \to [r_0]_l^+$
	$[c_n]_l^+ \to [\,]_l^+ \texttt{yes}$

Just as it is described above, the pattern sends out of the membrane an object `yes` to acknowledge the presence of all the objects r_i. This answer can be obviously changed to another one more suitable for the purpose of the P system being designed.

The *All Present* Pattern with Membrane Creation. When working with polarizationless P systems it is necessary to look for another mechanisms allowing the two step process of checking for the presence of the objects and increasing the counter when they are detected. Membrane creation provides one of them, given place to the following pattern:

Initial setup:	$[c_0 r_{i_1} \ldots r_{i_k}]_l$
Rules:	$r_i \to [\,]_i, \quad 1 \leq i \leq n$
	$c_{i-1}[\,]_i \to [c_i]_i, \quad 1 \leq i \leq n$
	$[c_i]_i \to [\,]_i c_i, \quad 1 \leq i \leq n$
	$[c_n]_l \to [\,]_l \texttt{yes}$

This pattern acknowledges the presence of an object r_i by creating a membrane with label i. The counter traverses these membranes in order, simply by

entering and immediately exiting them. If any object r_i is missing, no membrane with label i is created and the counter gets stalled.

Again the final answer `yes` when all the objects are present is arbitrary and can be changed to another more suitable one.

The *All Present* Pattern with Cooperation. When objects can cooperate in evolution rules, verifying that all objects r_i are within the target membrane is a triviality.

Initial setup:	$[c_0 r_{i_1} \dots r_{i_k}]_l$
Rules:	$[c_0\, r_1 \dots r_n \to c_n]_l$
	$[c_n]_l \to [\,]_l$`yes`

Since complete cooperation is so powerful, we often restrict to minimal cooperation, where the left-hand side of an evolution rule is restricted to length at most two. In this case it is also easy to build an *all present* pattern:

Initial setup:	$[c_0 r_{i_1} \dots r_{i_k}]_l$
Rules:	$[c_i r_{i+1} \to c_{i+1}]_l, \quad 0 \le i < n$
	$[c_n]_l \to [\,]_l$`yes`

2.3 The *No If Not Yes* Pattern

To solve a decision problem means to provide, for any of its instances, either the answer *yes* or the answer *no*. When designing a solution to a hard problem by means of a brute force search, it usually happens that the P system is able to signal the true solutions from the candidate ones that have been produced. Thus, since those signals can be effortlessly detected, it is easy to manage the positive instances of the problem. The difficulty arises with the negative instances, for which we have to guarantee that no signal will be generated, meaning that all candidate solutions have been discarded.

The *no if not yes* pattern comes to the rescue. The idea is very simple: we assume that the exact computation step when the signal would be detected can be figured out, what is almost always true. The pattern handles a counter that is increased at each step of the computation. If a signal is detected, the object `yes` is sent out to the environment and the computation is halted, which includes, of course, stopping the counter. Otherwise, the counter will go beyond the step where the signals should have been detected, so we can safely send the object `no` out to the environment and halt the computation.

The *No If Not Yes* Pattern with Polarizations. To provide a plain implementation of this pattern in P systems with polarizations, we will presuppose that the signal objects are collected within the skin membrane and that the charge of this membrane is initially neutral.

Initial setup:	$[c_0]^0_{skin}$
	signal objects s appear in the skin at step n

Rules: $[c_i \to c_{i+1}]^0_{skin}, \quad 0 \le i \le n$

$[s]^0_{skin} \to [\,]^+_{skin}\texttt{yes}$

$[c_{n+1}]^0_{skin} \to [\,]^0_{skin}\texttt{no}$

Therefore, the computation is supposed to stop after step $n+2$. Any solution to the instance of the problem makes an object s to enter the skin membrane at step n. This, in turn, makes the skin membrane to send an object `yes` out to the environment and to become positively polarized, what blocks the counter. In the case that the instance has no solution, the skin membrane remains neutrally polarized, and with the aid of the counter an object `no` is sent out to the environment.

The *No If Not Yes* Pattern with Dissolution. When instead of polarizations it is dissolution what is available, the impossibility of dissolving the skin membrane compels us to work in an ancillary membrane within it. The pattern is simply translated as follows: the appearance of a signal object s at step n dissolves that membrane, preventing the counter to keep on advancing, and releasing an object `yes` into the skin; if the above does not happen, the counter dissolves the membrane at step $n+1$, releasing an object `no` into the skin; the latter just has to send out whichever of the answer objects is received, and the computation then stops.

Initial setup: $[[c_0]_l]_{skin}$

signal objects s appear in membrane l at step n

Rules: $[c_i \to c_{i+1}]_l, \quad 0 \le i \le n$

$[s]_l \to \texttt{yes}$

$[c_{n+1}]_l \to \texttt{no}$

$[\texttt{yes}]_{skin} \to [\,]_{skin}\texttt{yes}$

$[\texttt{no}]_{skin} \to [\,]_{skin}\texttt{no}$

The *No If Not Yes* Pattern with Cooperation. With (minimal) cooperation as the control mechanism, the pattern can be implemented by combining an auxiliary object able to react with the signal objects and a counter to account for their absence. The appearance of any of the signal objects at step n removes the auxiliary object, releasing an object `yes` into the skin. Otherwise it remains in the system and cooperate with the counter to release an object `no` into the skin. Whichever the answer object received is sent out to the environment by the skin and the computation then stops.

Initial setup: $[c_0\, c]_{skin}$

signal objects s appear in the skin at step n

Rules: $[c_i \to c_{i+1}]_{skin}, \quad 0 \le i < n+1$

$[cs \to \texttt{yes}]_{skin}$

$[c_{n+1}c \to \texttt{no}]_{skin}$

$$[\texttt{yes}]_{skin} \rightarrow [\ ]_{skin}\texttt{yes}$$
$$[\texttt{no}]_{skin} \rightarrow [\ ]_{skin}\texttt{no}$$

3 Practical Examples

To illustrate the use of the design patterns previously described, in this section we elaborate some solutions to a hard problem, namely the propositional satisfiability problem. The SAT problem was the first one proved to be **NP**-complete [8], and can be stated as follows: *given a Boolean formula in conjunctive normal form (CNF), determine whether or not it is satisfiable, that is, whether there exists an assignment to its variables on which it evaluates to true.*

There are already numerous efficient solutions to SAT in the membrane computing literature, devised in different frameworks, for example, the ones in [6,7,9–12] and many more. Taking inspiration from these solutions, in the subsections coming next we exhibit three new solutions to SAT, with the focus on making apparent how they integrate the design patterns.

These new solutions will be provided within the framework of P systems with active membranes, first introduced by Păun [5] as a type of P systems that, abstracting the process of cell mitosis, provide rules for making the membranes to divide. Along with this kind of operations, these systems can also: make an object evolve within a membrane; send an object into or out of a membrane; dissolve a membrane. Besides, each of the membranes has an associated charge –positive, negative or neutral– subject to changes by the applications of the rules. Finally, rules are triggered by a unique object, disallowing their cooperation.

Since the computational power of this model has been shown to be as high as to be able to efficiently solve hard problems, several variants have been considered in an attempt to determine what borderlines in efficiency provides each of its ingredients. Thus, with respect to the mechanism generating exponential space, membrane creation (abstracting cell autopoiesis) and membrane separation (abstracting cell meiosis) have been studied. The role of polarization, dissolution and cooperation, and even of membrane labels, have also been analyzed. A survey of diverse results that have been obtained can be found in [7]. Our solutions will differ in the ingredients utilized, to show how the design patterns are able to adapt to distinct conditions.

For a better understanding of the solutions, we refer to [7] for the semantics of the variants with membrane division and creation and to [13] for the semantics of the variant with membrane separation. Some details to take into account: the working mode is maximal parallelism, meaning that the rules are applied to all objects and all membranes at the same time in parallel; for active membranes with division or separation rules, the semantics states that each membrane can be affected by at most one rule of the send in, send out, dissolution and division/separation type (all types but evolution); when creation rules are used, the standard semantics is polarizationless and, besides, it only restricts the simultaneous application of dissolution rules; it is assumed that to perform

a computation step, the system applies first the evolution rules and then all the others.

The solutions presented here are uniform: for every $m \geq 1$ and every $n \geq 1$ they construct a P system $\Pi(m,n)$ for dealing with any Boolean formula in CNF with m clauses and n variables. The input alphabet of this P system will always be

$$\Sigma(m,n) = \{x_{i,j}, \bar{x}_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n\}$$

Given a Boolean formula $\varphi = C_1 \wedge \cdots \wedge C_m$, before starting a computation of $\Pi(m,n)$ an object $x_{i,j}$ (respectively, $\bar{x}_{i,j}$) has to be put inside its input membrane for any clause C_i and any variable x_j such that x_j is in C_i (respectively, $\neg x_j$ is in C_i). The computation will then carry out these operations:

- *Generate* all the possible assignments for φ: the *exponential space* pattern will be useful here.
- *Compute* for each of the assignments which clauses take a true value: this can be done after the previous operation ends, but usually also at the same time, as we will see.
- *Check* for every assignment, in parallel, if all the clauses take a true value: the *all present* pattern is clearly suitable for this task.
- *Decide* if the formula φ is satisfiable or not: the *no if not yes* pattern will be needed for this.

3.1 Solution to SAT Using Polarizations and Membrane Division

Given φ a Boolean formula in CNF with m clauses and n variables, the recognizer P system $\Pi(m,n)$ of the solution that decides if φ is satisfiable or not is constructed in polynomial time from:

- A working alphabet:

$$\begin{aligned}\Sigma(m,n) &\cup \{d_i \mid 0 \leq i \leq 2n\} \\ &\cup \{r_{i,j} \mid 0 \leq i \leq m, 1 \leq j \leq n+1\} \\ &\cup \{c_i \mid 0 \leq i \leq 3n+2m+3\} \\ &\cup \{d, \texttt{yes}, \texttt{no}\}\end{aligned}$$

- A set $\{1,2\}$ of labels for the membranes.
- An initial membrane structure and contents of the membranes $[c_0[d\,d_0]_2^0]_1^0$ (the input membrane is the one labelled 2).
- A set of rules $\mathit{GenerateCompute} \cup \mathit{Sync} \cup \mathit{Check} \cup \mathit{Decide}$ where:
  - The rules in $\mathit{GenerateCompute}$ are:
    1. $[d_i]_2^0 \rightarrow [d_{i+1}]_2^+ [d_{i+1}]_2^-$ $\qquad 0 \leq i < n$
    2. $[x_{i,1} \rightarrow r_{i,1}]_2^+$, $[\bar{x}_{i,1} \rightarrow \lambda]_2^+$, $[x_{i,1} \rightarrow \lambda]_2^-$, $[\bar{x}_{i,1} \rightarrow r_{i,1}]_2^-$ $\qquad 1 \leq i \leq m$

3. $[x_{i,j} \to x_{i,j-1}]_2^+,\ [\bar{x}_{i,j} \to \bar{x}_{i,j-1}]_2^+$ $\quad 1 \leq i \leq m, 2 \leq j \leq n$
   $[x_{i,j} \to x_{i,j-1}]_2^-,\ [\bar{x}_{i,j} \to \bar{x}_{i,j-1}]_2^-$
4. $[d \to d^2]_2^0$
   $[d]_2^+ \to [\ ]_2^0 d,\ [d]_2^- \to [\ ]_2^0 d$

- The rules in *Sync* are:

5. $[r_{i,j} \to r_{i,j+1}]_2^0 \quad 1 \leq i \leq m, 1 \leq j < n+1$
6. $[d_i \to d_{i+1}]_2^0 \quad n \leq i < 2n-1$
7. $[d_{2n-1} \to d_{2n} c_0]_2^0$
8. $[d_{2n}]_2^0 \to [\ ]_2^+ d_{2n}$

- The rules in *Check* are:

9. $[r_{1,n+1}]_2^+ \to [\ ]_2^- r_{1,n+1}$
10. $[r_{i,n+1} \to r_{i-1,n+1}]_2^- \quad 1 \leq i \leq m$
11. $[c_i \to c_{i+1}]_2^- \quad 0 \leq i < m$
12. $r_{1,n+1}[\ ]_2^- \to [r_{0,n+1}]_2^+$
13. $[c_m]_2^+ \to [\ ]_2^+ \texttt{yes}$

- The rules in *Decide* are:

14. $[c_i \to c_{i+1}]_1^0 \quad 0 \leq i < 3n+2m+3$
15. $[\texttt{yes}]_1^0 \to [\ ]_1^+ \texttt{yes}$
16. $[c_{3n+2m+3}]_1^0 \to [\ ]_1^0 \texttt{no}$

In this P system rules number 1 implement the *exponential space* pattern, with the membranes dividing when neutrally charged. Charges + and − are used to sequentially track the different values assigned to the variables. This way, rules number 2–3 are able to determine, along the process of generating the 2^n membranes representing the different assignments, which clauses are taking the true value. Both membrane division and clause value determination are intercalated by means of rules number 4.

Since only three charges are available and both the *exponential space* and the *all present* pattern make use of them, we need a mechanism preventing the patterns to interfere. A regular technique is to append subscripts to the common objects, allowing them to differentiate. For $\Pi(m,n)$, although all objects $r_{i,j}$ represent that the i-th clause is true, the generation/computation stage works with those with $0 \leq j \leq n$, whereas the checking stage works with those with $j = n+1$. One caveat is that we have to assure that all the objects are the correct ones before letting the checking stage start, and that is the duty of rules number 5–8.

Next, rules number 9–13 carry out the *all present* pattern in each of the 2^n internal membranes, to verify if for their associated assignments all the clauses take the true value and send out a `yes` object to the skin membrane if so. Finally, rules number 14–16 provide the definitive answer by means of a *no if not yes* pattern.

3.2 Solution to SAT Using Dissolution and Membrane Creation

Given φ a Boolean formula in CNF with m clauses and n variables, the recognizer P system $\Pi(m,n)$ of the solution that decides if φ is satisfiable or not is constructed in polynomial time from:

– A working alphabet:

$$\begin{aligned}\Sigma(m,n) \cup &\{x_{i,j,l}, \bar{x}_{i,j,l} \mid 1 \leq i \leq m, 1 \leq j \leq n, l = t, f\}\\ &\cup \{d_{i,t}, d_{i,f} \mid 0 \leq i \leq n\}\\ &\cup \{r_i, r_{i,l} \mid 1 \leq i \leq m, l = t, f\}\\ &\cup \{c_i \mid 0 \leq i \leq 3n + 2m + 5\}\\ &\cup \{\texttt{yes}, \texttt{no}\}\end{aligned}$$

– A set $\{a, b, c, t, f, 1, \ldots, m\}$ of labels for the membranes.
– An initial membrane structure and contents of the membranes $[[c_0 d_{0,t} d_{0,f}]_b]_a$ (the input membrane is the one labelled b).
– A set of rules $\mathit{GenerateCompute} \cup \mathit{Link} \cup \mathit{Check} \cup \mathit{Decide}$ where:
  - The rules in *GenerateCompute* are:

    1. $[d_{0,t} \to [d_{1,t} d_{1,f}]_t]_b$
       $[d_{0,f} \to [d_{1,t} d_{1,f}]_f]_b$
    2. $[d_{i,t} \to [d_{i+1,t} d_{i+1,f}]_t]_l$
       $[d_{i,f} \to [d_{i+1,t} d_{i+1,f}]_f]_l$ $\quad 1 \leq i < n, l = t, f$
    3. $x_{i,1}[\,]_t \to [r_i]_t,\ \bar{x}_{i,1}[\,]_f \to [r_i]_f \quad 1 \leq i \leq m$
    4. $[x_{i,j} \to x_{i,j,t} x_{i,j,f}]_l$
       $[\bar{x}_{i,j} \to \bar{x}_{i,j,t} \bar{x}_{i,j,f}]_l$
       $x_{i,j,l}[\,]_l \to [x_{i,j-1}]_l$
       $\bar{x}_{i,j,l}[\,]_l \to [\bar{x}_{i,j-1}]_l$ $\quad 1 \leq i \leq m, 2 \leq j \leq n, l = t, f$
    5. $[r_i \to r_{i,t} r_{i,f}]_l$
       $r_{i,l}[\,]_l \to [r_i]_l$ $\quad 1 \leq i \leq m, l = t, f$

  - The rules in *Link* are:

    6. $[d_{n,t} \to [c_0]_c]_l \quad l = t, f$
    7. $r_{i,t}[\,]_c \to [r_i]_c \quad 1 \leq i < m$

  - The rules in *Check* are:

    8. $[r_i \to [\,]_i]_c \quad 1 \leq i \leq m$
    9. $c_{i-1}[\,]_i \to [c_i]_i \quad 1 \leq i \leq m$
    10. $[c_i]_i \to [\,]_i c_i \quad 1 \leq i \leq m$
    11. $[c_m]_c \to [\,]_c \texttt{yes}$
    12. $[\texttt{yes}]_l \to [\,]_l \texttt{yes} \quad l = t, f$

  - The rules in *Decide* are:

    13. $[c_i \to c_{i+1}]_b \quad 0 \leq i < 3n + 2m + 5$

14. $[\texttt{yes}]_b \rightarrow \texttt{yes}$
15. $[c_{3n+2m+5}]_b \rightarrow \texttt{no}$
16. $[\texttt{yes}]_a \rightarrow [\,]_a\texttt{yes}$
17. $[\texttt{no}]_a \rightarrow [\,]_a\texttt{no}$

In this P system rules number 1–2 implement the *exponential space* pattern to create a nested membrane structure, each of whose inner membranes stands for a different assignment to the variables of φ. As this membrane structure is created, but at a slower pace, the objects $x_{i,j}$ that represent this formula traverse it, computing at the same time which of the clauses makes true, which is signified by the objects r_i. Rules number 3–5 are in charge of this, including the use of the object duplication technique to be able to transmit a copy of the objects from each membrane to both of the membranes created inside it.

Rules number 6–7 link the generation/computation stage with the checking one, preventing the latter to start until the whole nested membrane structure has been constructed. They then create inside each of the internal membranes a new membrane where the *all present* pattern, implemented by rules number 8–11, is used to verify if all the clauses are true. Note, however, that objects r_i arrive at different times, but this does not affect the pattern performance. Also, the affirmative answers traverse, by means of rule number 12, the membrane structure back until arriving to membrane b.

Finally, rules number 13–17 use the *no if not yes* pattern to provide the definitive answer. Here, the time needed by the objects $x_{i,j}$ and r_i to travel forward and by the objects `yes` to travel backward through the membrane structure has been taken into account before concluding that the formula is unsatisfiable.

3.3 Solution to SAT Using Polarizations and Membrane Separation

Given φ a Boolean formula in CNF with m clauses and n variables, the recognizer P system $\Pi(m,n)$ of the solution that decides if φ is satisfiable or not is constructed in polynomial time from:

– A working alphabet $\Gamma_1 \cup \Gamma_2$ with:

$$\begin{aligned}\Gamma_1 =&\{x_{i,j}, \bar{x}_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n\}\\ &\cup \{d_i \mid 0 \leq i \leq 2n+1\}\\ &\cup \{\tilde{d}_i \mid 0 \leq i \leq n\}\\ &\cup \{r_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n+1\}\\ &\cup \{c_i \mid 0 \leq i \leq 3n+2m+4\}\\ &\cup \{d, \texttt{yes}, \texttt{no}\}\end{aligned}$$

and

$$\begin{aligned}\Gamma_2 =&\{x'_{i,j}, \bar{x}'_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n\}\\ &\cup \{d'_i \mid 0 \leq i \leq 2n+1\}\end{aligned}$$

$$\cup \{\tilde{d}'_i \mid 0 \leq i \leq n\}$$
$$\cup \{r'_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n+1\}$$
$$\cup \{d'\}$$

– A set $\{1, 2\}$ of labels for the membranes.
– An initial membrane structure and contents of the membranes $[c_0[d_0\tilde{d}_0 d]^0_2]^0_1$ (the input membrane is the one labelled 2).
– A set of rules $\mathit{GenerateCompute} \cup \mathit{Sync} \cup \mathit{Check} \cup \mathit{Decide}$ where:
  • The rules in $\mathit{GenerateCompute}$ are:

1. $[d_i]^0_2 \to [\Gamma_0]^+_2[\Gamma_1]^-_2$
   $[d'_i]^0_2 \to [\Gamma_0]^+_2[\Gamma_1]^-_2$ $\quad 0 \leq i < n$
2. $[\tilde{d}_i \to d_{i+1}d'_{i+1}\tilde{d}_{i+1}\tilde{d}'_{i+1}]^0_2$
   $[\tilde{d}'_i \to d_{i+1}d'_{i+1}\tilde{d}_{i+1}\tilde{d}'_{i+1}]^0_2$ $\quad 0 \leq i < n$
3. $[x_{i,j} \to x_{i,j}x'_{i,j}]^0_2,\ [x'_{i,j} \to x_{i,j}x'_{i,j}]^0_2$
   $[\bar{x}_{i,j} \to \bar{x}_{i,j}\bar{x}'_{i,j}]^0_2,\ [\bar{x}'_{i,j} \to \bar{x}_{i,j}\bar{x}'_{i,j}]^0_2$ $\quad 1 \leq i \leq m, 1 \leq j \leq n$
   $[r_{i,j} \to r_{i,j+1}r'_{i,j+1}]^0_2,\ [r'_{i,j} \to r_{i,j+1}r'_{i,j+1}]^0_2$
4. $[d \to d^2d'^2]^0_2,\ [d' \to d^2d'^2]^0_2$
5. $[x_{i,1} \to r_{i,1}]^+_2,\ [\bar{x}_{i,1} \to \lambda]^+_2$
   $[x'_{i,1} \to r_{i,1}]^+_2,\ [\bar{x}'_{i,1} \to \lambda]^+_2$
   $[x_{i,1} \to \lambda]^-_2,\ [\bar{x}_{i,1} \to r_{i,1}]^-_2$ $\quad 1 \leq i \leq m$
   $[x'_{i,1} \to \lambda]^-_2,\ [\bar{x}'_{i,1} \to r_{i,1}]^-_2$
6. $[x_{i,j} \to x_{i,j-1}]^+_2,\ [\bar{x}_{i,j} \to \bar{x}_{i,j-1}]^+_2$
   $[x'_{i,j} \to x'_{i,j-1}]^+_2,\ [\bar{x}'_{i,j} \to \bar{x}'_{i,j-1}]^+_2$
   $[x_{i,j} \to x_{i,j-1}]^-_2,\ [\bar{x}_{i,j} \to \bar{x}_{i,j-1}]^-_2$ $\quad 1 \leq i \leq m, 2 \leq j \leq n$
   $[x'_{i,j} \to x'_{i,j-1}]^-_2,\ [\bar{x}'_{i,j} \to \bar{x}'_{i,j-1}]^-_2$
7. $[d]^+_2 \to [\ ]^0_2 d,\ [d]^-_2 \to [\ ]^0_2 d$
   $[d']^+_2 \to [\ ]^0_2 d',\ [d']^-_2 \to [\ ]^0_2 d'$

  • The rules in Sync are:

8. $[r_{i,j} \to r_{i,j+1}]^0_2,\ [r'_{i,j} \to r'_{i,j+1}]^0_2$ $\quad 1 \leq i \leq m, 1 \leq j < n+1$
9. $[r'_{i,n+1} \to r_{i,n+1}]^0_2$ $\quad 1 \leq i \leq m$
10. $[d_i \to d_{i+1}]^0_2,\ [d'_i \to d'_{i+1}]^0_2$ $\quad n \leq i < 2n$
11. $[d_{2n} \to d_{2n+1}c_0]^0_2,\ [d'_{2n} \to d'_{2n+1}c_0]^0_2$
12. $[d_{2n+1}]^0_2 \to [\ ]^+_2 d_{2n+1},\ [d'_{2n+1}]^0_2 \to [\ ]^+_2 d'_{2n+1}$

  • The rules in Check are:

13. $[r_{1,n+1}]^+_2 \to [\ ]^-_2 r_{1,n+1}$
14. $[r_{i,n+1} \to r_{i-1,n+1}]^-_2$ $\quad 1 \leq i \leq m$
15. $[c_i \to c_{i+1}]^-_2$ $\quad 0 \leq i < m$

16. $r_{1,n+1}[\,]_2^- \to [r_{0,n+1}]_2^+$
17. $[c_m]_2^+ \to [\,]_2^+\texttt{yes}$

- The rules in *Decide* are:

18. $[c_i \to c_{i+1}]_1^0 \qquad 0 \leq i < 3n + 2m + 4$
19. $[\texttt{yes}]_1^0 \to [\,]_1^+\texttt{yes}$
20. $[c_{3n+2m+4}]_1^0 \to [\,]_1^0\texttt{no}$

In this P system rules number 1–2 implement the *exponential space* pattern for membrane separation, whereas rules number 3–4 include the standard object duplication technique to preserve the objects when separating the membranes. For the rest, the behaviour of the system is analogous to the one from the solution with polarizations and membrane division in Subsect. 3.1. Just only rule number 9 is added to get rid of the prime objects, so that we do not have to duplicate the rules for the subsequent operations.

3.4 Solution to SAT Using Minimal Cooperation (with Minimal Production) and Membrane Division

Given φ a Boolean formula in CNF with m clauses and n variables, the recognizer P system $\Pi(m, n)$ of the solution that decides if φ is satisfiable or not is constructed in polynomial time from:

– A working alphabet:

$$\begin{aligned}\Sigma^*(m,n) \cup & \{a_{i,k} \mid 1 \leq i \leq n, 1 \leq k \leq i\} \\ & \cup \{t_{i,k}, f_{i,k} \mid 1 \leq i \leq n, i \leq k \leq n+p+1\} \\ & \cup \{\beta_k \mid 0 \leq k \leq n+2p+2\} \\ & \cup \{c_j \mid 1 \leq j \leq m\} \\ & \cup \{d_j \mid 0 \leq j \leq m\} \\ & \cup \{T_i, F_i \mid 1 \leq i \leq n\} \\ & \cup \{x_{i,j,k}, \bar{x}_{i,j,k}, x^*_{i,j,k} \mid 0 \leq i \leq n, 1 \leq j \leq m, 1 \leq k \leq n+p\} \\ & \cup \{\alpha, \texttt{no}, \texttt{yes}\}\end{aligned}$$

– A set $\{1, 2\}$ of labels for the membranes.
– An initial membrane structure and contents of the membranes $[\alpha\, \beta_0 [d_0\ a_{i,1}\, T_i^p\, F_i^p]_2]_1$ (the input membrane is the one labelled 2).
– A set of rules $\mathit{Generate} \cup \mathit{Remove} \cup \mathit{Compute} \cup \mathit{Check} \cup \mathit{Decide}$ where:
  - The rules in *Generate* are:

1. $[a_{i,i}]_2 \to [t_{i,i}]_2[f_{i,i}]_2 \qquad 1 \leq i \leq n$
2. $[a_{i,k} \to a_{i,k+1}]_2 \qquad 2 \leq i \leq n, 1 \leq k \leq i-1$
3. $[t_{i,k} \to t_{i,k+1}]_2$ $[f_{i,k} \to f_{i,k+1}]_2 \qquad 1 \leq i \leq n-1, i \leq k \leq n-1$

- The rules in *Remove* are:

4. $[t_{i,k}\, F_i \to t_{i,k+1}]_2$
   $[f_{i,k}\, T_i \to f_{i,k+1}]_2$ $\quad 1 \leq i \leq n, n \leq k \leq n+p-1$

- The rules in *Compute* are:

5. $[x_{i,j,k} \to x_{i,j,k+1}]_2$
   $[\bar{x}_{i,j,k} \to \bar{x}_{i,j,k+1}]_2$ $\quad 1 \leq i \leq n, 1 \leq j \leq p, 0 \leq k \leq n+p-1$
   $[x^*_{i,j,k} \to x^*_{i,j,k+1}]_2$

6. $[T_i\, x_{i,j,n+p} \to c_j]_2$
   $[T_i\, \bar{x}_{i,j,n+p} \to \#]_2$
   $[T_i\, x^*_{i,j,n+p} \to \#]_2$
   $[T_i\, x_{i,j,n+p} \to \#]_2$ $\quad 1 \leq i \leq n, 1 \leq j \leq p$
   $[T_i\, \bar{x}_{i,j,n+p} \to c_j]_2$
   $[T_i\, x^*_{i,j,n+p} \to \#]_2$

- The rules in *Check* are:

7. $[d_i\, c_{i+1} \to d_{i+1}]_2$ $\quad 1 \leq i \leq m-1$

- The rules in *Decide* are:

8. $[\beta_i \to \beta_{i+1}]_2$ $\quad 0 \leq i \leq n+2p+1$
9. $[d_m]_2 \to [\,]_2 d_m$
10. $[\alpha\, d_m \to \texttt{yes}]_2$
11. $[\beta_{n+2p+2}\, \alpha \to \texttt{no}]_2$
12. $[\texttt{yes}]_1 \to [\,]_1 \texttt{yes}$
13. $[\texttt{no}]_1 \to [\,]_1 \texttt{no}$

Note that for this solution a different *input alphabet* is needed:

$$\Sigma^*(m,n) = \{x_{i,j,0}, \bar{x}_{i,j,0}, x^*_{i,j,0} \mid 1 \leq i \leq n, 1 \leq j \leq m\}$$

Then, given a Boolean formula $\varphi = C_1 \wedge \cdots \wedge C_m$, before starting a computation of $\Pi(m,n)$ an object $x_{i,j}$ (respectively, $\bar{x}_{i,j}$) has to be put inside its input membrane for any clause C_j and any variable x_i such that x_i is in C_j (respectively, $\neg x_i$ is in C_j). If neither x_i nor $\neg x_i$ appear in C_j, then an object $x^*_{i,j,0}$ has to be put inside its input membrane.

In this P system rules number 1–2 implement the *exponential space* pattern with division rules. This way, rules number 3 are able to synchronize objects $t_{i,k}$ and $f_{i,k}$ for the next stage. Rules number 4 remove the wrong "truth assignment objects" to keep only the objects that represent the real truth assignment of such a membrane. Rules number 5–6 synchronize $cod(\varphi)$ with the rest of the system and compute the clauses that are satisfied by the corresponding truth assignment.

Rules number 7 implement the *all present* pattern with minimal cooperation, and finally rules number 8–10 return the answer by means of the *no if not yes* pattern.

4 Conclusions

Many variants of P systems are powerful enough as to permit the existence of (theoretical) efficient solutions to hard problems. Indeed, a plethora of such solutions to **NP**-complete, **PP**-complete, **PSPACE**-complete problems and the like can be found in the membrane computing literature.

A careful analysis of those solutions reveals that a number of techniques and constructions are repeatedly applied. Following what is recommended practice in software engineering, it is advisable to devise abstractions of those techniques and constructions. These design patterns may speed up the elaboration of solutions to new problems and increase the confidence in their correct functioning.

This paper aims to promote the developing of design patterns for solutions to hard problems within membrane computing. As a starting point, three of them are introduced, namely: the *exponential space* pattern, to create an exponential number of membranes in linear time; the *all present* pattern, to check if all of a number of objects are present in a membrane; and the *no if not yes* pattern, to supply the `no` object when the `yes` object has not appeared in a membrane at a specific computation step.

To highlight the advantages attained from the utilization of design patterns, several solutions to the SAT problem are given. Although the variants of P systems considered work with different features, the proposed design patterns can be implemented in all of them, what we exploit to use the same structure for each of the solutions.

There are three clear lines for future work:

- Provide implementations of the design patterns in variants of P systems other than P systems with active membranes, for example, P systems with symport/antiport rules or tissue P systems.
- Abstract more design patterns from existent or new solutions. In particular, a review of the solutions to **PP**-complete and PSPACE-complete problems that can be found in the literature (for example [14–18]) shows that other types of patterns are required. Namely, for the former, the ability to "count" is needed, whereas for the latter a hierarchical separation of the space is usually performed.
- Apply the design patterns to obtain efficient solutions to new hard problems.

Acknowledgments. The authors are very grateful to Mario J. Pérez-Jiménez for his unconditional support, unlimited generosity, patience and enthusiasm, and particularly for his skillful advising and guiding as their "scientific father".

The authors also acknowledge the support from research project TIN2017-89842-P, cofinanced by Ministerio de Economía, Industria y Competitividad (MINECO) of Spain, through the Agencia Estatal de Investigación (AEI), and by Fondo Europeo de Desarrollo Regional (FEDER) of the European Union.

References

1. Fortnow, L., Homer, S.: A short history of computational complexity. Bull. Eur. Assoc. Theor. Comput. Sci. (EATCS) **80**, 95–133 (2003)

2. Păun, Gh.: Computing with membranes. J. Comput. Syst. Sci. **61**(1), 108–143 (2000)
3. Păun, Gh.: Membrane Computing. An Introduction. Natural Computing Series. Springer, Heidelberg (2002). https://doi.org/10.1007/978-3-642-56196-2
4. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2009)
5. Păun, Gh.: P systems with active membranes: attacking NP-complete problems. J. Autom. Lang. Comb. **6**(1), 75–90 (2001)
6. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: Complexity classes in models of cellular computing with membranes. Nat. Comput. **2**(3), 265–285 (2003)
7. Pérez-Jiménez, M.J., Riscos-Núñez, A., Romero-Jiménez, Á., Woods, D.: Complexity: membrane division, membrane creation. In: Păun et al. [4], chap. 12, pp. 302–336
8. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
9. Gutiérrez-Naranjo, M.Á., Pérez-Jiménez, M.J., Romero-Campero, F.J.: A uniform solution to SAT using membrane creation. Theor. Comput. Sci. **371**(1–2), 54–61 (2007). https://doi.org/10.1016/j.tcs.2006.10.013
10. Pan, L., Ishdorj, T.O.: P systems with active membranes and separation rules. J. Univers. Comput. Sci. **10**(5), 630–649 (2004). https://doi.org/10.3217/jucs-010-05-0630
11. Pérez-Jiménez, M.J., Romero-Jiménez, Á., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division. J. Autom. Lang. Comb. **11**(4), 423–434 (2006)
12. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del Amor, M.Á., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Reaching efficiency through collaboration in membrane systems: dissolution, polarization and cooperation. Theor. Comput. Sci. **701**, 226–234 (2017). https://doi.org/10.1016/j.tcs.2017.04.015
13. Valencia-Cabrera, L., Orellana-Martín, D., Martínez-del Amor, M.Á., Riscos-Núñez, A., Pérez-Jiménez, M.J.: From distribution to replication in cooperative systems with active membranes: a frontier of the efficiency. Theor. Comput. Sci. **736**, 15–24 (2018). https://doi.org/10.1016/j.tcs.2017.12.012
14. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Elementary active membranes have the power of counting. Int. J. Nat. Comput. Res. **2**(3), 35–48 (2011). https://doi.org/10.4018/jncr.2011070104
15. Leporati, A., Manzoni, L., Mauri, G., Porreca, A.E., Zandron, C.: The counting power of P systems with antimatter. Theor. Comput. Sci. **701**, 161–173 (2017). https://doi.org/10.1016/j.tcs.2017.03.045
16. Alhazov, A., Pérez-Jiménez, M.J.: Uniform solution of, QSAT using polarizationless active membranes. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 122–133. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74593-8_11
17. Sosík, P.: The computational power of cell division in P systems: beating down parallel computers? Nat. Comput. **2**(3), 287–298 (2003). https://doi.org/10.1023/A:1025401325428
18. Alhazov, A., Martín-Vide, C., Pan, L.: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. Fundam. Inform. **58**(2), 67–77 (2003)