Computationally Hard Problems Addressed Through P Systems

Mario J. Pérez-Jiménez, Alvaro Romero-Jiménez,
Fernando Sancho-Caparrini

Department Computer Science and Artificial Intelligence
E.T.S. Ingeniería Informática, University of Seville
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
{Mario.Perez,Alvaro.Romero,Fernando.Sancho}@cs.us.es

Summary. In this chapter we present a general framework to provide efficient solutions to decision problems through families of cell-like membrane systems constructed in a semi-uniform way (associating with *each instance* of the problem one P system solving it) or a uniform way (all instances of a decision problem having the same *size* are processed by the same system). We also show a brief compendium of efficient semi-uniform and uniform solutions to hard problems in these systems, and we explicitly describe some of these solutions.

1 Introduction

Many interesting problems of the real world are presumably intractable (unless **P** = **NP**) and hence it is not possible to obtain algorithmic solutions when we address large instances of those problems on an electronic computer.

From this point of view, membrane systems have two attractive features: they are inherently parallel and nondeterministic. Can the parallelism and nondeterminism of P systems be used to solve hard problems in feasible time? The answer is yes, but we must point out two facts. On the one hand, we have to deal with the nondeterminism in such a way that the solutions obtained by using these devices are algorithmic solutions in the classical sense, that is, the answers of the computations of the system must be reliable. On the other hand, the drastic decrease of the execution time from exponential to polynomial is not achieved for free, but with the use of an exponential amount of space, although this space is created in polynomial time.

In this chapter we present the theoretical requirements for a P system to provide an *algorithmic solution* to an abstract decision problem (a precise definition of the latter is given in Section 2). First, all computations of the system must halt, providing a positive or negative answer to the problem (i.e., for a particular instance of it). Second, we impose that the systems be confluent. This is a generalization of the notion of determinism for which we require all possible computations to provide the same answer. This way we do not obtain a contradictory result. In Section 3, P systems verifying these two properties are called recognizer systems.

It is important to note that all the feasible solutions to hard problems obtained by means of these biologically inspired devices so far presented do not use a single P system, but a family of systems. However, there are significative differences between those solutions, dividing them in two groups: the *semi-uniform* solutions, which associate with *each instance* of the problem one P system solving it, and the *uniform* solutions, which associate with *each possible size* of the instances of the problem one P system that can solve all instances of that size. A formal definition of these two concepts can be found in Sections 4 and 5.

Another possible classification can be considered with respect to the existence or not in the system of a membrane where the input data is introduced before the computation starts. Usually, the semi-uniform solutions are performed by P systems without input, whereas the uniform solutions are performed by P systems with input. In Section 4 we present a compendium of known semi-uniform solutions to hard problems by P systems without input, and a detailed description of two of these solutions: one to the *Satisfiability Problem* and the other to the *Hamiltonian Path Problem.* Finally, in Section 5 we do the same for known uniform solutions to hard problems by P systems with input, detailing the ones corresponding to the *Decision Knapsack Problem (0/1)* and to the *Common Algorithmic Decision Problem.*

2 Abstract Problems

Membrane systems provide devices with the ability to solve problems. But these machines are equipped only with tools able to handle inputs and outputs that are multisets of symbol objects. Hence, we are forced to treat these problems as collections of multisets.

In general we define an *abstract problem* X to be a pair $X = (I_X, S_X)$ where I_X is a language over a finite alphabet, whose elements are called *instances* of the problem, S_X is a function whose domain is I_X, and for any instance $i \in I_X$, the set $S_X(i)$ is finite (the elements of this set are called *candidate solutions*). Observe that the set of instances of an abstract problem is a finite or enumerable set.

As an example, consider the problem MAX-CLIQUE of finding a largest clique in a given unidirected graph. Recall that a *clique* in a graph is a set of vertices such that their each pair is connected by an edge. An instance of MAX-CLIQUE is an undirected graph, and a candidate solution is a set of vertices. An *exact solution* will be a set of vertices which is a clique with the maximal number of vertices. The problem MAX-CLIQUE itself is the relation that associates each undirected graph with each largest cliques in the graph (an exact solution). Since largest cliques in an uniderected graph are not necessarily unique, a given problem instance may have more than one exact solution; that is, the binary relation associated with the problem is not necessarily univocal.

In this chapter we will work only with *decision problems*, that is, abstracts problems that require either a *yes* or a *no* answer. Formally, a *decision problem* X is an abstract problem (I_X, S_X) such that S_X is a total boolean function (that is, a predicate) over I_X.

For example, the following is a decision version of the problem CLIQUE: given an undirected graph G and a natural number k, determine whether or not G has a clique of size at least k. An instance for CLIQUE is a pair (G, k) where G is an undirected graph and k is a natural number, and a candidate solution is 1 (*yes*) or 0 (*no*). If $i = (G, k)$ is an instance of this problem, then $S_{CLIQUE}(i) = 1$ (*yes*) if there exists a clique in G of size at least k, and $S_{CLIQUE}(i) = 0$ (*no*) otherwise.

There exists a natural correspondence between languages and decision problems in the following way: each language L over an alphabet Σ has a decision problem, $X_L = (I_{X_L}, S_{X_L})$, associated with it, where $I_{X_L} = \Sigma$ and $S_{X_L} = \{(x, 1) \mid x \in L\} \cup \{(x, 0) \mid x \notin L\}$; reciprocally, given a decision problem $X = (I_X, S_X)$, the language L_X over I_X corresponding to it is defined as $L_X = \{a \in I_X \mid S_X(a) = 1\}$.

Let M be a Turing machine such that the result of any halting computation is *yes* or *no*. Let L be a language over an alphabet Σ. If M is a deterministic device then we say that M *recognizes* or *decides* L whenever, for any string a over Σ, if $a \in L$ then the answer of M on input a is *yes*, and *no* otherwise. If M is a nondeterministic Turing machine, then we say that M *recognizes* or *decides* L if the following is true: for any string a over Σ, $a \in L$ if and only if there exists a computation of M with input a such that the answer is *yes*. That is, an input string a is accepted if there is *some* accepting computation of M on input a.

Notice the difference in the definition of acceptance by deterministic and nondeterministic Turing machines. An input string a is accepted by a deterministic Turing machine M if *the* computation of M on input a halts and answers *yes*. A nondeterministic Turing machine M accepts a string a if there exists *some* computation of M on input a answering *yes*; that is, there exists a sequence of nondeterministic choices that results in *yes*. In this case, it is possible that we accept a string but that there exists another computation with the same input that either halts and answers *no*, or does not halt.

In some sense, we can state that nondeterministic devices do not properly capture the intuitive idea underlying the concept of algorithm, because the result of such a machine on an input is not reliable, since the answer of the device is not always the same.

In the context of computability theory, we consider a problem X to be solved when we have a *general method* (described in a model of computation) that works for any instance of the problem. From a practical point of view, such methods run only over a finite set of instances whose cardinality depends on the available resources.

We say that a Turing machine M solves a decision problem X if M *recognizes* the language associated with X; that is, for any instance a of the problem: (a) in the deterministic case, the machine (with input a) outputs *yes* if the answer of the problem is *yes*, and outputs *no* otherwise, and (b) in the nondeterministic case, some computation of the machine (with input a) outputs *yes* if the answer of the problem is *yes*.

As for when the instances of abstract problems are strings, we can consider their size in a natural manner: the size of an instance is the length of the string. Then, how do the resources required to execute a method increase according to the size of the instance? This is a fundamental question in computational complexity theory.

Let M be a deterministic Turing machine. Let R be a resource used by the device (for example, the number of steps executed before the machine halts, the *time* of execution). We consider a function f_R mapping nonnegative integers to nonnegative integers defined as follows: $f_R(n)$ is the maximum, over all instances a of size n, of the amount of resource R used when the device M is executed with input a. For example, if R is the time of execution, then we say that M operates in time $f_R(n)$; if f_R turns to be a polynomial function, then we say that M works in polynomial time.

3 Recognizer P Systems

In this chapter we use membrane computing as a framework to address the resolution of decision problems; that is why we consider P systems as *recognizer* devices. Since P systems work in a nondeterministic manner, we need to adapt the usual definition of the processes of acceptance in nondeterministic Turing machines.

In order to accept or reject an input string/multiset it should be enough to read the answer of *any* computation of the system. Hence it is necessary to require a condition of *confluence* in the following sense: every computation of the system is a halting computation, and (on the same input, if any) all computations have the same output.

Definition 1. *A* recognizer P system *is a P system with external output such that:*

1. *The working alphabet contains two distinguished elements* yes *and* no.
2. *All computations halt.*
3. *If $\mathcal{C}$ is a computation of the system, then either some object* yes *or some object* no *(but not both) must have been released into the environment, and only in the last step of the computation.*

For recognizer P systems, we say that a computation $\mathcal{C}$ is an *accepting computation* (or *rejecting computation*) if the object *yes* (or *no*) appears in the external environment associated with the corresponding halting configuration of $\mathcal{C}$. Hence, these devices send to the environment an accepting or rejecting answer at the end of their computations. Therefore, if we want these kinds of systems to properly solve decision problems, we have to require all responses to be consistent, in the sense that the system must always give the same answer.

4 Recognizer P Systems Without Input

The first results about *solvability* of **NP**-complete problems in polynomial (even linear) time by membrane systems were given by Gh. Păun [24], C. Zandron et al. [40], S.N. Krishna et al. [10], and A. Obtulowicz [15] in the framework of P systems that lack an input membrane. Thus, the constructive proofs of such results need to design *one* system for *each* instance of the problem.

This method for solving problems provides a *specific* algorithmic solution, in the following sense: if we wanted to apply such a method to some decision problem then a system should be constructed for every instance of the problem. However, the construction is done in polynomial time, which prevents the solving of the problem during the "programming" phase.

Now, we formalize these ideas in the following definition.

Definition 2. *Let $X = (I_X, \theta_X)$ be a decision problem. We say that X is solvable in polynomial time by a family of recognizer membrane systems* without input $\mathbf{\Pi} = (\Pi(w))_{w \in I_X}$ *if the following are true:*

- *The family $\mathbf{\Pi}$ is "Turing polynomially uniform"; that is, there exists a deterministic Turing machine which, in polynomial time, constructs the system $\Pi(w)$ starting from the instance $w \in I_X$.*
- *The family $\mathbf{\Pi}$ is polynomially bounded; that is, there exists a polynomial function $p(n)$ such that for each $w \in I_X$, every computation of $\Pi(w)$ halts in at most $p(|w|)$ steps.*
- *The family $\mathbf{\Pi}$ is sound; that is, for each instance of the problem $w \in I_X$, if there exists an accepting computation of $\Pi(w)$, then $\theta_X(w) = 1$ (the corresponding answer of the problem is* yes*).*
- *The family $\mathbf{\Pi}$ is complete; that is, for each instance of the problem $w \in I_X$, if $\theta_X(w) = 1$ (the answer to the problem is* yes*), then every computation of $\Pi(w)$ is an accepting computation (the system also responds* yes*).*

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

Notice that in the above definition we consider two different tasks. The first one is the construction of the family solving the problem, which we require to be done in polynomial time; that is, there exists a deterministic Turing machine M, and a polynomial function $q(n)$ such that for each $w \in I_X$, $M(w)$ provides a complete description of the system $\Pi(w)$ in at most $q(|w|)$ steps. This is precomputation time, expressed in the number of *sequential steps*.

The second task is the execution of the systems $\Pi(w)$ of the family, and for this task we impose that the total number of steps performed by the computations of system $\Pi(w)$ is bounded by the polynomial function $p(n)$. This is the real computation time, and it is described by the number of *parallel steps*.

In this section we use recognizer membrane systems *without input membrane* in order to solve decision problems. In this context, we need to associate with each instance of the problem such a system, accepting or rejecting it. Bearing in mind the nondeterminism of the system, we require the confluence condition; that is, all branches of a computation associated with the instance eventually reach a unique configuration.

Unless **P**=**NP**, to solve an **NP**-complete problem in an efficient way using P systems we have to create an exponential workspace in polynomial time. This is possible in various types of membrane systems, for example, through *membrane division* [24] (we can repeatedly divide membranes in order to obtain 2^n membranes in n steps), *membrane creation* [9], [13] (new membranes are produced under the influence of the existing objects in a membrane), *string replication* [4] (in a rewriting membrane system using string objects we can generate exponentially many strings in linear time), or by using *precomputed resources* [25], [6] (we start from an arbitrarily large initial membrane structure, without objects placed in its regions, and we trigger a computation by introducing both objects and rules related to a given problem in a specified membrane).

In [23] Gh. Păun explains the convenience of solving many **NP**-complete problems in a *uniform manner*, that is, with P systems which are very similar to each other. This idea about the uniformity of the P systems able to solve a decision problem was formulated for the first time by A. Obtulowicz [17] in relation to a proposed solution to SAT: the family of P systems (without input membrane) described in [24] is generated in a logarithmic space by a multitape deterministic Turing machine. We say that this construction is *semi-uniform*: the systems of the family solving the decision problem are constructed starting not from the *size* of an instance, but from an instance only; that is, in a recursive manner, for each instance of the problem a P system associated with it is constructed.

In contrast, in Section 5 we will define the concept of *uniform* construction of families of P systems solving a decision problem.

Now, we briefly comment the different efficient solutions of **NP**-complete problems in the framework of P systems without input, described in a semi-uniform way, proposed until now.

The first efficient solution to SAT is given by Gh. Păun in [24], using division for non-elementary membranes. This result was improved by Gh. Păun, Y. Suzuki, H. Tanaka, and T. Yokomori in [29] using only division for elementary membranes (in that paper a solution to HPP using membrane creation is also presented).

In [2], A. Alhazov and T.O. Ishdorj present a linear time solution to SAT through P systems with active membranes without polarizations but using some membrane rules (merging, separation, and release).

The first efficient solution to HPP by P systems is due to S.N. Krishna and R. Rama [10], but using rules for d-division, with an arbitrary d (a solution to Vertex Cover is also provided in this paper). The result of S.N. Krishna and R. Rama about HPP was improved by A. Păun in [21] by using only rules for 2-division.

Different efficient solutions to several variants of SAT (3-SAT, *Not-all-equal* 3 SAT, *One-in-three* 3SAT, and *Minimum 2-satisfiability*) are decribed by Gh. Păun, Y. Suzuki, H. Tanaka, and T. Yokomori in [29]. Also, in that paper solutions to several problems of graph theory (*Vertex Cover*, *k-closure*, *Independent set*, *Graph 3-colourability*, and *Monochromatic triangle*) are given by P systems with active membranes using cooperative rules.

Other efficient solutions to SAT and HPP are given by J. Castellanos et al. [4], through P systems using string replications (similar solutions are given by S.N. Krishna and R. Rama in [11]), and by E. Czeizler [6], through P systems using precomputed resources.

A polynomial solution to *Integer Factorization Problem* is presented by A. Obtulowicz in [16] through deterministic P systems with active membranes but using cooperative rules.

P. Sosik in [39] provides a semi-uniform efficient solution to QBF (satisfiability of quantified propositional formulas), a well known **PSPACE**-complete problem in the framework of P systems with active membranes but using 2-division for nonelementary membranes.

S.N. Krishna and R. Rama [12] show how P systems with membrane division can theoretically break the most widely used cryptosystem, DES. That is, given an arbitrary (plain text, cipher text) pair, one can recover the DES key in linear time with respect to the length of the key.

In this section we present a semi-uniform solution to *SAT* problem due to C. Zandron, G. Mauri, and C. Ferretti [41] (with slight modifications the above solution can be adapted to a linear time solution by recognizer P systems, as we point out at the end of subsection 4.1). We also describe a quadratic time semi-uniform solution to the Hamiltonian Path Problem, a variant of a solution presented by M.J. Pérez-Jiménez, A. Romero-Jiménez, and F. Sancho-Caparrini in [35].

4.1 A Linear Time Solution to SAT

The Satisfiability Problem (SAT) is the following decision problem:

> *Given a propositional formula in conjunctive normal form, determine if there exists a truth assignment of its variables which makes the formula true.*

In what follows we present a family of P systems with active membranes using 2-division that solves SAT in linear time.

Given an instance ϕ of SAT we construct a P system $\Pi(\phi)$ whose functioning can be divided into the following stages:

(a) Generate all the possible truth assignments for the variables of ϕ. (This will be done in a nondeterministic way.)
(b) Apply the assignments generated in the previous stage to the formula.
(c) Check if all the clauses have value *true*.
(d) Answer *yes* or *no* depending on the results from (c).

Let us suppose that ϕ is a formula with m clauses and n variables. That is, $\phi = C_1 \wedge \cdots \wedge C_m$ for some $m \geq 1$, with $C_i = y_{i,1} \vee \cdots \vee y_{i,p_i}$ for some $p_i \geq 1$ and $y_{i,j} \in \{x_k, \neg x_k \mid 1 \leq k \leq n\}$, for each $1 \leq i \leq m, 1 \leq j \leq p_i$.

The P system $\Pi(\phi)$ is defined as follows.

- The working alphabet is

$$\begin{aligned} \Gamma(\phi) = {} & \{a_i, t_i, f_i \mid 1 \leq i \leq n\} \cup \{r_i \mid 0 \leq i \leq m\} \\ & \cup \{c_i \mid 1 \leq i \leq m-1\} \cup \{d_i \mid 0 \leq i \leq n\} \cup \{yes\}. \end{aligned}$$

- The set of labels is $\{1, 2\}$.
- The initial membrane structure is $[_1 [_2\ ]_2^0]_1^0$ (each membrane with label 2 is said to be an internal membrane).
- The initial multisets associated with the membranes are $\mathcal{M}_1 = \lambda$ and $\mathcal{M}_2 = a_1 a_2 \ldots a_n d_0$.
- The rules are:

(a.1) $[_2 a_i]_2^0 \rightarrow [_2 t_i]_2^0 [_2 f_i]_2^0$, for $1 \leq i \leq n$,

(a.2) $[_2 d_k \rightarrow d_{k+1}]_2^0$, for $0 \leq k \leq n-2$,

(a.3) $[_2 d_{n-1} \rightarrow d_n c]_2^0$,

(a.4) $[_2 d_n]_2^0 \rightarrow [_2]_2^+ d_n$,

(b.1) $[_2 t_i \rightarrow r_{h_{i,1}} \ldots r_{h_{i,j_i}}]_2^+$, for $1 \leq i \leq n$, and the clauses $C_{h_{i,1}}, \ldots, C_{h_{i,j_i}}$ contain the literal x_i,

(b.2) $[_2 f_i \rightarrow r_{h_{i,1}} \ldots r_{h_{i,j_i}}]_2^+$, for $1 \leq i \leq n$, and the clauses $C_{h_{i,1}}, \ldots, C_{h_{i,j_i}}$ contain the literal $\neg x_i$,

(c.1) $[_2 r_1]_2^+ \rightarrow [_2]_2^- r_1$, (c.2) $[_2 c_i \to c_{i+1}]_2^-$, for $1 \leq i \leq m$,

(c.3) $[_2 r_k \to r_{k-1}]_2^-$, for $2 \leq k \leq m$,

(c.4) $r_1 [_2]_2^- \to [_2 r_0]_2^+$,

(c.5) $[_2 c_{m+1}]_2^+ \to [_2]_2^+ yes$,

(d.1) $[_1 yes]_1^0 \to [_1]_1^0 yes$.

Let us see now that this P system indeed solves `SAT` for the formula ϕ.

The objects a_i contained in the initial internal membrane of the system correspond to the variables x_i. By using a rule from (a.1), for i nondeterministically chosen, we produce the truth values *true* and *false* assigned to variable x_i. The truth values are represented by the objects t_i and f_i, respectively, and are placed in two separate copies of the internal membrane. Since the charge remains neutral for both membranes, the process can continue.

In this way, in n steps we assign truth values to all variables. Hence, we get all the 2^n different truth assignments for the formula ϕ, placed in 2^n separate internal membranes, which, in turn, are placed in the skin membrane. Note that, in spite of the fact that in each step the object a_i is nondeterministically chosen, after n steps we get the same result, regardless of the objects used in each step.

On the other hand, the objects d_i are used as counters, to control when the process described above has finished. The initial internal membrane starts with the object d_0 and at each division step we pass from d_k to d_{k+1} using the rules from (a.2). Also, each new internal membrane created by division gets copies of these objects, keeping their own counter this way. At the step before the last division, the rule from (a.3) introduces both d_n and c_1. The former object will exit the membrane (at step $n+1$ using the rule from (a.4)), changing its polarization to positive, and this finishes the first stage and starts the second one in that membrane. The latter object will be used in the third stage.

In the second stage we look for the clauses satisfied by the truth assignments associated with each internal membrane. We do this in one step in parallel for all the 2^n membranes and in parallel for all the truth values present in them, using the rules from (b.1) and (b.2). These rules introduce an object r_i in the membrane for each clause satisfied by the truth value being considered.

If, after completing this step, there is at least one internal membrane which contains all symbols $r_1, \ldots, r_m$, this means that the truth assignment associated with that membrane satisfies all the clauses, and therefore satisfies the formula ϕ. Otherwise, if no internal membrane gets all the objects $r_1, \ldots, r_m$, the formula ϕ is not satisfiable. Note that the satisfiability problem has already been solved, in $n+2$ steps, making an essential use of the parallelism, but we still have to "read" the answer and send a suitable signal out of the system.

We use the rules from (c.1–4) in the third stage to check if some internal membrane has an associated truth assignment that makes true all the clauses of ϕ. To do this we carry out a loop in which we check in each step if the object r_1 is present, eliminate it, and perform a rotation of the objects $r_2, \dots, r_m$, decreasing their subscripts by 1. It is clear that an internal membrane contains all the objects $r_1, \dots, r_m$ (that is, the truth assignment associated with it satisfies all the clauses and, hence, the formula) if and only if we can run m steps of the loop.

Let us take a closer look at how one step of the loop is performed. First the rule from (c.1) checks whether or not the object r_1 is present in the membrane. If this is the case, r_1 is sent out of the membrane, and the polarization of the membrane is changed to negative. The membranes which do not contain the object r_1 remain positively charged and will no longer evolve; no further rule can be applied to them. Next, for all the internal membranes negatively charged (that is, those that had contained an object r_1) the rules from (c.3) decrease the subscripts of the objects $r_2, \dots, r_m$ in that membrane, and the rule from (c.4) gives back the object r_1 from the skin membrane, changed to a dummy object r_0 which will never evolve again, and changing the polarization of the membrane to positive so that the process can be started again. Note the important fact that in the skin membrane we have a number of copies of the object r_1 equal to the number of membranes with a negative charge. Because a rule from (c.4) can only involve one membrane and because these rules are applied in the maximally parallel way, each membrane which contained before an object r_1 will now contain an object r_0 and will be able to continue running the loop.

Simultaneously, in an internal membrane negatively charged, the rule (c.2) increases the subscripts of the objects c_i. These objects are used to count the number of steps of the previous loop that have been carried out. Thus, if an object c_{m+1} appears in the membrane, it means the membrane contained all the objects $r_1, \dots, r_m$ since all the m steps of the loop have been able to run. Also, the membrane is positively charged because the rule from (c.4) has just been applied. Therefore, we can apply the rule from (c.5), which sends out the object c_{m+1} to the skin as an object *yes*. Finally, this object is sent out to the environment by means of the rule from (d.1).

The family of P systems $\Pi(\phi)$ constructed as above for each propositional formula ϕ in conjunctive normal form is *Turing polynomially uniform*. Indeed, the evolution rules are computable in a uniform way; the size of the working alphabet is $4n + 2m + 2$; the number of membranes in the initial membrane structure is 2; the maximum cardinality of the initial multisets is $n + 1$; the total number of rules is $4n + 2m + 4$ and the maximum length of a rule is $\max(7, m + 3)$.

However, this family should be adjusted in several respects in order to fulfill the conditions requested by Definition 2 from Section 4 for a linear time solution to `SAT`.

1. First, the P systems constructed above, although confluent, are nondeterministic. This is not really a problem, but if we want deterministic P

systems, we have to change the rules for stage 1 so that they cover all the objects a_i sequentially, instead of choosing them nondeterministically.

2. Then, the P systems may not halt in linear time, because the rule from (d.1) can only be applied once at a given time. If, for example, the formula ϕ is valid, then we may have 2^n objects *yes* in the skin membrane at the last step of the computation. These objects are then sent out to the environment one by one, so the total time used will be exponential. The solution is easy: we change the polarization of the skin when the first object *yes* is sent out, so that the rule from (c.2) cannot be applied any more and the system halts.
3. Third, the P systems above are not recognizer systems. If the formula is satisfiable, we obtain the answer *yes* in the environment, but if the formula is not satisfiable the system halts without answering anything. To solve this problem we can add objects to the skin membrane that count the number of computation steps that have been carried out. With the aid of these counters we can check if the system should have already sent an object *yes* in case the formula is satisfiable. Otherwise, we can be sure that the formula is not satisfiable and send an object *no* to the environment.

4.2 A Quadratic Time Solution to `HPP`

A Hamiltonian path in a graph G is a path that visits each of the nodes of G exactly once. The Hamiltonian Path Problem (`HPP`) is the following decision problem:

> *Given a graph G, determine if there exists a Hamiltonian path in G.*

In the following discussion we will construct, in a way similar to that of Section 7.2.2 of [25], a family of recognizer P systems with active membranes using 2-division that solves `HPP` in quadratic time. Specifically, given an instance G of `HPP` of size n (that is, with n nodes), we construct a P system $\Pi(G)$ whose functioning can be divided into the following stages:

(a) Generate all the possible paths in G of length n.
(b) For each of the previous paths, determine whether it visits all the nodes of G.
(c) Answer *yes* or *no* depending on the results from (b).

Consequently, this P system solves `HPP` for G, because a path in G is Hamiltonian if and only if it has length n and visits all the nodes of G.

Before describing $\Pi(G)$ in full detail, we have to fix several notations. Let G be a graph of size n. We denote by $V = \{v_1, \dots, v_n\}$ the set of nodes and by E the set of edges of G. Given a node v_i, we assume that the set of nodes $adj(v_i) = \{v_{j^i_1}, \dots, v_{j^i_{k(i)}}\}$ adjacent to v is ordered such that $j^i_1 < \dots < j^i_{k(i)}$ (and, since we suppose G to be connected, it is nonempty).

For example, let us consider the graph in Figure 1. For this graph, $V = \{v_1, v_2, v_3\}$ and $E = \{\{v_1, v_2\}, \{v_1, v_3\}\}$. Also, $adj(v_1) = \{v_2, v_3\}$, so $k(1) = 2$ and $j_1^1 = 2, j_2^1 = 3$; $adj(v_2) = \{v_1\}$, so $k(2) = 1$ and $j_1^2 = 1$; $adj(v_3) = \{v_1\}$, so $k(3) = 1$ and $j_1^3 = 1$.

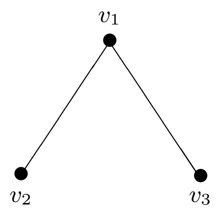

Fig. 1. Example graph.

The P system $\Pi(G)$ is defined as follows.

- The working alphabet is

$$\begin{aligned}\Gamma(G) = {} & \{v_i \mid 1 \le i \le n\} \cup \{v_{i,j} \mid 1 \le i, j \le n\} \cup \{a_{i \to j,l} \mid 1 \le i, j, l \le n\} \\ & \cup \{r'_i \mid 1 \le i \le n\} \cup \{r_i \mid 0 \le i \le n\} \cup \{c_i \mid 1 \le i \le n+1\} \\ & \cup \{d_i \mid 0 \le i \le n^2 + 2n + 4\} \cup \{c, a_{n+1}, yes, no\}.\end{aligned}$$

- The set of labels is $\{1, 2\}$.
- The initial membrane structure is $[_1\ [_2\ ]_2^0\ ]_1^0$ (each membrane with label 2 is said to be internal).
- The initial multisets associated with the membranes are $\mathcal{M}_1 = d_0$ and $\mathcal{M}_2 = v_1$.
- The rules are:

(a.1) $[_2 v_i]_2^0 \to [_2 v_{i,1}]_2^- [_2 v_{i+1}]_2^0$, for $1 \le i \le n-2$,

(a.2) $[_2 v_{n-1}]_2^0 \to [_2 v_{n-1,1}]_2^- [_2 v_{n,1}]_2^-$,

(a.3) $[_2 v_{i,l} \to r'_i a_{i \to j_1^i, l}]_2^-$, for $1 \le i \le n, 1 \le l \le n-1$,

(a.4) $[_2 v_{i,n} \to r'_i c a_{n+1}]_2^-$, for $1 \le i \le n$,

(a.5) $[_2 a_{i \to j_k^i, l}]_2^- \to [_2 v_{j_k^i, l+1}]_2^- [_2 a_{i \to j_{k+1}^i, l}]_2^-$, for $1 \le i \le n, 1 \le k \le k(i) - 1, 1 \le l \le n-1$,

(a.6) $[_2 a_{i \to j_{k(i)}^i, l} \to v_{j_{k(i)}^i, l+1}]_2^-$, for $1 \le i \le n, 1 \le l \le n-1$,

(a.7) $[_2 a_{n+1}]_2^- \to [_2]_2^+ a_{n+1}$,

(b.1) $[_2 r'_i \to r_i]_2^+$, for $1 \le i \le n$,

(b.2) $[_2 c \to c_1]_2^+$,

(b.3) $[_2 r_1]_2^+ \to [_2]_2^- r_1$,

(b.4) $[_2 r_i \to r_{i-1}]_2^-$, for $1 \le i \le n$,

(b.5) $[_2 c_i \to c_{i+1}]_2^-$, for $1 \le i \le n$,
(b.6) $r_1 [_2]_2^- \to [_2 r_0]_2^+$,
(b.7) $[_2 c_{n+1}]_2^+ \to [_2]_2^+ c_{n+1}$,

(c.1) $[_1 c_{n+1}]_1^0 \to [_1]_1^+ yes$,
(c.2) $[_1 d_i \to d_{i+1}]_1^0$, for $0 \le i \le n^2 + 2n + 3$,
(c.3) $[_1 d_{n^2+2n+4}]_1^0 \to [_1]_1^- no$.

Let us see now that this P system indeed solves HPP for G.

The rules from (a.1) to (a.7) perform the first stage; that is, they generate all the possible paths in G of length n. In this stage, the objects v_i denote the starting node of the path to be generated, whereas the objects $v_{i,l}$ means that we are considering the node v_i as the lth element of the path.

We first create by succesive divisions, using the rules from (a.1) and (a.2), n internal membranes in which the paths starting from each of the nodes of G will be generated. Each of these membranes contains a different object $v_{i,1}$; that is, the first node in the path being considered in the corresponding membrane is the node v_i. Also, these membranes are negatively charged, and they remain this way through the first stage.

Suppose now that an internal membrane contains an object $v_{i,l}$, meaning that the current node being considered as the lth element of the path is the node v_i. The rule (a.3) keeps a record of this fact by means of the object r'_i and starts the process of choosing the next node in the path. This process is performed by the rules from (a.5) and (a.6) which generate a new membrane for each of the nodes adjacent to v_i. This is done by successive divisions of the internal membrane using the objects of type $a_{i \to j,l}$ to cover all the nodes in $adj(v_i)$. These objects transform themselves into an object $v_{j,l+1}$ in one of the new membranes created, to indicate that the $(l+1)$th node to include in the path is v_j, and into another object $a_{i \to j',l}$ in the other new membrane, to consider the next node adjacent to v_i.

Observe that the generation of the paths is done in parallel but not simultaneously, because we continue generating the path as soon as we choose the node adjacent to the current one. In other words, we do not wait until all the membranes considering a node adjacent to the current one have been generated to continue with the next element of the path.

Finally, when the last node of the path is reached, the rules from (a.4) are applied, keeping a record of this last node and introducing an object c to be used in the next stage and an object a_{n+1} to start the stage.

For the rules corresponding to the second stage not to be fired at the same time as the rules corresponding to the first stage, the sets of objects used in each of them are disjoint. At the beginning of the second stage the objects r'_i and c produced in the first stage are transformed into objects r_i and c_1, respectively. Now we can check for the existence of a membrane containing all the objects $r_1, \dots, r_n$ the same way as is done in the third stage of the solution to SAT presented in subsection 4.1.

Finally, suppose that there is a Hamiltonian path in G. Then, at least one of the internal membranes created along the first stage contains all the objects $r_1, \dots, r_n$. An object c_{n+1} is then sent out of this membrane at the end of the second stage. Using the rule (c.1) this object is sent to the environment as an object *yes* and the charge of the skin is changed to positive. Hence, the rule (c.3) cannot be applied and, since the P system eventually halts, the answer to the `HPP` problem for G is positive.

On the other hand, if there is no Hamiltonian path in G, no internal membrane contains all the objects $r_1, \dots, r_n$. Therefore, all these membranes stall at some moment of the second stage without sending out an object c_{n+1}, which, in turn, cannot be sent out to the environment to produce a positive answer.

We focus our attention on the rule (c.2); this rule is applied at each step of the computation and uses objects d_i to count how many steps have been performed. It is easy to see that stage 1 lasts at most $n^2 + 1$ steps and stage 2 lasts at most $2n + 2$ steps. Hence, if at step $n^2 + 2n + 4$ the charge of the skin has not changed to positive by means of the rule (c.1), then the answer must be negative; this is shown using the rule (c.3) to send an object *no* to the environment. The P system then halts.

Now, we are going to justify that the family $\mathbf{\Pi} = (\mathit{\Pi}(G))_{G \in \mathbf{HPP}}$ solves the problem `HPP` in quadratic time.

First, the above description of the evolution rules is computable in a uniform way. It is also a polynomial description, since the size of the working alphabet is $n^3 + 2n^2 + 6n + 10$; the number of membranes in the initial membrane structure is 2; the maximum cardinal of the initial multisets is 1; the total number of evolution rules is at most $n^3 + 7n + 9$; and the maximum length of a rule is 7. Hence, the family $\mathbf{\Pi}$ is *Turing polynomially uniform.*

Second, the family $\mathbf{\Pi}$ is *polynomially bounded*, since $\mathit{\Pi}(G)$ is deterministic for every G and the total number of steps performed by the computation of $\mathit{\Pi}(G)$ is at most $n^2 + 2n + 5$, which is quadratic in the size of G.

Third, from the description of the functioning of the P system $\mathit{\Pi}(G)$ it can be seen that the family $\mathbf{\Pi}$ is sound and complete.

5 Recognizer P Systems with Input

Recall that a P system with input is a tuple $(\mathit{\Pi}, \mathit{\Sigma}, i_{\mathit{\Pi}})$, where (a) $\mathit{\Pi}$ is a P system with working alphabet $\mathit{\Gamma}$ and p membranes labeled $1, \dots, p$, with initial multisets $\mathcal{M}_1, \dots, \mathcal{M}_p$ associated with them; (b) $\mathit{\Sigma}$ is an (input) alphabet strictly contained in $\mathit{\Gamma}$, and the initial multisets are over $\mathit{\Gamma} - \mathit{\Sigma}$; and (c) $i_{\mathit{\Pi}}$ is the label of a distinguished (input) membrane. If m is a multiset over $\mathit{\Sigma}$, then the *initial configuration of* $(\mathit{\Pi}, \mathit{\Sigma}, i_{\mathit{\Pi}})$ *with input* m is $(\mu, \mathcal{M}_1, \dots, \mathcal{M}_{i_{\mathit{\Pi}}} \cup m, \dots, \mathcal{M}_p)$.

In this section we deal with recognizer P systems *with input membrane* and we propose to solve hard problems in a *uniform* way in the following sense: all instances of a decision problem that have the same *size* (according to a given polynomial time computable criterion) are processed by the same system, to which an apropriate input, that depends on the concrete instance, is supplied.

This method for solving problems provides a *general purpose* algorithmic solution in the following sense: if we want to implement such a solution, a system constructed to solve an instance of the problem can also to be used when trying to solve another instance of the same *size*.

Now, we formalize these ideas in the following definition.

Definition 3. *Let $X = (I_X, \theta_X)$ be a decision problem. We say that X is solvable in polynomial time by a family of recognizer membrane systems* with input $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ *if the following is true:*

- *The family $\mathbf{\Pi}$ is Turing polynomially uniform; that is, there exists a deterministic Turing machine that constructs in polynomial time the system $\Pi(n)$ starting from $n \in \mathbf{N}$.*
- *There exist two polynomial time computable functions, cod and s over the set I_X of instances of X, such that:*
  - *For every $w \in I_X$, $s(w)$ is a natural number and cod(w) is an input multiset of the system $\Pi(s(w))$.*
  - *The family $\mathbf{\Pi}$ is polynomially bounded; that is, there exists a polynomial function $p(n)$ such that for each $w \in I_X$ every computation of the system $\Pi(s(w))$ with input cod(w) is halting and, moreover, performs at most $p(|w|)$ steps.*
  - *The family $\mathbf{\Pi}$ is sound; that is, for each $w \in I_X$ if there exists an accepting computation of the system $\Pi(s(w))$ with input cod(w), then $\theta_X(w) = 1$.*
  - *The family $\mathbf{\Pi}$ is complete; that is, for each $w \in I_X$, if $\theta_X(w) = 1$, then every computation of the system $\Pi(s(w))$ with input cod(w) is an accepting computation.*

The soundness property means that if given an instance we obtain an *acceptance response* of the system associated with it (and individualized by the apropriate input multiset) through some computation, then the answer to the problem (for that instance) has to be *yes*. The completeness property means that if we obtain an *affirmative* response to the problem, then any computation of the system associated with it (and individualized by the apropriate input multiset) must be an accepting one.

Note that in the above definition we consider three different tasks. The first is the construction of the family solving the problem, which we require to be done in polynomial time. The second is the task performed by the polynomial time computable functions *cod* and s. The third is the execution of the systems $\Pi(n)$ of the family (with the appropriate input multiset), for which we impose the total number of steps performed by their computations to be bounded by a polynomial function.

Hence, to solve an instance w, we first of all need to compute the natural number $s(w)$, obtain the input multiset *cod*(w), and construct the system

$\Pi(s(w))$. This is a *precomputation stage*, running in polynomial time represented by the number of *sequential steps*. Next, we execute the system $\Pi(s(w))$ with input $cod(w)$. This is the proper *computation stage*, also running in polynomial time, but now the time is represented by the number of *parallel steps*.

Consequently, in order to solve a decision problem in this context, we need two polynomial time computable functions over the set of instances of the problem, in such a way that the first function assigns the P system that will process the instance when we give a suitable input provided by the second function. Then the system will accept or reject the instance. Bearing in mind the nondeterminism of the system, we require the confluence condition; that is, all computations of the system associated with the instance must always have the same answer.

Now, we briefly comment on the different efficient solutions of **NP**-complete problems obtained in the framework of P systems with input, described in a uniform way, so far proposed. Unless stated otherwise, the solutions cited are described in the usual framework of P systems with active membranes using 2-division, with three electrical charges, without change of membrane labels, without cooperation, and without priority.

M.J. Pérez-Jiménez, A. Romero-Jiménez, and F. Sancho-Caparrini present in [34] a linear time solution to *SAT* in a semi-uniform manner but in such a way that we can easily decribe that solution in an uniform manner (see [35]). Other interesting uniform linear time solutions to *SAT* are given by:

- A. Alhazov [1], through P systems with active membranes and using only two electrical charges.
- L. Pan, A. Alhazov, and T.O. Ishdorj [18], in the framework of P systems with active membranes, and without division, without polarizations, but using three types of membrane rules (separation, merging, and release).
- L. Pan and T.O. Ishdorj [19], through P systems with separation rules instead of division rules, in two different cases: in the first, using polarizations and separation rules; and in the second without polarizations and without change of membrane labels, but using separation rules with change of membrane labels.
- Gh. Păun, M.J. Pérez-Jiménez, and A. Riscos-Núñez [26], using P systems with tables of rules (each membrane is associated with several sets of rules, one of which is nondeterministically chosen in each computation step); in particular, they consider tables with obligatory rules, which are distinguished rules which must be applied at least once when the table is applied.

In [36], M.J. Pérez-Jiménez, A. Romero-Jiménez, and F. Sancho-Caparrini present the first linear time uniform solution to the *VALIDITY* problem, for formulas in conjuctive normal form.

Different efficient solutions to graph problems (*Vertex Cover*, *Clique*) in a uniform way are presented by A. Alhazov, C. Martín-Vide, and L. Pan in [3].

The first efficient and uniform solutions to numerical **NP**-complete problems were given by M.J. Pérez-Jiménez and A. Riscos-Núñez in [30] where a solution to the *Knapsack* problem was presented. A uniform solution to *Multiset 0–1 Knapsack* problem is given in the same framework by L. Pan and C. Martín-Vide in [20].

Other uniform solutions to numerical problems are the following:

- M.J. Pérez-Jiménez and A. Riscos-Núñez provide in [31] a solution to *Subset Sum*, and in [7] give a solution to the *Partition* problem.
- M.J. Pérez-Jiménez and F.J. Romero-Campero present in [32] a solution to the *Bin Packing* problem, and in [33] give a solution to *Common Algorithmic Problem*, a problem with a property of local universality in the sense that many other interesting **NP**-complete problems can be reduced to it in linear time.

In this section we present a linear time uniform solution to the *Knapsack* problem, and a quadratic time uniform solution to the decision version of the *Common Algorithmic Problem.*

5.1 A Linear Time Solution to the Decision Knapsack Problem

The decision *Knapsack* problem (0/1) is as follows:

> *Given a knapsack of capacity $k \in \mathbf{N}$, a set A of n elements, where each element has a "weight" $w_i \in \mathbf{N}$ and a "value" $v_i \in \mathbf{N}$, and given a constant $c \in \mathbf{N}$, decide whether or not there exists a subset of A such that its weight does not exceed k and its value is greater than or equal to c.*

The instances of the problem will be represented by tuples of the form $(n, (w_1, \ldots, w_n), (v_1, \ldots, v_n), k, c)$, where n is the size of the set A, $(w_1, \ldots, w_n)$ and $(v_1, \ldots, v_n)$ are the weights and the values, respectively, of the elements from A, and k and c are the constants mentioned above. The funtions w and v can be extended to every subset of A in a natural way from the data in the instance. Also, we consider the size of the instance to be $\langle n, k, c\rangle$ (where $\langle\rangle$ is a polynomial encoding of the tuple, for example, using the Cantor pair function).

In the discussion that follows we will construct a family of recognizer P systems with active membranes using 2-division and with input that solves the decision *Knapsack* problem (0/1) in linear time. Specifically, given the size n of the set A and two constants k and c, we construct a P system $\Pi(n, k, c)$ that solves all the instances of size $\langle n, k, c\rangle$, given an appropriate encoding for the input membrane of the weights and values of the elements from A. The functioning of this P system can be divided in the following stages:

(a) Generate all the subsets of A, computing simultaneously the weights and the values of the subsets.

(b) For all the subsets check if the condition $w(B) \leq k$ holds.
(c) For all the subsets of A that satisfy the first condition, check if the condition $v(B) \geq c$ holds.
(d) Answer *yes* or *no* according to the results of the two checking stages.

The P system $\Pi(n, k, c)$ is defined as follows.

- The input alphabet is $\Sigma(n, k, c) = \{x_1, \ldots, x_n, y_1, \ldots, y_n\}$.
- The working alphabet is

$$\begin{aligned}\Gamma(n,k,c) = \{&a_0, a, \bar{a}_0, \bar{a}, b_0, b, \bar{b}_0, \bar{b}, \hat{b}_0, \hat{b}, d_+, d_-, e_0, \ldots, e_n,\\ &q_0, \ldots, q_{2k+1}, q, \bar{q}, \bar{q}_0, \ldots, \bar{q}_{2c+1}, x_0, \ldots, x_n, y_0, \ldots, y_n,\\ &yes, no, z_0, \ldots, z_{2n+2k+2c+6}, \#\}.\end{aligned}$$

- The set of labels is $\{s, e\}$.
- The initial membrane structure is $[_s [_e\ ]_e^0]_s^0$ (each membrane with label e is said to be internal).
- The input membrane is the one with label e.
- The initial multisets associated with the membranes are $\mathcal{M}_s = z_0$ and $\mathcal{M}_e = e_0\bar{a}^k\bar{b}^c$.
- The rules are:

(a.1) $[_e e_i]_e^0 \rightarrow [_e q]_e^- [_e e_i]_e^+$, for $0 \leq i \leq n$,
(a.2) $[_e e_i]_e^+ \rightarrow [_e e_{i+1}]_e^0 [_e e_{i+1}]_e^+$, for $0 \leq i \leq n-1$,
(a.3) $[_e x_0 \rightarrow \lambda]_e^+, \quad [_e x_i \rightarrow x_{i-1}]_e^+$, for $1 \leq i \leq n$,
(a.4) $[_e x_0 \rightarrow \bar{a}_0]_e^0$,
(a.5) $[_e y_0 \rightarrow \lambda]_e^+, \quad [_e y_i \rightarrow y_{i-1}]_e^+$, for $1 \leq i \leq n$,
(a.6) $[_e y_0 \rightarrow \bar{b}_0]_e^0$,
(a.7) $[_e q \rightarrow \bar{q}q_0]_e^-, \quad [_e \bar{a}_0 \rightarrow a_0]_e^-, \quad [_e \bar{a} \rightarrow a]_e^-, \quad [_e \bar{b}_0 \rightarrow \hat{b}_0]_e^-, \quad [_e \bar{b} \rightarrow \hat{b}]_e^-$,

(b.1) $[_e a_0]_e^- \rightarrow [_e]_e^0 \#, \quad [_e a]_e^0 \rightarrow [_e]_e^- \#$,
(b.2) $[_e q_{2j} \rightarrow q_{2j+1}]_e^-$, for $0 \leq j \leq k$,
(b.3) $[_e q_{2j+1} \rightarrow q_{2j+2}]_e^0$, for $0 \leq j \leq k-1$,
(b.4) $[_e q_{2j+1}]_e^- \rightarrow [_e]_e^+ \#$, for $0 \leq j \leq k$,
(b.5) $[_e \bar{q} \rightarrow \bar{q}_0]_e^+, \quad [_e \hat{b}_0 \rightarrow b_0]_e^+, \quad [_e \hat{b} \rightarrow b]_e^+, \quad [_e a \rightarrow \lambda]_e^+$,

(c.1) $[_e b_0]_e^+ \rightarrow [_e]_e^0 \#, \quad [_e b]_e^0 \rightarrow [_e]_e^+ \#$,
(c.2) $[_e \bar{q}_{2j} \rightarrow \bar{q}_{2j+1}]_e^+$, for $0 \leq j \leq c$,
(c.3) $[_e \bar{q}_{2j+1} \rightarrow \bar{q}_{2j+2}]_e^0$, for $0 \leq j \leq c-1$,
(c.4) $[_e \bar{q}_{2c+1}]_e^+ \rightarrow [_e]_e^0 yes, \quad [_e \bar{q}_{2c+1}]_e^0 \rightarrow [_e]_e^0 yes$,

(d.1) $[_s z_i \rightarrow z_{i+1}]_s^0$, for $0 \leq i \leq 2n+2k+2c+5$, (d.2) $[_s z_{2n+2k+2c+6} \to d_+ d_-]_s^0$,

(d.3) $[_s d_+]_s^0 \to [_s]_s^+ d_+, \quad [_s d_- \to no]_s^+$,

(d.4) $[_s yes]_s^+ \to [_s]_s^0 yes, \quad [_s no]_s^+ \to [_s]_s^0 no.$

Let us see if this P system solves the *Knapsack* problem for every instance of size $\langle n, k, c\rangle$. First of all we must define a polynomial encoding of the problem into the family $\mathbf{\Pi}$ in order to give a suitable input to the system. Given an instance $u = (n, (w_1, \dots, w_n), (v_1, \dots v_n), k, c)$ of the *Knapsack* problem, we define $cod(u) = x_1^{w_1} \dots x_n^{w_n} y_1^{v_1} \dots y_n^{v_n}$. Now we will informally describe how the system $\Pi(n, k, c)$ with input $cod(u)$ works.

As we have just shown, the objects x_i and y_i will represent the weights and values of the elements of A. On the other hand, the objects $\bar{a}$ and $\bar{b}$ (the first will change to a for the second stage, while the second will change to $\hat{b}$ for the second stage and to b for the third stage), included from the beginning by definition of the system, represent the constants k and c, respectively.

In the first stage of the computation the initial internal membrane is continuosly divided by means of the rules from (a.1) and (a.2). These membrane divisions are controlled by the objects e_i, which represent the elements of the set A being considered. The charges of the newly created membranes indicate whether or not the element has been included in the subsets of A that are being generated. We show in Figure 2 an example for $n = 4$ of the membrane generation tree that is obtained.

Let us introduce the concept of subset *associated* with an internal membrane through the following recursive definition:

- The subset associated with the initial internal membrane is the empty one.
- When an object e_j appears in a neutrally charged internal membrane (with $j < n$), the jth element of A is selected and added to the previously associated subset. Once the stage is over, the associated subset will not be modified any more.
- When a division rule is applied, the two newborn internal membranes inherit the associated subset from the original one.

What we intend to get is a single internal membrane for each subset of A but, as we will later see, the membranes are not generated simultaneously (it can be shown that the membrane corresponding to the subset $\{a_{i_1}, \dots, a_{i_r}\}$ is generated at the $(i_r + r + 2)$th computation step).

After a division rule from (a.1) is applied, the two new membranes will behave in quite different ways. On the one hand, in the negatively charged membrane (we have marked such membranes in Figure 2 with a circle) the first stage ends, and in the next step the rules from (a.5) will be applied, renaming the objects to prepare the third stage. This is a significant step, so we will designate as *relevant* those membranes that have negative charge and contain an object q_0. A relevant membrane will not further divide during the computation, and its associated subset will remain unchanged. On the other hand, the positively charged membranes will continue the generation of

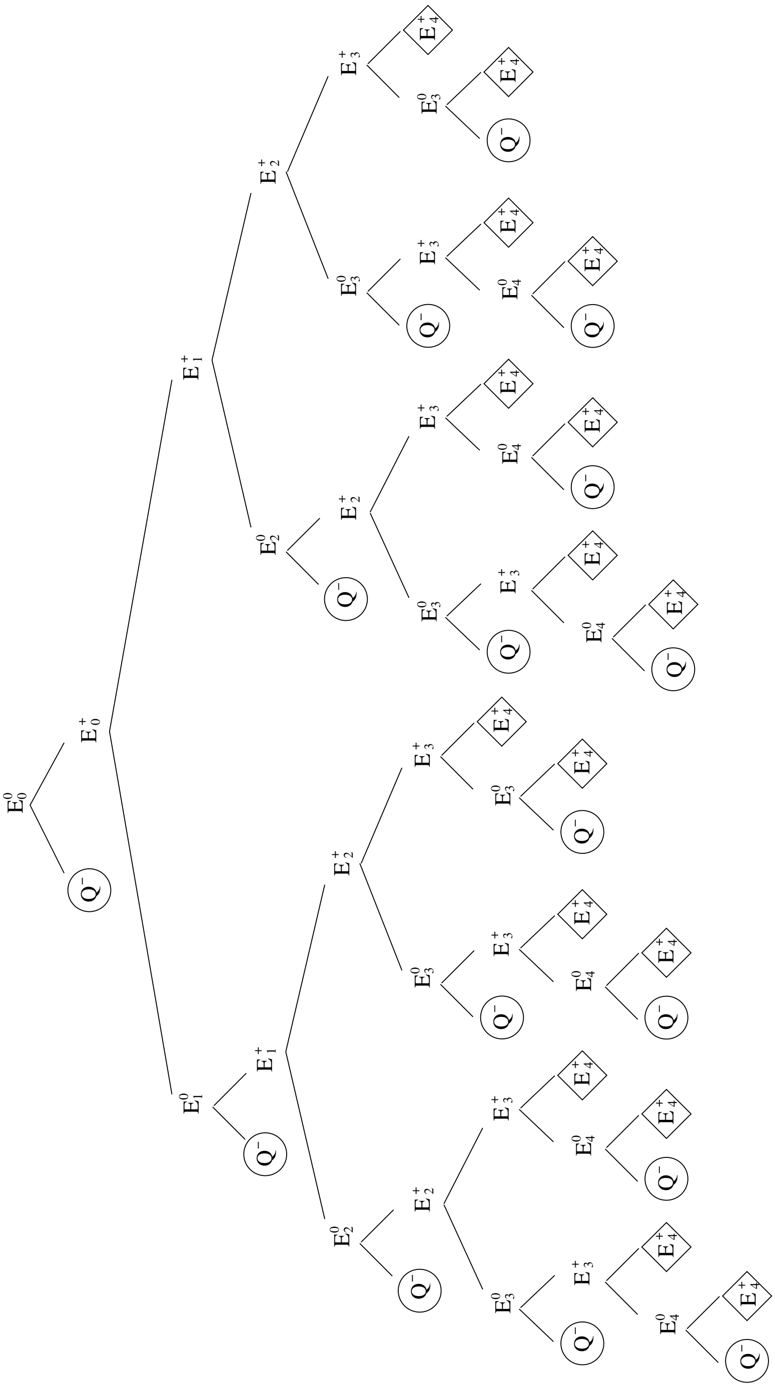

Fig. 2. Membrane generation tree for $n = 4$.

subsets of A; they will give rise to membranes associated with subsets that are obtained by adding elements of A of index $i+1$ or greater to the current subset. Note that if $i = n$, then the membrane cannot continue generating subsets, since it is not possible to add elements of indices greater than n. This has been taking into account because, as there is no rule from (a.1) or (a.2) that can be applied to an object e_n in a positively charged membrane (see the membranes surrounded by a diamond in Figure 2), the membrane halts.

Thus, since the indices of objects e_i never decrease, the relevant membranes are generated in a kind of lexicographic order, in the following sense: if the jth element of A has already been added to the associated subset, then no element with index lower than j will be added later to the subset associated with that membrane, or to the subsets associated with its descendants.

The generation of subsets and the computation of their weights and values are carried out in parallel. In fact, there is only a gap of one step of computation between the time an element is added to the associated subset and the time the new weight and value of the subset are updated. The rules involved here are those from (a.3–6). Recall that the objects x_i represent the weights of the elements of A and the objects y_i represent their values. The rules from (a.3) and (a.5) performed a rotation of these objects so that the objects x_0 and y_0 correspond to the weight and the value of the current element of A; that is, if a positively charged internal membrane contains an object e_i, meaning that the element of A being considered is the ith one, then the objects x_0 represent the weight of the element and the objects y_0 represent the value. On the other hand, if an internal membrane contains an object e_i and is neutrally charged, it means that the ith element of A is going to be added to the subset associated with the membrane. Then the weight and value of the subset have to be updated. This is done by means of the rules from (a.4) and (a.6), which transform the objects x_0 and y_0 into objects $\bar{a}_0$ and $\bar{b}_0$ representing the partial weight and value of the subset (note that the objects $\bar{a}_0$ will change to a_0 for the second stage, and the objects $\bar{b}_0$ will change to $\hat{b}_0$ for the second stage and to b_0 for the third stage).

The purpose of the rules from (a.5) is to prevent the rules for the second stage from being applied to an internal membrane that has not yet reached the end of the first stage. For that, we rename the objects obtained at the end of the latter, $q, \bar{a}_0, \bar{a}, \bar{b}_0$, and $\bar{b}$, to objects $\bar{q}, q_0, a_0, a, \hat{b}_0$, and $\hat{b}$ that are not used in it. The objects $\bar{q}, \hat{b}_0$, and $\hat{b}$ are not used until the third stage, so they remain unchanged through the second stage. The objects a_0 and a are used in a loop that checks if the weight of the subset of A being considered satisfies the condition of being less than or equal to k. The rules from (b.1) eliminate alternatively (the alternation is controlled by the negative and neutral charges of the membrane), at each step of the loop, one by one, until we run out of one (or both) of the objects. It is easy to see that the required condition is verified if and only if the loop can make an even number of steps. The objects q_i are then used to count the number of steps performed by the loop.

Let us see in a more detail how this loop works. Let B be a subset of a certain weight w_B. The evolution of the relevant membrane associated with it along the second stage is described in Table 1.

Multiset	Charge	Parity of q_i
$q_0 a_0^{w_B} a^k \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	EVEN
$q_1 a_0^{w_B-1} a^k \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	0	ODD
$q_2 a_0^{w_B-1} a^{k-1} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	EVEN
$\vdots$	$\vdots$	$\vdots$
$q_{2j} \ a_0^{w_B-j} a^{k-j} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	EVEN
$q_{2j+1} \ a_0^{w_B-(j+1)} a^{k-j} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	0	ODD
$\vdots$	$\vdots$	$\vdots$

Table 1. Comparison of weight with k.

Note 1. Observe that the index of q_i coincides with the total number of copies of a and a_0 that have already been erased during the comparison.

Note 2. If $B = \{a_{i_1}, \ldots, a_{i_r}\}$ with $i_r \neq n$, then there will be in the multiset some objects x_j and y_j, for $1 \leq j \leq n - i_r$, but they are irrelevant for this stage and therefore they will be omitted.

If the number w_B of objects a_0 is less than or equal to the number k of objects a, then the result of this stage is successful and we can proceed with the next stage. This situation is described in Table 2.

Multiset	Charge	Parity of q_i
$\vdots$	$\vdots$	$\vdots$
$q_{2w_B-1} \ a^{k-w_B+1} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	0	ODD
$q_{2w_B} \ a^{k-w_B} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	EVEN
$q_{2w_B+1} \ a^{k-w_B} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	ODD
$a^{k-w_B} \ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$+$	ODD

Table 2. Weight less than or equal to k.

If the number w_B of objects a_0 is greater than the number k of objects a, then every time the rules from (b.2) can be applied (that is, for $j = 0, \ldots, k$), the first rule from (b.1) will also be applied. Thus, we can never get to a situation where the index of the counter q_i is an odd number and the charge of the internal membrane is negative. This means that the rule (b.4) can never be applied, and moreover, the membrane gets *blocked* (it will not evolve anymore during the computation). This situation is described in Table 3.

Multiset	Charge	Parity of q_i
$\vdots$	$\vdots$	$\vdots$
$q_{2k-1}\ a_0^{w_B-k}a\ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	0	ODD
$q_{2k}\ a_0^{w_B-k}\ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	$-$	EVEN
$q_{2k+1}\ a_0^{w_B-(k+1)}\ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	0	ODD

Table 3. Weight greater than k.

Let us suppose that the second stage has successfully finished in an internal membrane. That means that this membrane encodes a subset $B \subseteq A$ such that $w(B) \leq k$. Then, after applying the rule (b.4), this membrane gets a positive charge. For the objects used in this stage, so as not to fire the rules corresponding to the next stage, they are renamed by means of the rules from (b.5). In this way, the objects $\hat{b}_0$ are transformed to objects b_0 and the objects $\hat{b}$ to objects b. The counter $\bar{q}_i$ is initialized to $\bar{q}_0$.

The third stage works in a similar way as the second one, using the rules from (c.1), (c.2), and (c.2) corresponding to the rules from (b.1), (b.2), and (b.3), respectively. The end of the stage, however, is different. In this stage, we have to check if the number of objects b_0, corresponding to the value of the subset of A associated with the membrane, is greater than or equal to the number of objects b, corresponding to the constant c. Therefore, to pass to the final stage the two rules from (c.1) must have been applied c times each. Consequently, the rules from (c.2) and (c.3) take the counter of the loop to q_{c+1}, when the rules from (c.4) send it out to the skin as a *yes* object. Table 4 summarizes all the process described above.

Finally, rules from (d.1–3) are associated with the skin membrane and take care of the output stage. The counter z_i, used by the rules from (d.1) and (d.2), waits through $2n + 2k + 2c + 7$ steps ($2n + 3$ steps for the first stage, $2k + 2$ steps for the second stage and $2c + 2$ for the third stage). After all these steps are performed, we are sure that all the inner membranes have already finished their checking stages (or have already got blocked), and thus, the output process is activated.

Then, the skin will be neutrally charged and will contain the objects d_+ and d_-. Furthermore, some objects *yes* will be present in the skin if and only if both checking stages have been successful in at least one internal membrane.

Multiset	Charge	Parity of q_i
$\bar{q}_0 b_0^{v_B} b^c$	$+$	EVEN
$\bar{q}_1 b_0^{v_B-1} b^c$	0	ODD
$\vdots$	$\vdots$	$\vdots$
$\bar{q}_{2c-1} b_0^{v_B-c} b$	0	ODD
$\bar{q}_{2c} b_0^{v_B-c}$	$+$	EVEN
$\bar{q}_{2c+1} b_0^{v_B-(c+1)}$ (if $v_B > c$)	0	ODD
or $\bar{q}_{2c+1}$ (if $v_B = c$)	$+$	ODD
$b_0^{v_B-(c+1)}$ or $\emptyset$	$+$	

Table 4. Comparison of value with c.

The output process then begins. First the object d_+ is sent out to the environment, giving a positive charge to the skin. Then the object d_- evolves to *no* inside the skin and, simultaneously, if there exists any object *yes* present in the membrane, it is sent out of the system, giving a neutral charge to the skin and making the system halt (in particular, further evolution of the object *no* is avoided).

Otherwise, if none of the membranes has successfully passed both checking stages, then there will be no object *yes* present in the skin when the output stage begins. Thus, after the object *no* is generated, the skin will still have a positive charge, so the object will be sent out to the environment and the system will halt.

Now we are going to justify that the family $\mathbf{\Pi} = (\Pi(n,k,c))_{n,k,c\in\mathbf{N}}$ solves the decision *Knapsack* problem (0/1) in linear time.

First, the above description of the system is computable in a uniform way. It is also a polynomial description, since the size of the input alphabet is $2n$; the size of the working alphabet is $5n + 4k + 4c + 31$; the size of the set of labels is 2; the number of membranes in the initial membrane structure is 2; the maximum cardinal of the initial multisets is 3; the total number of evolution rules is $6n + 5k + 4c + 34$; and the maximum length of a rule is 7. Hence, the family $\mathbf{\Pi}$ is *Turing polynomially uniform.*

Second, the family $\mathbf{\Pi}$ is linearly bounded, since given an instance $u = (n, (w_1, \dots, w_n), (v_1, \dots, v_n), k, c)$ of the problem, the total number of steps performed by the unique computation of the system $\Pi(n,k,c)$ given the input $cod(u)$ is at most $2n + 2k + 2c + 10$.

Third, from the description of the functioning of the P system $\Pi(n,k,c)$ it can be seen that the family $\mathbf{\Pi}$ is sound and complete.

5.2 A Quadratic Time Solution to CADP

The Common Algorithmic Problem (CAP) [8] is the following optimization problem:

> *Let S be a finite set and F be a family of subsets of S, called forbidden sets. Find the cardinality of a maximal subset of S which does not include any set belonging to F.*

The Common Algorithmic Problem can be transformed into a roughly equivalent decision problem by supplying a target value to the quantity to be optimized, and asking the question as to whether or not this value can be attained.

The Common Algorithmic Decision Problem (CADP) is the following decision problem:

> *Let S be a finite set, F be a family of subsets of S, and $k \in \mathbf{N}$. Determine if there exists a subset A of S such that $|A| \geq k$, and which does not include any set belonging to F.*

We will say that a problem X is a subproblem of another problem Y if there exists a linear time reduction from X to Y (using logarithmic bounded space). That is, X is a subproblem of Y if we can pass from the former to the latter by a simple rewriting process.

Next, we present some **NP**-complete decision problems that are subproblems of CADP.

- The Independent Set Decision Problem (ISD): *Given an undirected graph G, and $k \in \mathbf{N}$, determine whether or not G has an independent set of size at least k.*
- The Vertex Cover Decision Problem (VCD): *Given an undirected graph G, and $k \in \mathbf{N}$, determine whether or not G has a vertex cover of size at most k.*
- The Clique Decision Problem (CDP): *Given an undirected graph G, and $k \in \mathbf{N}$, determine whether or not G has a clique of size at least k.*
- The Hamiltonian Path Problem (HPP).
- The Satisfiability Problem (SAT).
- The Tripartite Matching Problem: *Given three sets B, G, and H, each containing n elements, and a ternary relation $T \subseteq B \times G \times H$, determine whether or not there exists a subset T' of T such that $|T'| = n$ and no two triples belonging to T' have a component in common.*

In what follows we will construct a family of recognizer P systems with active membranes using 2-division and with input that solves CADP in polynomial time. Specifically, given the size n of the set S, the size m of the set F, and a constant k, we construct a P system $\Pi(n, m, k)$ that solves the problem for all the instances of size $\langle n, m, k \rangle$, given as input an appropriate encoding of the subsets of S belonging to F. The functioning of this P system can be divided into the following stages:

(a) Generate maximal subsets A of S not including any element of F. For this, we start from the complete set S and eliminate one element from each of the forbidden sets.
(b) For all the previous subsets, compute their cardinality.
(c) Check if any of the subsets has cardinality greater than or equal to k (in fact, we check if the cardinality is greater than $k-1$).
(d) Answer *yes* or *no* according to the results from the previous stage.

The P system $\Pi(n,m,k)$ is defined as follows.

- The input alphabet is $\Sigma(n,m,k) = \{s_{i,j} \mid 1 \le i \le m, 1 \le j \le n\}$.
- The working alphabet is

$$\begin{aligned}\Gamma(n,m,k) = {} & \Sigma(n,m,k) \cup \{a_i \mid 1 \le i \le m\} \cup \{c_i \mid 0 \le i \le 2n+1\} \\ & \cup \{ch_i \mid 0 \le i \le 2k-1\} \cup \{f_j \mid 1 \le j \le n+1\} \\ & \cup \{e_{i,j,l} \mid 1 \le i \le m, 1 \le j \le n, -1 \le l \le j+1\} \\ & \cup \{g_j \mid 0 \le j \le nm+m+1\} \\ & \cup \{z, s_+, s_-, S_+, S_-, S, o, \tilde{O}, O, t, neg, \#, yes, preno, no\}.\end{aligned}$$

- The set of labels is $\{1,2\}$.
- The initial membrane structure is $[_1 [_2 \ ]_2^0]_1^0$ (each membrane with label 2 is said to be internal).
- The input membrane is the one with label 2.
- The initial multisets associated with the membranes are $\mathcal{M}_1 = \lambda$ and $\mathcal{M}_2 = g_0 z^m s_+^n o^{k-1}$.
- The rules are:

(a.1) $[_2 s_{1,j} \to f_j]_2^0$, for $1 \le j \le n$,
(a.2) $[_2 s_{i,j} \to e_{i,j,j}]_2^0$, for $2 \le i \le m, 1 \le j \le n$,
(a.3) $[_2 f_1]_2^0 \to [_2 \#]_2^0 [_2 s_-]_2^+$,
(a.4) $[_2 f_j \to f_{j-1}]_2^0$, for $2 \le j \le n+1$,
(a.5) $[_2 f_j \to \lambda]_2^+$, for $1 \le j \le n+1$,
(a.6) $[_2 e_{i,j,l} \to e_{i,j,l-1}]_2^0$, for $2 \le i \le m, 1 \le j \le n, 0 \le l \le j+1$,
(a.7) $[_2 e_{2,j,l} \to f_{j+1}]_2^+$, for $1 \le j \le n, -1 \le l \le j+1, l \ne 0$,
(a.8) $[_2 e_{i,j,l} \to e_{i-1,j,j+1}]_2^+$, for $3 \le i \le m, 1 \le j \le n, -1 \le l \le j+1, l \ne 0$,
(a.9) $[_2 e_{i,j,0} \to a_{i-1}]_2^+$, for $2 \le i \le m, 1 \le j \le n$,
(a.10) $[_2 z]_2^+ \to [_2]_2^0 \#$,
(a.11) $[_2 a_1]_2^0 \to [_2]_2^+ \#$,
(a.12) $[_2 a_1 \to \lambda]_2^+$, $[_2 a_i \to a_{i-1}]_2^+$, for $2 \le i \le m$,
(a.13) $[_2 g_j \to g_{j+1}]_2^0$, $[_2 g_j \to g_{j+1}]_2^+$, for $0 \le j \le nm+m$,
(a.14) $[_2 g_{nm+m+1} \to c_0 neg]_2^0$,
(a.15) $[_2 neg]_2^0 \to [_2]_2^- \#$,

(a.16) $[_2 s_+ \to S_+]_2^-$, $[_2 s_- \to S_-]_2^-$, $[_2 o \to \tilde{O}]_2^-$,
(a.17) $[_2 z]_2^- \to \#$,

(b.1) $[_2 S_-]_2^- \to [_2]_2^+ \#$, $[_2 S_+]_2^+ \to [_2]_2^- \#$,
(b.2) $[_2 c_i \to c_{i+1}]_2^-$, $[_2 c_i \to c_{i+1}]_2^+$, for $0 \leq i \leq 2n$,
(b.3) $[_2 c_{2n+1} \to ch_0 t]_2^-$,
(b.4) $[_2 t]_2^- \to [_2]_2^0 \#$,
(b.5) $[_2 S_+ \to S]_2^0$, $[_2 \tilde{O} \to O]_2^0$,

(c.1) $[_2 S]_2^0 \to [_2]_2^+ \#$, $[_2 O]_2^+ \to [_2]_2^0 \#$,
(c.2) $[_2 ch_i \to ch_{i+1}]_2^0$, $[_2 ch_i \to ch_{i+1}]_2^+$, for $0 \leq i \leq 2k-2$,
(c.3) $[_2 ch_{2k-1}]_2^+ \to [_2]_2^+ yes$, $[_2 ch_{2k-1}]_2^0 \to [_2]_2^0 preno$,

(d.1) $[_1 yes]_1^0 \to [_1]_1^+ yes$,
(d.2) $[_1 preno \to no]_1^0$,
(d.3) $[_1 no]_1^0 \to [_1]_1^- no$.

Let us see if this P system solves `CADP` for every instance of size $\langle n, m, k\rangle$. First of all we must define a polynomial encoding of the problem into the family $\mathbf{\Pi}$ in order to give a suitable input to the system. Given an instance $u = (S, F, k)$ of the problem, where $S = \{a_1, \ldots, a_n\}$, $F = \{B_1, \ldots, B_m\}$, and $B_i = \{a_{j_1^i}, \ldots, a_{j_{k(i)}^i}\}$, we define $cod(u) = s_{1,j_1^1} \ldots s_{1,j_{k(1)}^1} \ldots s_{m,j_1^m} \ldots s_{m,j_{k(m)}^m}$. That is, the object $s_{i,j}$ introduced in the initial membrane will represent the fact that the set B_i contains the element a_j.

Now we informally describe how the system $\Pi(n, m, k)$ with input $cod(u)$ works.

To perform the first stage we start from the complete set S and make a loop to consider sequentially all the sets $B_1, \ldots, B_m$. Inside this loop we make another loop in which we generate a number of diferent subsets of S obtained by eliminating only one element of the current set B_i.

The core of this stage are the rules from groups (a.3–8). For these rules, the objects f_i represent the elements of the current set B_i, while the objects $e_{i,j,l}$ represent the elements of the sets B_j not yet considered. The purpose of the rules from (a.1) and (a.2) is now clear; we have to change the representation of the sets B_i from the objects $s_{i,j}$ to the objects f_i and $e_{i,j,l}$, and we do this in such a way that B_1 is the first forbidden set considered.

In the rule from (a.3) the object f_1 represents the element to eliminate from the current forbidden set. This rule creates two new membranes, one neutrally charged and the other positively charged. The former means that we have decided not to eliminate the element, so with the rule (a.4) we perform a rotation of the subscripts of the objects f_i, for the elements of B_i to be considered for elimination in a sequential way. The latter membrane means that we have eliminated the element and that we can proceed with the next forbidden set – but before that we have to do several things.

First we eliminate, by means of the rules from (a.5), the remaining objects f_i, since to meet the cardinal maximality condition we do not eliminate any other object of the forbidden set. This takes us to the following question: what if the element eliminated also belonged to a forbidden set B_j not considered yet? In that case the condition $B_j \nsubseteq A$ is fulfilled, and we do not have to eliminate any object from B_j. To control this from happening at the same time as for the elements f_i, we make a rotation of the subscripts of the objects $e_{i,j,l}$ (representing the elements of the forbidden sets not considered yet) so that they are always in correspondence with the objects f_i (representing the elements of the forbidden set being considered). The rotation is done to the third subscript of the objects, using the rules from (a.6), while the two first subscripts keep a record of which element it represented and which forbidden set contained it. In this way, before passing to the next step of the main loop we can "restore" the objects, by means of the rules from (a.7) and (a.8), and, using the objects a_i, we "memorize" which additional forbidden sets B_j satisfy $B_j \nsubseteq A$ by means of the rules from (a.9). Also, the rule from (a.10) uses the object z to count how many of these sets satisfy the previous condition.

Note that when restoring the objects, as described above, the third subscript gets a value which exceeds by one what it should be. This is because before continuing with the next step of the main loop we have to check if the next forbidden set to consider is not included in A; that is, we have to check for the existence of an object a_1. If this is the case, the rule from (a.11) skips that step, changing the polarization of the membrane to positive. We can then restore again the objects $e_{i,j,l}$, using the rules from (a.7) and (a.8), and perform a rotation of the subscripts of the objects a_i, using the rules from (a.12).

To synchronize the finalization of this first stage in all the generated internal membranes, we use the objects g_i as counters (rules from (a.13)). Since the worst case lasts at most $nm + m$ computation steps, when the rule from (a.14) is applied introducing the objects c_0 and neg, we can be sure that the first stage has reached the end in all the internal membranes. The object neg is then sent out to change the polarization of the membrane to negative (rule from (a.15)), so that the second stage can start, before which we make a renaming of objects to obtain the ones that will be used in this stage (rules from (a.17)).

A careful look at how the internal membranes are created by the rule from (a.3) shows two things. The first one is that, when the element of the forbidden set is eliminated (that is, in the membrane with positive charge), an element s_- is introduced. This object counts how many elements have been eliminated. It is also possible to obtain an internal membrane where none of the elements of the forbidden set being considered have been eliminated. In this case, when the end of the first stage is reached in the membrane, there will be z objects left. The rule from (a.17) allows us to dissolve these disturbing membranes.

The task of the second stage is to compute the cardinality of the subsets of S that are associated with the internal membranes created in the first stage. We have seen in the previous paragraph that the object s_- (changed to S_- in this stage) represents the number of objects eliminated. On the other hand, the object s_+ (changed to S_+ in this stage) represents the total cardinality of the set S, and this is why this object has multiplicity n in the initial multiset of the initial internal membrane. It is clear then that we have only to do a subtraction. For that, the rules from (b.1) alternatively erase the objects S_- and S_+. The latter can only be erased after the former has been erased. Hence, we have only to wait $2n$ computation steps to get the result of the subtraction. This is the purpose of the rules from (b.2), that use the objects c_i as counters. When the object c_{2n+1} appears, we can pass to the next stage, so the rule from (b.3) introduces the initial counter for that stage, ch_0, and an object t that allows us to change the polarization of the membrane to neutral (rule from (b.4)), which in turn allows us to rename the objects to obtain the ones used in the third stage (rules from (b.5)).

To check if the cardinality of the subset is greater than or equal to k, we check if it is greater than $k-1$. This is why we keep $k-1$ objects o (now transformed to O) from the beginning of the computation. The rules from (c.1) erase once and again an object S, changing the polarization to positive, and an object O, changing the polarization back to neutral. If we wait $2k-2$ computation steps (this is done by the rules from (c.2) using the objects ch_i as counters), the comparison is finished. If the final polarization is positive, then the cardinality is greater than $k-1$, so the rule from (c.3) sends an object *yes* to the skin. Otherwise, what is sent is an object *preno*.

Finally, the output stage is very simple. We have only to be careful, looking for the positive answers before looking for the negative ones. If there is an object *yes* in the skin, it is sent out to the environment, the charge of the skin membrane is changed to positive (rule from (d.1)), and the system halts. If not, the objects *preno* are changed to objects *no* (rule from (d.2)) and one of them is sent out to the environment, the charge of the skin membrane is changed to negative (rule from (d.3)), and the system halts.

Now we are going to justify that the family $\mathbf{\Pi} = (\Pi(n,m,k))_{n,m,k\in\mathbf{N}}$ solves `CADP` in polynomial time.

First, the above description of the system is computable in a uniform way. It is also a polynomial description, since the size of the input alphabet is mn; the size of the working alphabet is at most $mn^2+4mn-m+3n+2k+18$; the size of the set of labels is 2; the number of membranes in the initial membrane structure is 2; the maximum cardinality of the initial multisets is $m+n+k$; the total number of evolution rules is at most $2mn^2+8mn+3m-2n^2+n+4k+23$; and the maximum length of a rule is 7. Hence, the family $\mathbf{\Pi}$ is *Turing polynomially uniform*.

Second, the family $\mathbf{\Pi}$ is quadratically bounded, since, given an instance $u = (S, F, k)$ of the problem, the total number of steps performed by the unique computation of the system $\Pi(n, m, k)$, given the input $cod(u)$, lasts at most $mn + m + 2n + 2k + 8$ steps.

Third, from the description of the functioning of the P system $\Pi(n, m, k)$ it can be seen that the family $\mathbf{\Pi}$ is sound and complete.

6 Conclusions

The possibility of finding a systematic and suitable framework to address in an efficient way the resolution of many practical problems that are presumably intractable (unless **P**=**NP**) is studied in this chapter.

We consider P systems as recognizer devices. Solutions to **NP**-complete problems are looked for in this framework by making use of appropriate families of P systems that can be constructed in a semi-uniform or uniform way. In this chapter we have discussed the differences between these constructions, and have presented a survey of the different solutions known in the current literature of membrane systems. Also, we have described in some detail two semi-uniform solutions, to `SAT` and `HPP`, and two uniform solutions, to the *Knapsack* problem and the *Common Algorithmic problem.*

Acknowledgement. The authors wish to acknowledge the support of the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds.

References

1. A. Alhazov: P Systems with Active Membranes and Two Polarizations. In [27], 20–35.
2. A. Alhazov, T.-O. Ishdorj: Membrane Operations in P Systems with Active Membranes. In [27], 37–52.
3. A. Alhazov, C. Martín-Vide, L. Pan: Solving Graph Problems by P Systems with Restricted Elementary Active Membranes. In *Aspects of Molecular Computing* (N. Jonoska, Gh. Păun, G. Rozenberg, eds.), LNCS 2950, Springer, Berlin, 2004, 1–22.
4. J. Castellanos, Gh. Păun, A. Rodríguez-Patón: P Systems with Worm-Objects. *IEEE 7th International Conference on String Processing and Information Retrieval, SPIRE 2000*, La Coruña, Spain, 64–74.
5. A. Cordón-Franco, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F. Sancho-Caparrini: Implementing in Prolog an Effective Cellular Solution for the Knapsack Problem. In [14], 140–152.
6. E. Czeizler: Self-Activating P Systems. In *Membrane Computing. International Workshop WMC-CdeA 2002, Curtea de Argeş, Romania, August 2002, Revised Papers* (Gh. Păun, G. Rozenberg, A, Salomaa, C. Zandron, eds.), LNCS 2597, Springer, Berlin, 2003, 234–246.

7. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: A Fast P System for Finding a Balanced 2-Partition. *Soft Computing*, 9, 7 (2005).
8. T. Head, M. Yamamura, S. Gal: Aqueous Computing: Writing on Molecules. *Proceedings of the Congress on Evolutionary Computation 1999*, IEEE Service Center, Piscataway, NJ, 1999, 1006–1010.
9. M. Ito, C. Martín-Vide, Gh. Păun: Characterization of Parikh Sets of ET0L Languages in Terms of P Systems. In *Words, Semigroups, and Transducers* (M. Ito, Gh. Păun, S. Yu, eds.), World Scientific, Singapore, 2001, 239–254.
10. S.N. Krishna, R. Rama: A Variant of P Systems with Active Membranes: Solving NP-Complete Problems. *Romanian Journal of Information Science and Technology*, 2, 4 (1999), 357–367.
11. S.N. Krishna, R. Rama: P Systems with Replicated Rewriting. *Journal of Automata, Languages and Combinatorics*, 6, 1 (2001), 345–350.
12. S.N. Krishna, R. Rama: Breaking DES Using P Systems. *Theoretical Computer Science*, 299, 1-3 (2003), 495–508.
13. M. Madhu, K. Kristhivasan: P Systems with Membrane Creation: Universality and Efficiency. In *Proceedings of Third International Conference on Universal, Machines and Computations*, Chişinău, Moldova, 2001 (M. Margenstern, Y. Rogozhin, eds.), LNCS 2055, Springer, Berlin, 2001, 276–287.
14. C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Membrane Computing. International Workshop WMC2004, Tarragona, Spain, July 2003, Revised Papers*. LNCS 2933, Springer, Berlin, 2004,
15. A. Obtulowicz: Deterministic P Systems for Solving SAT Problem. *Romanian Journal of Information Science and Technology*, 4, 1-2 (2001), 551–558.
16. A. Obtulowicz: On P Systems with Active Membranes: Solving the Integer Factorization Problem in a Polynomial Time. In *Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View* (C.S. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 2235, Springer, Berlin, 2001, 267–285.
17. A. Obtulowicz: Note on Some Recursive Family of P Systems with Active Membranes. Submitted, 2004.
18. L. Pan, A. Alhazov, T.-O. Ishdorj: Further Remarks on P Systems with Active Membranes, Separation, Merging, and Release Rules. In [27], 316–324.
19. L. Pan, T.-O. Ishdorj: P Systems with Active Membranes and Separation Rules. *Journal of Universal Computer Science*, 10, 5 (2004), 630–649.
20. L. Pan, C. Martín-Vide: Solving Multiset 0–1 Knapsack Problem by P Systems with Input and Active Membranes. In [27], 342–353.
21. A. Păun: On P Systems with Membrane Division. In *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer, London, 2000, 187–201.
22. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* Nr. 208, 1998.
23. Gh. Păun: Computing with Membranes: Attacking **NP**-Complete Problems. In *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer, London, 2000, 94–115.
24. Gh. Păun: P systems with Active Membranes: Attacking **NP**-Complete Problems. *Journal of Automata, Languages and Combinatorics*, 6, 1 (2001), 75–90.
25. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.

26. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez: P Systems with Tables of Rules. In *Theory is Forever, Essays Dedicated to Arto Salomaa on the Ocassion of His 70th Birthday* (J. Karhumaki, H. Maurer, Gh. Păun, G. Rozenberg, eds.), LNCS 3113, Springer, Berlin, 2004, 235–249.
27. Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, eds.: *Proceedings of the Second Brainstorming Week on Membrane Computing*, Sevilla, February 2004, Report RGNC 01/04, Univ. of Sevilla, 2004.
28. Gh. Păun, G. Rozenberg: A Guide to Membrane Computing. *Theoretical Computer Science*, 287 (2002), 73–100.
29. Gh. Păun, Y. Suzuki, H. Tanaka, T. Yokomori: On the Power of Membrane Division in P Systems. *Theoretical Computer Science*, 324, 1 (2004), 61–85.
30. M.J. Pérez-Jiménez, A. Riscos-Núñez: A Linear Time Solution to the Knapsack Problem Using Active Membranes. In [14], 250–268.
31. M.J. Pérez-Jiménez, A. Riscos-Núñez: Solving the Subset-Sum Problem by P Systems with Active Membranes. *New Generation Computing*, in press.
32. M.J. Pérez-Jiménez, F.J. Romero-Campero: An Efficient Family of P Systems for Packing Items Into Bins. *Journal of Universal Computer Science*, 10, 5 (2004), 650–670.
33. M.J. Pérez-Jiménez, F.J. Romero-Campero: Attacking the Common Algorithmic Problem by Recognizer P Systems. In *Pre-proceedings of the Machines, Computations and Universality, MCU'2004 (abstracts)*, September 21-26, 2004, Sankt Petesburg, p. 27.
34. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: *Teoría de la Complejidad en Modelos de Computación con Membranas*. Ed. Kronos, Sevilla, 2002.
35. 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.
36. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: Solving VALIDITY Problem by Active Membranes with Input. In *Proceedings of the Brainstorming Week on Membrane Computing*, Tarragona, February 2003 (M. Cavaliere, C. Martín-Vide, Gh. Păun, eds.) Report GRLMC 26/03, 2003, 279–290.
37. A. Romero-Jiménez: *Complexity and Universality in Cellular Computing Models*. PhD. Thesis, University of Seville, Spain, 2003.
38. A. Romero-Jiménez, M.J. Pérez-Jiménez: Simulating Turing Machines by P Systems with External Output. *Fundamenta Informaticae*, 49, 1-3 (2002), 273–287.
39. P. Sosik: The Computational Power of Cell Division. *Natural Computing*, 2, 3 (2003), 287–298.
40. C. Zandron, C. Ferreti, G. Mauri: Solving NP-Complete Problems Using P Systems with Active Membranes. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), Springer, London, 2000, 289–301.
41. C. Zandron, G. Mauri, C. Ferreti, Universality and Normal Forms on Membrane Systems. In *Proceedings of International Workshop on Grammar Systems, 2000* (R. Freund, A. Kelemenova, eds.), Bad Ischl, Austria, July 2000, 61–74.