A Linear-Time Solution to the Knapsack Problem Using P Systems with Active Membranes

Mario J. Pérez-Jiménez and Agustin Riscos-Núñez

Dpto. de Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática. Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, España
{marper, ariscosn}@us.es

Abstract. Up to now, P systems dealing with numerical problems have been rarely considered in the literature. In this paper we present an effective solution to the Knapsack problem using a family of deterministic P systems with active membranes using 2-division. We show that the number of steps of any computation is of linear order, but polynomial time is required for pre-computing resources.

1 Introduction

Membrane Computing is an emergent branch in the field of Natural Computing. Since Gh. Păun introduced it (see [3]) much work has been done, with motivations and tools coming from various directions. Computer scientists, biologists, formal linguists and complexity theoreticians have contributed enriching the field with their different points of view.

The present paper is focused on the design of a family of P systems that solves a numerical NP-complete problem, and on the formal verification of this solution. The idea is inspired from the solution presented in [5] for the Subset-Sum problem.

The analysis of the solution presented here will be done from the point of view of the complexity classes. A *complexity class* for a model of computation is a collection of problems that can be solved (or languages that can be decided) by some devices of this model with *similar* computational resources.

In this paper we present a *polynomial complexity class* in cellular computing with membranes inspired in some ideas of Gh. Păun ([3], section 7.1) discussed with some members of the Research Group on Natural Computing from the University of Seville. This class allows us to detect some intrinsic difficulties of the resolution of a problem in the model above mentioned.

The paper is organized as follows: first a formal definition of recognizer P systems is given in the next section; then, in section 3 the polynomial complexity class $\mathbf{PMC}_{\mathcal{AM}}$ is introduced; in sections 4 and 5 a cellular solution for the Knapsack problem is presented, together with some comments; a computational study of the solution is developed in section 6; some consequences of this study together with the conclusions are given in sections 7 and 8.

2 Preliminaries

Recall that a decision problem, X, is a pair (I_X, θ_X) such that I_X is a language over a finite alphabet (the elements of I_X are called *instances*) and θ_X is a total boolean function over I_X.

Definition 1. *A* P system with input *is a tuple* (Π, Σ, i_Π)*, where:*

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

Definition 2. *Let* (Π, Σ, i_Π) *be a P system with input. Let* Γ *be the working alphabet of* Π*,* μ *the membrane structure and* $\mathcal{M}_1, \ldots, \mathcal{M}_p$ *the initial multisets of* Π*. Let* m *be a multiset over* Σ*. The* initial configuration of (Π, Σ, i_Π) with input m *is* (μ_0, M_0)*, where* $\mu_0 = \mu$*,* $M_0(j) = \mathcal{M}_j$*, for each* $j \neq i_\Pi$*, and* $M_0(i_\Pi) = \mathcal{M}_{i_\Pi} \cup m$*.*

Remark 1. We will denote by I_Π the set of all inputs of the P system Π. That is, I_Π is a collection of multisets over Σ.

The computations of a P system with input m, are defined in a natural way. The only novelty is that the initial configuration must be the initial configuration of the system associated with the input multiset $m \in M(\Sigma)$.

In the case of P systems with input and with external output, the concept of computation is introduced in a similar way but with a slight variant. In the configurations, we will not work directly with the membrane structure μ but with another structure associated with it including, in some sense, the environment.

Definition 3. *Let* $\mu = (V(\mu), E(\mu))$ *be a membrane structure. The* membrane structure with environment *associated with* μ *is the rooted tree* $Ext(\mu)$ *such that:* (a) *the root of the tree is a new node that we will denote env;* (b) *the set of nodes is* $V(\mu) \cup \{env\}$*; and* (c) *the set of edges is* $E(\mu) \cup \{\{env, skin\}\}$*. The node env is called* environment *of the structure* μ*.*

Note that we have only included a new node representing the environment which is only connected with the skin, while the original membrane structure remains unchanged. In this way, every configuration of the system contains information about the contents of the environment.

Definition 4. *An* accepting P system *is a P system with input,* (Π, Σ, i_Π)*, and with external output, such that the output alphabet contains only two elements:* Yes *and* No*.*

This definition is stated in a general way, but in this paper P systems with active membranes will be used. We refer to [3] (see chapter 7) for a detailed definition of evolution rules, transition steps, and configurations in this model.

Now let us define the *Output* function for our P systems. Given a computation $\mathcal{C} = \{C^i\}_{i<r}$, we denote by M^j_{env} the content of the environment in the configuration C^j.

Definition 5. *The output of a computation* $\mathcal{C} = \{C^i\}_{i<r}$ *is:*

$$Output(\mathcal{C}) = \begin{cases} Yes, & if\ \mathcal{C}\ is\ halting,\ Yes \in M^{r-1}_{env}\ and\ No \notin M^{r-1}_{env}, \\ No, & if\ \mathcal{C}\ is\ halting,\ No \in M^{r-1}_{env}\ and\ Yes \notin M^{r-1}_{env}, \\ not\ defined, & otherwise. \end{cases}$$

If $\mathcal{C}$ satisfies any of the two first conditions, then we say that it is a *successful computation.*

Definition 6. *An accepting P system is said to be* valid *if for every halting computation, and only for them, one copy of object Yes or one copy of object No (but not copies of both) is sent out (and this happens in the last step of the computation).*

Definition 7. *We say that $\mathcal{C}$ is an* accepting *computation (respectively,* rejecting *computation) if the object Yes (respectively, No) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$, that is, if $Yes = Output(\mathcal{C})$ (respectively, $No = Output(\mathcal{C})$).*

Definition 8. *A* recognizer P system *is a valid accepting P system such that all its computations halt.*

Such recognizer systems are specially suitable when trying to solve decision problems.

3 The Complexity Class $\mathbf{PMC}_{\mathcal{AM}}$

Roughly speaking, a computational complexity study of a solution for a problem is an estimation of the resources (time, space, ...) that are required through all the processes that take place in the way from the bare instance of the problem up to the final answer.

The first results about "solvability" of **NP**–complete problems in polynomial time (even linear) by cellular computing systems with membranes were obtained using variants 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.

If we would implement such a solution of some decision problem in a laboratory, then we have to cope with the diffficulty that a system constructed to solve a concrete instance is useless when trying to solve another instance. This drawback can be easily overtaken if we consider a P system with input. Then, the same system could solve different instances of the problem, provided that the corresponding input multisets are introduced in the input membrane.

Instead of looking for a single system that solves a problem, we prefer designing a family of P systems such that each element decides all the instances of "equivalent size", in certain sense.

Let us now introduce some basic concepts before the definition of the complexity class itself.

Let us denote by $\mathcal{AM}$ the class of language recognizer P systems with active membranes using 2-division (see [3], section 7.2).

Definition 9. *Let L be a language and $\mathbf{\Pi} = (\Pi(t))_{t \in \mathbb{N}}$ a family of P systems with active membranes using 2-division. A* polynomial encoding *of L in $\mathbf{\Pi}$ is a pair (cod, s) of polynomial-time computable functions, $cod : L \to \bigcup_{t \in \mathbb{N}} I_{\Pi(t)}$, and $s : L \to \mathbb{N}$ such that for every $u \in L$ we have $cod(u) \in I_{\Pi(s(u))}$.*

That is, for each word u of the language L, we have a multiset $cod(u)$ and a number $s(u)$ associated with it such that $cod(u)$ is input multiset for the P system $\Pi(s(u))$.

Lemma 1. *Let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be languages. Let $\mathbf{\Pi} = (\Pi(t))_{t \in \mathbb{N}}$ a family of P systems with active membranes using 2-division. If $r : \Sigma_1^* \to \Sigma_2^*$ is a polynomial-time reduction from L_1 to L_2, and (cod, s) is a polynomial encoding of L_2 in $\mathbf{\Pi}$, then $(g \circ r, h \circ r)$ is a polynomial encoding of L_1 in $\mathbf{\Pi}$.*

Proof. This result follows directly from the previous definition. For a detailed proof, we refer the reader to [7]. □

Considering all the definitions already presented, we are now ready to give the definition of the complexity class $\mathbf{PMC}_{\mathcal{AM}}$, which is based on the one given in [7].

Definition 10. *We will say that a decision problem, $X = (I_X, \theta_X)$, is* solvable *in polynomial time by a family of recognizer P systems with active membranes using 2-division, and we denote this by $X \in \mathbf{PMC}_{\mathcal{AM}}$, if there exists a family of P systems, $\mathbf{\Pi} = \big(\Pi(t)\big)_{t \in \mathbb{N}}$, with the following properties:*

1. *The family $\mathbf{\Pi}$ is consistent with regard to the class $\mathcal{AM}$; that is, $\forall t \in \mathbb{N}$ $(\Pi(t) \in \mathcal{AM})$.*
2. *The family $\mathbf{\Pi}$ is polynomially uniform, by Turing machines; that is, there exists a deterministic Turing machine constructing $\Pi(t)$ from t in polynomial time.*
3. *There exist two functions, $cod : I_X \to \bigcup_{t \in \mathbb{N}} I_{\Pi(t)}$ and $s : I_X \to \mathbb{N}^+$, computable in polynomial time, such that:*
   - *For every $u \in I_X$, $cod(u) \in I_{\Pi(s(u))}$.*
   - *The family $\mathbf{\Pi}$ is polynomially bounded with regard to (X, cod, s); that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of the system $\Pi(s(u))$ with input $cod(u)$ is halting and, moreover, it performs at most $p(|u|)$ steps.*

- *The family* $\mathbf{\Pi}$ *is sound with regard to* (X, cod, s)*; that is, for each* $u \in I_X$ *it is verified that if there exists an accepting computation of the system* $\Pi(s(u))$ *with input* $cod(u)$*, then* $\theta_X(u) = 1$.
- *The family* $\mathbf{\Pi}$ *is complete with regard to* (X, cod, s)*; that is, for each* $u \in I_X$ *it is verified that if* $\theta_X(u) = 1$*, then every computation of the system* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting one.*

Remark 2. Note that, as a consequence of the above definition, the complexity class $\mathbf{PMC}_{\mathcal{AM}}$ is closed under complement, because recognizer P systems are being used.

Proposition 1. *Let* X *and* Y *be decision problems such that* X *is reducible to* Y *in polynomial time. If* $Y \in \mathbf{PMC}_{\mathcal{AM}}$*, then* $X \in \mathbf{PMC}_{\mathcal{AM}}$.

That is, the complexity class $\mathbf{PMC}_{\mathcal{AM}}$ is stable under polynomial-time reduction. The proof of this result can also be found in [7].

4 Solving the Decision Knapsack Problem in Linear Time

The decision *Knapsack* problem (0/1) can be stated as follows:

> *Given a knapsack of capacity* $k \in \mathbb{N}$*, a set* A *of* n *elements, where each element has a "weight"* $w_i \in \mathbb{N}$ *and a "value"* $v_i \in \mathbb{N}$*, and given a constant* $c \in \mathbb{N}$*, 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*.*

We will represent the instances of the problem using 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 values, respectively, of the elements from A; moreover, k and c are the constants above mentioned. We can define in a natural way additive functions w and v that correspond to the data in the instance.

We address the resolution of the problem via a brute force algorithm, in the framework of recognizer P systems with active membranes using 2-division, without cooperation nor dissolution nor priority among rules. Our strategy will consist in:

- *Generation stage*: membrane division is used until a specific membrane for each subset of A is obtained.
- *Calculation stage*: in each membrane the weight and the value of the associated subset are calculated.
- *Checking stage for the "weight" function* w: the condition $w(B) \leq k$ is checked for every subset $B \subseteq A$.

– *Checking stage for the "value" function* v: the condition $v(B) \geq c$ is checked for every subset $B \subseteq A$.
– *Output stage*: the answer is delivered according to the results of both checking stages.

Let us consider a polynomial bijection, $\langle\ \rangle$, between $\mathbb{N}^3$ and $\mathbb{N}$ (e.g., $\langle x, y, z\rangle = \langle\langle x, y\rangle, z\rangle$, induced by the pair function $\langle x, y\rangle = (x + y) \cdot (x + y + 1)/2 + x$).

The family presented here is

$$\mathbf{\Pi} = \{(\Pi(\langle n, k, c\rangle), \Sigma(n, k, c), i(n, k, c)) \mid (n, k, c) \in \mathbb{N}^3\}.$$

For each element of the family, the input alphabet is $\Sigma(n, k, c) = \{x_1, \ldots, x_n, y_1, \ldots, y_n\}$, the input membrane is $i(n, k, c) = e$, and the P system $\Pi(\langle n, k, c\rangle) = (\Gamma(n, k, c), \{e, s\}, \mu, \mathcal{M}_s, \mathcal{M}_e, R)$ is defined as follows:

- Working alphabet:

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

- Membrane structure: $\mu = [_s\ [_e\ ]_e\ ]_s$.
- Initial multisets: $\mathcal{M}_s = z_0$; $\mathcal{M}_e = e_0\, \bar{a}^k\, \bar{b}^c$.
- The set of evolution rules, R, consists of the following rules:

(a) $[_e e_i]_e^0 \to [_e q]_e^- [_e e_i]_e^+$, for $i = 0, \ldots, n$.
$[_e e_i]_e^+ \to [_e e_{i+1}]_e^0 [_e e_{i+1}]_e^+$, for $i = 0, \ldots, n-1$.

The goal of these rules is to generate one membrane for each subset of A. When an object e_i ($i < n$) is present in a neutrally charged membrane, we pick the element a_i for its associated subset and divide the membrane. In one of the new membranes the object q appears, and this means that this membrane leaves the generation stage, no further elements will be added to the subset. But the other new membrane must generate membranes for other possible subsets which are obtained by adding elements of index $i + 1$ or greater.

(b) $[_e x_0 \to \bar{a}_0]_e^0$; $[_e x_0 \to \epsilon]_e^+$; $[_e x_i \to x_{i-1}]_e^+$, for $i = 1, \ldots, n$.
$[_e y_0 \to \bar{b}_0]_e^0$; $[_e y_0 \to \epsilon]_e^+$; $[_e y_i \to y_{i-1}]_e^+$, for $i = 1, \ldots, n$.

In the beginning, the multiplicity of the object x_j and y_j (with $1 \leq j \leq n$) encodes the weights and the values, respectively, of the corresponding elements of A. They are not present in the definition of the system, but they are inserted as input in the membrane labelled by e before starting the computation: for each $a_j \in A$, w_j copies of x_j and v_j copies of y_j have to be added to the input membrane. During the computation, at the same time as elements are added to the subset associated with a membrane, objects $\bar{a}_0$ and $\bar{b}_0$ are generated to store the weight and the value, respectively, of such subset.

(c) $[_e q \to \bar{q} q_0]_e^-$; $[_e \bar{a}_0 \to a_0]_e^-$; $[_e \bar{a} \to a]_e^-$.
$[_e \bar{b}_0 \to \hat{b}_0]_e^-$; $[_e \bar{b} \to \hat{b}]_e^-$.

When a membrane gets negatively charged, the two first stages (i.e., generation and calculation stages) end, and then some transition rules are applied. Objects $\bar{a}$ and $\bar{b}$, whose multiplicities encode in the initial multiset the constants k and c, respectively, as well as objects $\bar{a}_0$ and $\bar{b}_0$, are renamed for the next stage, when the multiplicities of a and a_0 are compared. Also an object q_0 appears, that will act as a counter, and objects $\bar{q}$, $\hat{b}_0$ and $\hat{b}$ appear as well, although they will remain inactive all through this stage. If the checking had an affirmative result, then the membrane would get positively charged. In this case this stage ends and objects $\bar{q}$, $\hat{b}_0$ and $\hat{b}$ are activated for the next one.

(d) $[_e a_0]_e^- \to [_e]_e^0 \#; \quad [_e a]_e^0 \to [_e]_e^- \#.$

These rules implement the comparison above mentioned (that is, they check whether $w(B) \leq k$ holds or not). They work as a loop that erases objects a_0 and a one by one alternatively, changing the charge of the membrane in each step.

(e) $[_e q_{2j} \to q_{2j+1}]_e^-$, for $j = 0, \ldots, k$.
$[_e q_{2j+1} \to q_{2j+2}]_e^0$, for $j = 0, \ldots, k-1$.

A counter that controls the previous loop is described here. The index of q_i and the electric charge of the membrane give enough information to point out if the number of objects a_0 is greater than (less than or equal to) the number of objects a.

(f) $[_e q_{2j+1}]_e^- \to [_e]_e^+ \#$, for $j = 0, \ldots, k$.

If a subset $B \subseteq A$ verifies the condition $w(B) \leq k$, then inside the relevant membrane that encodes it (this will be defined later) there will be less objects a_0 than a. This forces the loop described in (e) to halt: the moment will come when there are no objects a_0 left, and then the rule $[_e q_{2w(B)} \to q_{2w(B)+1}]_e^-$ will be applied but it will not be possible to apply the rule $[_e a_0]_e^- \to [_e]_e^0 \#$ at the same time. Thus, an object $q_{2w(B)+1}$ will be present in the membrane and the latter will be negatively charged, so the rule (f) will be applied allowing the checking stage for v to start.

(g) $[_e \bar{q} \to \bar{q}_0]_e^+; \quad [_e \hat{b}_0 \to b_0]_e^+; \quad [_e \hat{b} \to b]_e^+; \quad [_e a \to \epsilon]_e^+.$

Just as the rules from (c) did, these rules perform a renaming task before the checking loop for v begins.

(h) $[_e b_0]_e^+ \to [_e]_e^0 \#; \quad [_e b]_e^0 \to [_e]_e^+ \#.$

The checking loop for v is designed exactly as the one for w, but the electric charges involved are now neutral and positive.

(i) $[_e \bar{q}_{2j} \to \bar{q}_{2j+1}]_e^+$ for $j = 0, \ldots, c$.
$[_e \bar{q}_{2j+1} \to \bar{q}_{2j+2}]_e^0$ for $j = 0, \ldots, c-1$.

The counter q_i, described in (e), is now replaced by a slightly different one, $\bar{q}_i$. Both of them behave similarly in the first steps of the checking, but the way the fourth stage (checking for v) ends is different from the way the third stage does, because in the case of the checking with respect to v the successful end will be reached when the multiplicity of b_0 is greater than the multiplicity of b.

(j) $[_e\bar{q}_{2c+1}]_e^+ \to [_e]_e^0 Yes$; $[_e\bar{q}_{2c+1}]_e^0 \to [_e]_e^0 Yes$.

As pointed before, if a subset $B \subseteq A$ verifies the condition $v(B) \geq c$, then in the membrane that corresponds to that subset there will be more objects b_0 than b. This causes the rules from (h) to apply c times each, and after that the index of $\bar{q}_i$ will be $i = 2c$. Then the rule $[_e\bar{q}_{2c} \to \bar{q}_{2c+1}]_e^+$ will be applied (possibly together with $[_e b_0]_e^+ \to [_e]_e^0 \#$), and finally one of the rules from (j) will terminate the stage.

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

Before the answer is sent out, all the membranes should have either ended their two checking stages successfully or got to a blocking state otherwise. The counter z_j gives enough room to deal even with the worst case. The condition for the worst case (i.e., the case where the checking stages will last longer) is that the total set is a solution (i.e., it satisfies the two conditions). The number of steps will be maximum if the weight of A is exactly k.

Indeed, the membrane associated with the total set is the latter one to be generated, and it can avoid none of the steps of both checking stages.

(l) $[_s d_+]_s^0 \to [_s]_s^+ d_+$; $[_s d_- \to No]_s^+$; $[_s Yes]_s^+ \to [_s]_s^0 Yes$; $[_s No]_s^+ \to [_s]_s^0 No$.

Finally, the output process is activated. The skin membrane needs to be positively charged before the answer is sent out. Object d_+ takes care of this and then, if the answer is affirmative, an object Yes will be sent out recovering the neutral charge for the skin.

5 An Overview of the Computation

First of all we must define a polynomial encoding of the *Knapsack* problem in the family $\mathbf{\Pi}$ in order to study the complexity of the problem with respect to it. Given an instance $u = (n, (w_1, \dots, w_n), (v_1, \dots v_n), k, c)$ of the *Knapsack* problem, we define $s(u) = \langle n, k, c \rangle$ (recall the bijection mentioned in the previous section) and $cod(u) = x_1^{w_1} \dots x_n^{w_n} y_1^{v_1} \dots y_n^{v_n}$. Now we will informally describe how does the system $\Pi(s(u))$ with input $cod(u)$ work.

In the first step of the computation, the rule $[_e e_i]_e^0 \to [_e q]_e^- [_e e_i]_e^+$ is applied. Then the generation and calculation stages go on in parallel, following the instructions from the rules in (a) and (b). This two stages do not end in a membrane as long as an object e_j (with $0 \leq j \leq n$) belongs to it and its charge is positive or neutral. We intend to get a single working membrane for each subset of A but, as we will see later on, they are not generated altogether simultaneously (it can be checked that the membrane corresponding to the subset $\{a_{i_1}, \dots, a_{i_r}\}$ will arise in the $(i_r + r + 2)$-th step).

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

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

As we already said, the two first stages are carried out in parallel. Indeed, there is only a gap of one step of computation between the moment when an element is added to the associated subset and the moment when the new weight and value of the subset are updated. For example, for a_1, after two steps of computation we can see that there are three inner membranes in the configuration, and one of them is a neutrally charged membrane where the object e_1 occurs (see Fig. 1). Thus, the element a_1 is added to the associated subset. It can be proved that there are in that moment w_1 copies of x_0 and v_1 copies of y_0 in the membrane. So, in the next step, at the same time as the membrane divides, w_1 objects $\bar{a}_0$ and v_1 objects $\bar{b}_0$ will be generated in the membrane by the rules in (b). These rules will also modify the indices of all the objects x_i and y_i, with $i > 0$, so that w_2 copies of x_0 and v_2 copies of y_0 will be ready in the membrane in the next step.

After a division rule $[_e e_i]_e^0 \rightarrow [_e q]_e^- [_e e_i]_e^+$ is applied, the two new membranes will behave in a quite different way. On one hand, in the negatively charged membrane (we have marked such membranes in Fig. 1 with a circle) the two first stages end, and in the next step the rules in (c) will be applied, renaming the objects to prepare the third stage. This is a significant moment, so we will call *relevant* those membranes that have a negative charge and contain an object q_0. A relevant membrane will not divide anymore during the computation, and its associated subset will remain unchanged.

On the other hand, the positively charged membrane will continue the generation stage and it will give rise to membranes associated with subsets that are obtained adding to the current one elements of index $i+1$ or greater. Note that if $i = n$, then the membrane can not continue the generation stage, as there are no rules working for an object e_n in a positively charged membrane (see the membranes surrounded by a diamond in Fig. 1). It makes sense that these membranes get blocked, as it is not possible to add elements of indices greater than n.

Thus, as the indices of objects e_i never decrease, we notice that the relevant membranes are generated in a kind of "lexicographic order", in the following sense: if the j-th element of A has already been added to the associated subset, then no element with index lower than j will be added later on to the subset associated with that membrane nor to the subsets associated with its descendants.

The purpose of the rules in (d) is comparing the multiplicities of objects a_0 and a (that is, to perform the *checking stage for* w). Note that to prevent the

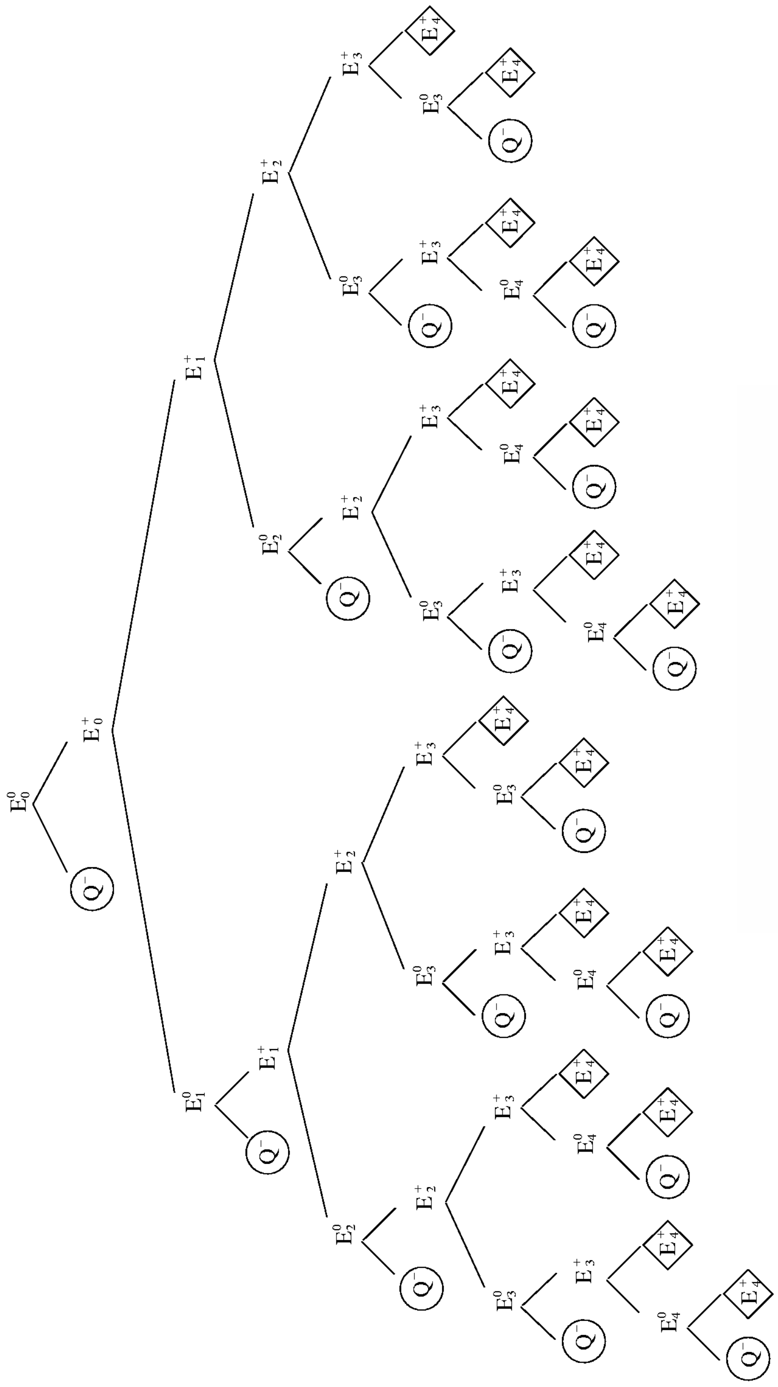

Fig. 1. Membrane generation for $n = 4$

checking rules in (h) from disturbing this stage, the objects $\bar{b}$ and $\bar{b}_0$ are renamed as $\hat{b}$ and $\hat{b}_0$, respectively, and will remain inactive all over this stage. Recall that the counter q_i is the one that controls the result of the checking. Let us briefly explain how the comparison loop works.

Let B be a subset of a certain weight w_B and a certain value v_B. Then the evolution of the relevant membrane associated with it along the third stage is described in Table 1:

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
$\vdots$	$\vdots$	$\vdots$

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

Note 2. If $B = \{a_{i_1}, \dots, a_{i_r}\}$ with $i_r \neq n$, then some objects x_j and y_j, for $1 \leq j \leq n - i_r$, will be in the multiset, 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 to the next checking (rules in (f) take care of preparing with an appropriate renaming the objects that will be compared). This process is described in Table 2.

If the number of objects a_0 is greater than k, then every time that the rule $[_e q_{2j} \rightarrow q_{2j+1}]_e^-$ is applied (that is, for $j = 0, \dots, k$), the rule $[_e a_0]_e^- \rightarrow [_e]_e^0 \#$ 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 is negative. That means that the rule (f) can never be applied, and even more, the membrane gets *blocked* (it will not evolve anymore during the computation). See Table 3 for details.

Let us suppose that the third stage has successfully finished in a membrane. That means that this membrane encodes a subset $B \subseteq A$ such that $w(B) \leq k$. Then, after applying the rule (f), this membrane gets a positive charge. Subsequently, the rules in (g) are applied. That is, a transition step is performed,

Table 2.

Multiset	Charge	Parity of q_i	rules
$q_{2w_B-1}\ a^{k-w_B+1}\ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	0	ODD	d,e
$q_{2w_B}\ a^{k-w_B}\ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	–	EVEN	e
$q_{2w_B+1}\ a^{k-w_B}\ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	–	ODD	f
$a^{k-w_B}\ \bar{q}\hat{b}_0^{v_B}\hat{b}^c$	+	ODD	g

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

renaming the objects $\hat{b}_0$ and $\hat{b}$ as b_0 and b, respectively, in order to allow the comparison of their multiplicities. The counter $\bar{q}_i$ is initialized to $\bar{q}_0$ and starts working to control the result of the comparison.

The fourth stage works in fact in a very similar way as the third one. Rules in (h) and (i) correspond to rules in (d) and (e), respectively, but the end of the stage is different (see Table 5). The difference lies in the rules in (j). In this stage, the checking is successful if the number of objects b_0 is greater or equal than the number of objects b (i.e., if $v(B) \geq c$) and so, to enter the next stage the two rules in (h) must have been applied c times each (consequently, rules in (i) have been applied the same number of times). This condition is guaranteed by the head of the rule (j). See Table 4 for details.

Finally, rules in (k) and (l) are associated with the skin membrane and take care of the output stage. The counter z_i, described in (k), waits for $2n+2k+2c+7$ steps ($2n+2$ steps for the first and the second stage, one transition step, $2k+1$ steps for the third one, another transition step, and $2c+2$ for the fourth one). 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.

In that moment, 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 inner membrane.

Then the output stage begins. First of all the object d_+ is sent out, giving 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 skin, it will be

Table 4.

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

sent out of the system, giving neutral charge to the skin and making the system halt (in particular, the further evolution of object No is avoided).

If it is the case that none of the membranes has successfully passed both checking stages, then there will not be any object Yes present in the skin at the time when the output stage begins. Thus, after the object No is generated, the skin will still have a positive charge, so it will be sent out. In that moment the system halts.

6 Formal Verification

In this section we prove that the family $\mathbf{\Pi}$ of P systems with active membranes designed in Section 4 provides a polynomial solution for the *Knapsack* problem, in the sense given in Definition 10; that is, $Knapsack \in \mathbf{PMC}_{\mathcal{AM}}$.

First of all, it is obvious that the family of P systems defined is $\mathcal{AM}$-consistent, because every P system of the family is a recognizer P system with active membranes using 2-division.

In second place, the family is polynomially uniform by Turing machines. It can be observed that the definition of the family is done in a recursive manner from a given instance, in particular from the constants n, k and c. Furthermore, the necessary resources to build an element of the family are of a linear order with respect to these constants:

– Size of the alphabet: $5n + 4k + 4c + 28 \in O(n + k + c)$,
– number of membranes: $2 \in \Theta(1)$,
– $|\mathcal{M}_e| + |\mathcal{M}_e| = k + c + 2 \in O(k + c)$,
– sum of the rules' lengths: $40n + 27k + 20c + 193 \in O(n + k + c)$.

Before going on, let us recall the functions $cod : I_{Knp} \to \bigcup_{t\in\mathbb{N}} I_{\Pi(t)}$ and $s : I_{Knp} \to \mathbb{N}$ defined for an instance $u = (n, (w_1, \ldots, w_n), (v_1, \ldots v_n), k, c)$ as $cod(u) = x_1^{w_1} \ldots x_n^{w_n} y_1^{v_1} \ldots y_n^{v_n}$, and $s(u) = \langle n, k, c\rangle$, respectively.

Both functions are total and polynomially computable. Furthermore, (cod, s) provides a polynomial encoding of the set of instances of the *Knapsack* problem for the family $\mathbf{\Pi}$ of P systems, as we have that, for any instance u, $cod(u)$ is a valid input for the P system $\Pi(s(u))$.

Please note that the instance $u = (n, (w_1, \ldots, w_n), (v_1, \ldots v_n), k, c)$ is introduced in the initial configuration through an input multiset; that is, encoded in an unary representation and, thus, we have that $|u| \in O(w_1 + \cdots + w_n + v_1 + \cdots + v_n + k + c)$.

To establish the verification we need to prove that every P system of the family is polynomially bounded and it is sound and complete with respect to $(Knapsack, cod, s)$.

6.1 Linearly Bounded

To check that $\Pi(s(u))$ with input $cod(u)$ is polynomially (in fact, linearly) bounded, it is enough to identify the step when the computation stops, or at least an upper bound. Actually, as we are going to see, the number of steps of the computations of the P systems of the family can be always bounded by a *linear* function. However, let us observe that the amount of pre-computed resources for each instance u is polynomial in the size of the instance, because $cod(u)$, $s(u)$ need to be calculated and $\Pi(s(u))$ need to be constructed.

First comes the *generation stage*. The rules in (a) are designed in such a way that a necessary and sufficient condition for a membrane to divide is that either it has neutral charge and it contains an object e_j, with $j \in \{0, 1, \ldots, n\}$, or it has positive charge and it contains an object e_j, with $j \in \{0, 1, \ldots, n-1\}$. In the first case the result of the division is a membrane with negative charge, where e_j is replaced by q, and a membrane with positive charge where the object e_j remains (if $j < n$, then the latter membrane is in the second case, and thus it will divide again). In the second case, one of the membranes obtained has neutral charge and the other one positive, and in both of them the object e_j is replaced by e_{j+1}. As the index set is finite, the divisions cannot go on indefinitely, thus the generation stage is a finite process. Furthermore, keeping in mind the fact that after two consecutive divisions the index of the object e_j in the membrane grows in one unit (if it has not been replaced by q), it can be deduced that after $2n + 1$ steps no more membrane divisions will take place.

For the *calculation stage* we follow a similar reasoning. It suffices to consider the objects x_i and y_i and to note how their subscripts decrease in each step (if $i > 0$) or how they are removed or renamed (if $i = 0$). Thus also this stage has a linear number of steps.

The third stage (*checking stage for w*) consists basically in a loop made from the pair of rules $[_e a_0]_e^- \to [_e]_e^0 \#$ and $[_e a]_e^0 \to [_e]_e^- \#$, hence in every lap of the loop two objects are removed. In parallel with the application of the previous two rules, the counter q_i evolves, and it causes this stage to finish in at most $2k + 1$ steps. If the checking is successful (that is, if $w_B \leq k$, where B is the associated subset), then the stage will spend $2w_B + 1$ steps, as we saw in the previous section; but if the checking is not successful (that is, if $w_B > k$), then the membrane will get blocked after $2k+1$ steps and will not proceed to the next stage.

In the case that a membrane successfully passes the third stage, in the next step the rules in (g) will be applied in such membrane, preparing the objects b and b_0 to compare their multiplicities and initializing the counter $\bar{q}_0$ that controls the result. As for the function w, the *checking stage for* v consists basically in the reiteration of a loop, but in this case the number of steps will be $2c+2$ for the affirmative case (that is, if $v(B) \geq c$) and in the negative case the process will get blocked after $2v_B + 2$ steps (if $v(B) < c$).

We can then conclude that in the inner membranes no rule is applied after $2n+2k+2c+7$ steps.

There is only the *output stage* left. The mechanism is simple: there is a counter z_j in the skin that evolves from z_0 to $z_{2n+2k+2c+6}$, this needs $2n+2k+2c+6$ steps in all. Then, $z_{2n+2k+2c+6}$ evolves producing two objects, d_+ and d_-, and after that the rule $[_s d_+]_s^0 \rightarrow [_s]_s^+ d_+$ is applied, changing the polarity of the skin to positive. In this moment, if the answer is affirmative, then the object d_- evolves to No inside the skin, and simultaneously an object Yes is sent out, so the system halts in the $(2n+2k+2c+9)$-th step. If, on the contrary, the answer is negative, then the object d_- evolves to No inside the skin but no object Yes is sent out simultaneously, so the skin still has a positive charge in the next step and consequently the rule $[_s No]_s^+ \rightarrow [_s]_s^0 No$ is applied, and the system halts in the $(2n+2k+2c+10)$-th step. Recall that $|u| \in O(w_1+\cdots+w_n+v_1+\cdots+v_n+k+c)$.

6.2 Soundess and Completeness

Now the soundness and completeness of the family of P systems $\mathbf{\Pi}$ with regard to $(Knapsack, cod, s)$ will be established. That is, it will be proved that given an instance u of the problem, the P system of the family associated with such instance, $\Pi(s(u))$, with input $cod(u)$, answers Yes if and only if the problem has an affirmative answer for the given instance.

The proof will be made studying the correct functioning of each stage separately. For the generation stage, let us consider a subset $B = \{a_{i_1}, \ldots, a_{i_r}\} \subseteq A$ (with $i_1 < i_2 < \cdots < i_r$, and $r \leq n$); then there must exist a single relevant membrane that encodes it. Actually, the sequence of membranes that leads to the latter is:

$$[_e e_0]_e^0 \Rightarrow [_e e_0]_e^+ \Rightarrow [_e e_1]_e^+ \Rightarrow \cdots \Rightarrow [_e e_{i_1-1}]_e^+ \Rightarrow [_e e_{i_1}]_e^0 \Rightarrow [_e e_{i_1}]_e^+ \Rightarrow \cdots \Rightarrow$$
$$\Rightarrow [_e e_{i_2-1}]_e^+ \Rightarrow [_e e_{i_2}]_e^0 \Rightarrow \cdots \Rightarrow [_e e_{i_r}]_e^0 \Rightarrow [_e q]_e^- \Rightarrow [_e q_0]_e^- .$$

Due to its recursive definition, the associated subset keeps in some sense track of the *history* of the membrane. Considering this, it is easy to see that the sequence sketched above leads actually to a uniquely determined relevant membrane with B as associated subset.

Next, to be sure that the second stage also works correctly, we will see that the multiplicities of the objects a_0 and b_0 in the membrane previously referred are $w(B)$ and $v(B)$, respectively. Note that to add the weights of the elements of A immediately after they are selected for the associated subset does not cause any trouble, because if an element a_j belongs to the subset encoded by a certain membrane, then it will also belong to the subsets encoded by its descendants and, thus, we can already add the corresponding weight.

Lemma 2. *In a positively charged inner membrane where the object e_i appears, for $0 \leq i \leq n-1$, the multiplicities of the objects x_j and y_j correspond exactly with the weight $w(a_{i+j})$ and with the value $v(a_{i+j})$ of the element $a_{i+j} \in A$, respectively, for every j such that $1 \leq j \leq n-i$.*

Proof. By induction on i.

For the base case, $i = 0$, we know that the multiset associated with the membrane e in the initial configuration is $e_0 \bar{a}^k \bar{b}^c x_1^{w_1} \cdots x_n^{w_n} y_1^{v_1} \cdots y_n^{v_n}$. Then the rule $[_e e_0]_e^0 \to [_e q]_e^- [_e e_0]_e^+$ will be applied (actually, it is the only rule that can be applied in this situation), and the multiset of the positively charged membrane will remain unchanged. Thus, for every j such that $1 \leq j \leq n-i$ the multiplicities of the objects x_j and y_j correspond exactly with the weight and the value of the element $a_{i+j} \in A$, as we wanted to prove.

For the inductive step, let us suppose that the result holds for $i < n-1$. Let us consider then a positively charged inner membrane where the object e_{i+1} appears. We will distinguish two cases:

Let us suppose first that the membrane has been obtained through the rule $[_e e_i]_e^+ \to [_e e_{i+1}]_e^0 [_e e_{i+1}]_e^+$. Then, from the inductive hypothesis we deduce that the multiplicities of the objects x_j and y_j in the original membrane correspond exactly with the weight and the value of the element $a_{i+j} \in A$, for every j such that $1 \leq j \leq n-i$. Note that also the rules in (b) (the ones for positively charged membranes) have been applied, and thus the multiplicities of the objects x_j and y_j in the membrane are equal to the multiplicities of the objects x_{j+1} and y_{j+1} in the father membrane, for $0 \leq j \leq n-i-1$. Then, we come to the conclusion that the multiplicities of the objects x_j and y_j correspond exactly with the weight and the value of the element $a_{i+j+1} \in A$ for $1 \leq j \leq n-(i+1)$. This completes the proof.

For the second case let us suppose that the membrane is obtained as the result of applying the rule $[_e e_{i+1}]_e^0 \to [_e q]_e^- [_e e_{i+1}]_e^+$. Following a reasoning similar to the previous one for the father membrane (as $[_e e_{i+1}]_e^0$ is obtained in its turn through the rule $[_e e_i]_e^+ \to [_e e_{i+1}]_e^0 [_e e_{i+1}]_e^+$), we come to the conclusion that the multiplicities of the objects x_j and y_j in the father membrane correspond exactly with the weight and the value of the element $a_{i+j} \in A$, for every j such that $1 \leq j \leq n-i$. Furthermore, the multiplicities of such objects do not change in the next step, because the only rule that is applied simultaneously to the division rule is $[_e x_0 \to \bar{a}_0]_e^0$, and this one cannot be applied to objects x_j and y_j with $j \geq 1$. □

Proposition 2. *In a relevant membrane the number of copies of the objects a_0 and $\hat{b}_0$ match exactly the weight and the value of the associated subset, respectively.*

Proof. Given an arbitrary relevant membrane, let $B = \{a_{i_1}, \ldots, a_{i_r}\}$ be its associated subset, with $i_1 < i_2 < \cdots < i_r$, and $r \leq n$. Then, as we said before, there is no other relevant membrane along the computation associated with the same subset.

In the sequence that leads to the relevant membrane there are $r + i_r + 1$ intermediate membranes between the initial one and the relevant one. Among them there is one that has a neutral charge and contains an object e_{i_l} for each $a_{i_l} \in B$. In addition, for each j with $0 \leq j \leq i_r - 1$ there is one membrane positively charged that contains the object e_j. Finally, there is also a negatively charged membrane that contains the object q.

If we apply Lemma 2 choosing $i = i_l - 1$ (with $1 \leq l \leq r$) and $j = 1$ to all the membranes that are positively charged and contain an object $e_{i_l - 1}$, we get that in each one of them the multiplicities of the objects x_1 and y_1 are exactly w_{i_l} and v_{i_l} (the weight and the value of $a_{i_l} \in A$, respectively). Since in every one of them the rules in (b) will be applied in the next step, we deduce that, for $1 \leq l \leq r$, in the neutrally charged membrane where e_{i_l} appears there will be w_{i_l} objects x_0 and v_{i_l} objects y_0. Consequently in the next step w_{i_l} copies of $\bar{a}_0$ and v_{i_l} copies of $\bar{b}_0$ will be generated in the membrane.

This holds for every i_l with $1 \leq l \leq r$, that is, for all the elements in B, so in the negatively charged membrane where q appears, the multiplicities of the objects $\bar{a}_0$ and $\bar{b}_0$ are the sum of the weights and the sum of the values of the elements in B, respectively. That is, they are the weight and the value of the subset B. Finally, after applying the rules in (c) in the last step of the sequence, we come to the desired result. □

Next the checking stages are studied. Let us consider a relevant membrane whose associated subset is $B = \{a_{i_1}, \ldots, a_{i_r}\}$, and let $w(B) = w_B$ and $v(B) = v_B$ be the weight and the value, respectively, of such subset. Then the multiset associated with the membrane is $q_0 \bar{q} a_0^{w_B} a^k \hat{b}_0^{v_B} \hat{b}^c x_1^{w_{i_r+1}} \cdots x_{n-i_r}^{w_n} y_1^{v_{i_r+1}} \cdots y_{n-i_r}^{v_n}$.

The point is to compare the number of objects a with the number of objects a_0 present in the membrane. The object q_0 will evolve in parallel with the comparison and it controls the end of the stage. The objects $\hat{b}_0$, $\hat{b}$, and $\bar{q}$ remain inactive during this stage, as we already said in the previous section. If any objects x_j or y_j, with $j > 0$, appear inside the membrane, then they do not evolve in this stage, as the membrane will not get positive charge until the end of the stage, and only if the checking is successful.

There are two possibilities: either $w_B \leq k$ or $w_B > k$. In the previous section we discussed what happens in each case. A membrane passes successfully the third stage if and only if $w_B \leq k$.

As soon as the third stage ends in a membrane (and it ends successfully) the membrane gets a positive charge, and thus the objects $\hat{b}_0$, $\hat{b}$, and $\bar{q}$ are reactivated (through the rules in (g)). In addition, objects x_j and y_j possibly present in the membrane will also evolve through the rules in (b), but we are not going to pay attention to them because they do not affect the membrane's charge nor the result of the v-checking. The loop formed by the two rules in (h) works just the same as the loop from the third stage, its goal is to compare the multiplicities of the objects b and b_0, and it does this eliminating such objects one by one (sending them out of the membrane) and changing the charge every step. As it was studied in the previous section, in this stage the counter $\bar{q}_i$ works in a different way as q_i did, because in this stage a successful result is obtained when the number of objects b_0 is greater than the number of b's.

An object Yes will be sent out to the skin at the end of this stage if and only if the counter $\bar{q}_i$ does not get blocked before arriving to $\bar{q}_{2c+1}$. That is, if and only if the membrane changes its charge every step during $2c$ steps. This is equivalent to the condition that one rule from (h) must be applied each one of the $2c$ first steps of this stage, and in its turn this is equivalent to asking that there are at least c copies of b and of b_0 as well. We already know that in the beginning of the stage there are c copies of b in the membrane, so what we are requiring is the multiplicity of b_0 to be greater or equal than c, that is, the value of the subset is greater or equal than c. Thus, the checking stage for v is well designed.

Until now, we have proved that for every subset $B \subseteq A$, a relevant membrane will appear sooner or later along the computation with B as associated subset and with exactly w_B copies of a_0 and v_B copies of $\hat{b}_0$ inside it. We have also proved that a relevant membrane will pass the third and the fourth stage (sending thus an object Yes to the skin) if and only if $w_B \leq k$ and $v_B \geq c$ hold. Finally, we must only see that the output stage works correctly too.

This is easily done if we keep in mind that in the moment when objects d_+ and d_- appear it is granted that all membranes have had enough time to achieve their two checking stages. That is, either they have successfully passed them and have sent an object Yes to the skin or they have got blocked. It suffices to note that the object Yes has some "priority" over the object No to go out.

Indeed, when the object d_+ leaves the system and gives a positive charge to the skin, the object No has not been generated yet. So if the instance has an affirmative solution, then some objects Yes (at least one) will already be present in the skin and thus in the following step one of these objects Yes will be sent out, while the object No is generated inside the skin. In the case of a negative solution, none of the membranes will sent an object Yes to the skin and consequently two steps after the moment when d_+ is sent out, an object No will also leave the system and the computation will halt.

7 Main Results

From the discussion in the previous section, and according to Definition 10, we deduce the following result:

Theorem 1. *Knapsack* $\in \mathbf{PMC}_{\mathcal{AM}}$.

Although the next result is a corollary of Theorem 1, we formulate it as another theorem, in order to stress its relevance.

Theorem 2. $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$.

Proof. It suffices to make the following observations: the *Knapsack* problem is **NP**–complete, $Knapsack \in \mathbf{PMC}_{\mathcal{AM}}$ and the class $\mathbf{PMC}_{\mathcal{AM}}$ is closed under polynomial-time reduction. □

This theorem can be extended, if we notice that the class $\mathbf{PMC}_{\mathcal{AM}}$ is closed under complement.

Theorem 3. $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$.

8 Conclusions

In this paper we have presented a family of recognizer P systems solving the Knapsack problem in *linear time*. This has been done in the framework of complexity classes in cellular computing with membranes.

We would like to clarify what does linear time mean. The linear bound is referred to the number of cellular steps, but we must not forget that a pre-computation is needed to get $g(u)$, $h(u)$, and $\Pi(h(u))$ and this pre-computation requires a polynomial number of steps (through a conventional computating model).

The design presented here is very similar to the one given for the Subset-Sum problem in [5]. The multiple common points allow us to be optimistic about future adaptations for other relevant numerical problems. For instance, starting from the skeleton of the design presented in this paper, a solution for the Partition problem is been developed at the moment, and of course we are aiming at a solution for the unbounded Knapsack problem.

The Prolog simulator of P systems [1] is a very useful tool that has helped to debug the design and to understand better how the P systems from the family $\mathbf{\Pi}$ work.

Acknowledgement. Support this research through the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds, is gratefully acknowledged.

References

1. Cordón-Franco, A., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J. Sancho-Caparrini, F.: A Prolog simulator for deterministic P systems with active membranes, *New Generation Computing*, to appear.
2. Păun, G.: Computing with membranes, *Journal of Computer and System Sciences*, **61**(1), 2000, 108–143.
3. Păun, G.: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
4. Păun, G., Rozenberg, G.: A guide to membrane computing, *Theoretical Computer Sciences*, 287, 2002, 73–100.
5. Pérez-Jiménez, M.J., Riscos-Núñez, A.: Solving the Subset-Sum problem by active membranes, *New Generation Computing*, to appear.
6. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: *Teoría de la Complejidad en modelos de computation celular con membranes*, Editorial Kronos, Sevilla, 2002.
7. Pérez-Jiménez, M.J., Romero-Jiménez, A., Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division, *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems*, Budapest, Hungary.