Miguel A. Gutiérrez-Naranjo · Mario J. Pérez-Jiménez
Agustín Riscos-Núñez

A fast P system for finding a balanced 2-partition

Abstract Numerical problems are not very frequently addressed in the P systems literature. In this paper we present an effective solution to the 2-Partition problem via a family of deterministic P systems with active membranes using 2-division. The design of this solution is a sequel of several previous works on other problems, mainly on the Subset-Sum and the Knapsack problems. Several improvements are introduced and explained.

Keywords Complexity class · Membrane computing · Active membranes · **NP**-Complete problem

1 Introduction

Membrane computing (we also say cellular computing) is a recent branch of natural computing initiated in [4]. Its goal is to abstract computing models from the structure and the functioning of living cells. These models, called *P systems*, are parallel distributed devices working with multisets of objects (see [6] for a detailed introduction to membrane computing).

The present paper is focused in the design of a family of P systems that solves a numerical NP-complete problem, and in the formal verification of this solution.

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.

Following the ideas presented in [6] and developed in [11], we first present a *polynomial complexity class* in cellular computing with membranes. Then, we prove that the 2-partition problem belongs to the class of problems which can be solved in a polynomial time by P systems with active membranes.

The paper is organized as follows: first a formal definition of recognizer P systems is given in the next section; then, in Sec. 3 the polynomial complexity class $\mathbf{PMC}_{\mathcal{AM}}$ is introduced; in Sec. (4,5) a cellular solution for the 2-Partition problem is presented, together with some comments, and finally some concluding remarks are given in Sec. 6.

M. A. Gutiérrez-Naranjo (✉) · M. J. Pérez-Jiménez
A. Riscos-Núñez
Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Sevilla University Avda Reina Mercedes s/n,
41012 Sevilla, Spain
E-mail: {magutier, marper, ariscosn}@us.es

2 Preliminaries

The reader is assumed to be familiar with basic elements of membrane computing, e.g., from [6] or [8].

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* $1, \ldots, p$, *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 as above. The initial configuration of* (Π, Σ, i_Π) *with input m is* $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots, \mathcal{M}_p)$.

Remark 1. We denote by I_Π the set of all possible inputs of the P system Π (i.e., I_Π is a collection of multisets over Σ).

The computations of a P system with input a multiset m over Σ, 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 the case of P systems with input and *with external output*, the concept of computation is introduced in a similar way but with a small change. In order to be able to handle the external output of the computations in a formal way, one defines a *membrane structure with environment* by just adding a virtual membrane surrounding the skin, and thus containing the whole system. In this way, the information about the contents of the environment can be included in the configurations of the system.

However, we will not get into details here. In what follows, we will refer to the objects that have been sent out of the system assuming that it is possible to "read" them.

Definition 3. *An accepting P system is a P system with input and with external output, such that the working alphabet contains two distinguished objects:* `Yes` *and* `No`.

Now, imposing some conditions to the behaviour of these accepting P systems, we will get the special subclass that will be used to solve (decision) problems. First, we need to define an *Output* function for our P systems. Given a computation $\mathcal{C} = \{C^i\}_{i<r}$, we will denote by M^j_{env} the content of the environment in the configuration C^j.

Definition 4. *The output of a computation* $\mathcal{C} = \{C^i\}_{i<r}$ *is:*

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

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

Definition 5. *An accepting P system is said to be valid if for every halting computation, and only for them, one symbol* `Yes` *or one symbol* `No` *(but not both) is sent out (in the last step of the computation).*

Definition 6. *We say that $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object* `Yes` *(respectively,* `No`*) is present in the environment associated with the halting configuration of $\mathcal{C}$.*

Definition 7. *A recognizer P system is a valid accepting P system such that all its computations halt.*

These recognizer systems are specially suitable when trying to solve decision problems. The definitions above are stated in a general way, but in this paper P systems within the active membrane model will be used. We refer to [6] (see chapter 7) for a detailed definition of evolution rules, transition steps, configurations and computations in this model.

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, etc.) 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 (see for instance [5] or [12]).

If we wanted to perform such a solution of some decision problem in a laboratory, we will find a drawback on this approach: a system constructed to solve a concrete instance is useless when trying to solve another instance. This issue 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.

Therefore, when attacking a problem in the membrane computing framework, we will design P systems that are able to decide all the instances of "equivalent size", in certain sense. We introduce some preliminary notions before the proper definition of the complexity class is given.

Let us denote by $\mathcal{AM}$ the class of recognizer P systems with active membranes using 2-division only for elementary membranes (see [6], sect 7.2).

Definition 8. *Let L be a language and $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbf{N}}$ a family of P systems. A polynomial encoding of L in $\mathbf{\Pi}$ is a pair (cod, s) of polynomial-time computable functions, $cod : L \to \bigcup_{t\in\mathbf{N}} I_{\Pi(t)}$, and $s : L \to \mathbf{N}$, 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 an 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 $\mathbf{\Pi} = (\Pi(t))_{t\in\mathbf{N}}$ be a family of P systems. 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 $(cod \circ r, s \circ r)$ is a polynomial encoding of L_1 in $\mathbf{\Pi}$.*

For a detailed proof, we refer the reader to [11].

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 [11].

Definition 9. *We 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, $\Pi = \left(\Pi(t)\right)_{t\in\mathbf{N}}$*, with the following properties:*

1. The family Π is consistent with regard to the class $\mathcal{AM}$; that is, $\forall t \in \mathbf{N}\ (\Pi(t) \in \mathcal{AM})$.
2. The family Π is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine constructing $\Pi(t)$ from t in polynomial time.
3. There exists a polynomial encoding (cod, s) of I_X in Π such that:
   - The family Π 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 Π 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 Π 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 we use recognizer P systems).*

We would like to remark also that a *confluence* condition is implicit in the above definition, in the following sense: a P system of the family (that can be nondeterministic) will produce allways the *same output* for a given input, irrespectively of the computation that is chosen in each run. Thus, once the input multiset is introduced we do not need to interact anymore with the system, we know that the system will output `Yes` if and only if the instance has an affirmative answer, and otherwise it will output `No`.

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 [11].

4 Solving the partition problem in linear time

The 2-*partition* problem can be stated as follows:

Given a set $A = \{a_1, \ldots, a_n\}$*, where each element* a_i *has a "weight"* $w_i \in \mathbf{N}$*, decide whether or not there exists a partition of* A *into two subsets such that they have the same weight.*

We will represent the instances of the problem using tuples of the kind $(n, (w_1, \ldots, w_n))$, where n is the size of the set A and $(w_1, \ldots, w_n)$ is the list of weights of the elements from A. We can define in a natural way an additive function w that corresponds to the data in the instance.

We will 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 priority among rules. Our strategy will consist in:

- *Generation stage*: Membrane division is used until a specific membrane for each pair (B, B^c) is obtained, where B is a subset of A that contains the element a_1 (this condition is stated to avoid considering twice the same pair; B^c is the complementary of B).
- *Calculation stage*: In each Membrane the weight of the associated subset and of its complementary are calculated.
- *Checking stage*: In each membrane it is checked if these two weights coincide.
- *Output stage*: The answer is delivered according to the results of the checkings.

The family presented here is

$$\Pi = \{(\Pi(n), \Sigma(n), i(n)) : n \in \mathbf{N}\}\ ,$$

where $\Sigma(n) = \{x_1, \ldots, x_n\}$, $i(n) = e$, and the P system $\Pi(n) = (\Gamma(n), \{e, r, s\}, \mu, \mathcal{M}_e, \mathcal{M}_r, \mathcal{M}_s, R)$ is defined as follows:

- Working alphabet:
  $\Gamma(n) = \Sigma(n) \cup \{a_0, a, b_0, b, c, d_0, d_1, d_2, e_1, \ldots, e_n,$
  $g, g_0, g_1, h_0, h_1, i_1, i_2, i_3, i_4, p, p_0, q,$
  $\mathtt{Yes}, \mathtt{No}, no, z_1, \ldots, z_{2n+1}, \#\}$
- Membrane structure: $\mu = [\ [\ ]_e\ [\ ]_r\ ]_s$.
- Initial multisets:
  $\mathcal{M}_e = e_1 g_1$, $\mathcal{M}_r = b_0 h_0$ and $\mathcal{M}_s = z_1$.
- The set of evolution rules, R, consists of the following rules:

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

The goal of these rules is to generate one membrane for each subset of A that contains a_1. This is done as follows:

- when $[e_i]_e^0$ divides, the negatively charged daughter membrane leaves the generation stage, the other one continues.
- when $[e_i]_e^+$ divides, in the neutrally charged daughter membrane the element $a_{i+1} \in A$ is added to the associated subset, B, but in the other one the element a_{i+1} is added to the complementary subset, B^c.

(b) $[x_1 \to a_0]_e^0$, $[x_1 \to p_0]_e^+$,
$[x_i \to x_{i-1}]_e^+$, for $i = 2, \ldots, n$,
$[x_i \to p_0]_e^-$, for $i = 2, \ldots, n$.

In the beginning, the multiplicities of the objects x_j (with $1 \le j \le n$) encode the weights 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 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 (as explained above), objects a_0 and p_0 are generated to store the weight of this subset and of its complementary. Notice that the electic charge of the membranes in each step is of chief importance in this stage. In Fig. 1 the division process is depicted, including the information about the charge at every moment.

(c) $[e_n]_e^+ \to \#$,
$[a_0 \to \#]_s^0, \quad [p_0 \to \#]_s^0, \quad [g_1 \to \#]_s^0$.

As there are not elements $a_i \in A$ with $i \geq n+1$, the membranes positively charged where the object e_n occurs must leave the generation stage; indeed, they are useless membranes (see membranes marked by a diamond in Fig. 1). These rules perform a "cleaning" task dissolving these spare membranes and erasing the contents that these dissolutions spill in the skin membrane.

(d) $[q \to i_1]_e^-, \quad [p_0 \to p]_e^-, \quad [a_0 \to a]_e^-,$
$[g_1]_e^- \to [\,]_e^- g_0$.

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 a_0 and p_0, whose multiplicities encode the weights of the associated subset and of its complementary, are renamed for the next stage, when their multiplicities are compared. Also an object g_0 is sent out and the total weight of all the elements that have not been considered in the generation stage is added to the complementary.

All these rules are applied simultaneously in the same step, and after this step we say that the membrane becomes *adult*.

(e) $[a]_e^- \to [\,]_e^0\#, \quad [p]_e^0 \to [\,]_e^-\#$.

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

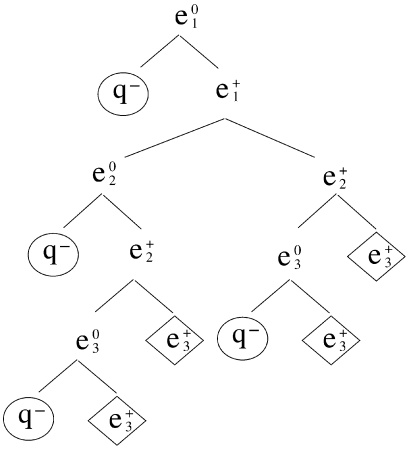

Fig. 1 Membrane division process for $n = 3$

(f) $[i_1 \to i_2]_e^-, \quad [i_2 \to i_1]_e^0$.

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

(g) $[i_1]_e^0 \to [\,]_e^+ no$.

If a subset $B \subseteq A$ verifies that $w(B) > w(B^c)$, then inside the adult membrane associated with it there will be less objects p than a. The moment will come when there are no objects p left, and then the rule $[i_2 \to i_1]_e^-$ will be applied but it will not be possible to apply the rule $[p]_e^0 \to [\,]_e^-\#$ at the same time. Thus, an object i_1 will be present in the membrane and the latter will be neutrally charged, so the rule (g) will be applied ending the checking stage with a negative result.

(h) $[i_2 \to i_3 c]_e^-,$
$[c]_e^- \to [\,]_e^0\#, \quad [i_3 \to i_4]_e^0,$
$[i_4]_e^0 \to [\,]_e^+\texttt{Yes}, \quad [i_4]_e^- \to [\,]_e^+ no$.

If, on the contrary, $w(B) \leq w(B^c)$ holds, then the objects a will be exhausted before the objects p. It is important to distinguish between the cases where the multiplicity of p is strictly greater than the multiplicity of a and the cases where both multiplicities coincide. This is why object c gives again a neutral charge to the membrane and then object i_4 checks if the rule $[p]_e^0 \to [_e]_e^-\#$ is applied or not.

(i) $[p \to \#]_e^+, \quad [a \to \#]_e^+$.

If after the checking loop of rules in (e) is finished there are still some objects p or a in the membrane, they can be erased (again, just for "cleaning" purposes).

(j) $[z_i \to z_{i+1}]_s^0, \quad \text{for} \quad i = 1, \ldots, 2n,$
$[z_{2n+1} \to d_0 d_1]_s^0, \quad d_0[\,]_r^0 \to [d_0]_r^-, \quad [d_1]_s^0 \to [\,]_s^+ d_1,$
$[g_0 \to g]_s^+,$
$g[\,]_e^+ \to [g]_e^0$.

Before the answer is sent out, the system has to make sure that all the adult membranes have finished their checking stages. This is done using the objects g_0 that are present in the skin (recall that they are put there via one of the rules in (d)) and the auxiliary membrane labelled by r (see the next set of rules).

First of all, we wait until the generation stage is over, evolving the counter z_i in the meanwhile. Then, there must be 2^{n-1} copies of g_0, because each adult membrane sends one, and there is one adult membrane for each $B \subseteq A$ such that $a_1 \in B$ (that is, 2^{n-1} in all). Thus we generate objects d_0 and d_1 to "activate" the objects g and the membrane r.

(k) $[h_0 \to h_1]_r^-, \quad [h_1 \to h_0]_r^+,$
$[b_0]_r^- \to [\,]_r^+ b, \quad g[\,]_r^+ \to [g]_r^-,$
$b[\,]_r^- \to [b_0]_r^+, \quad [g]_r^+ \to [\,]_r^- g,$
$[h_0]_r^+ \to [\,]_r^+ d_2, \quad [d_2]_s^+ \to [\,]_s^0 d_2$.

The membrane labelled by r is present in the initial configuration, but remains inactive until an object d_0 "wakes it up". The purpose of this membrane is to perform a loop where the objects g are involved, and thus we can detect if there are no objects g present in the skin region. This fact will mean that all the adult membranes have finished their checking stage, and that the system is ready to send out the answer (Yes or No).

(l) $[no \to \texttt{No}]_s^-$,
$[\texttt{Yes}]_s^- \to [\ ]_s^0\texttt{Yes}$,
$[\texttt{No}]_s^- \to [\ ]_s^0\texttt{No}$.

Finally, the output process is activated. The skin membrane needs to be negatively charged before the answer is sent out. Object d_2 takes care of this (see the previous set of rules) and then, if the answer is affirmative, an object Yes will be sent out recovering the neutral charge for the skin. Note that the answer Yes has priority over the negative answer, in the sense that we first check if there is any object Yes and then, if it is not the case, the answer No will be sent out.

Formal verification: a sketch. Now we will outline the formal verification of the design presented above. That is, we will sketch the proof that this family of P systems actually *solves* in polynomial time the Partition problem (2-*Part*, for short), according to Definition 9.

First of all, it is clear that the design corresponds to a family of P systems with active membranes, with input and with external output. It can be also easily checked that all the systems are deterministic, and after a detailed study, one can conclude that all the computations halt and that an object Yes or an object No is sent to the environment in their last step. Thus, all the P systems from the family are in $\mathcal{AM}$.

The amount of resources needed to build a P system $\Pi(n)$ (size of the alphabet, number of rules, maximal length of rules, and number of membranes and of objects in the initial configuration) is linearly bounded with respect to n, so the polynomially uniform condition is also fulfilled.

Concerning the third condition of Definition 9, and given an instance $u = (n, (w_1, \ldots, w_n))$ of the Partition problem, we can define the two functions *cod* and *s* as follows: $cod(u) = x_1^{w_1} \cdots x_n^{w_n}$ and $s(u) = n$. Then, in a similar way as it is done in [9] and [10], the computation of the P system $\Pi(s(u))$ with input $cod(u)$ can be studied to verify that actually the number of steps is of linear order with respect to $|u|$ (recall that the instance is encoded in an unary fashion in the system, through the input multiset) and that an object Yes (or an object No) is sent out in the last step of the computation, if and only if the instance has an affirmative answer (or a negative answer, respectively).

From the discussion presented here, and according to the Definition 9, we deduce that 2-*Part* $\in \mathbf{PMC}_{\mathcal{AM}}$. Thus, taking into account that this problem is **NP**–complete and the class $\mathbf{PMC}_{\mathcal{AM}}$ is stable under polynomial-time reduction and closed under complement, we also have $\mathbf{NP} \cup \mathbf{co\text{-}NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$.

5 Improving the design

If one studies how the generation stage works, one can notice that the number of spare membranes that are generated and immediately dissolved (the ones with positive charge and containing the object e_n) is actually 2^{n-1}, the same amount as of adult membranes. In the formal model we do not worry about this, because this space is created during the computation, and thus it is not needed *a priori*. But if we try to run a simulation of these P systems on a computer, then the space complexity becomes much more important. Even if we use the trick of dissolving the membranes immediately after being generated, the resources used are too large.

A possible solution is to avoid the generation of such useless membranes. This can be done for example by using a division strategy that follows a complete binary tree structure. We do not use here this idea, because we have the intention to get some of the adult membranes before the generation stage ends, instead of getting all the membranes together after a linear number of steps. This is motivated because we are looking for better efficiency in the best or average case than in the worst case.

Here is a proposal how this can be done:

Generation stage

$[e_i]_e^0 \to [q]_e^0[e_i]_e^+$, *for* $i = 1, \ldots, n-1$,

$[e_i]_e^+ \to [e_{i+1}]_e^0[e_{i+1}]_e^+$, *for* $i = 1, \ldots, n-2$,

$[e_n \to q]_e^0$,

$[e_{n-1}]_e^+ \to [\ ]_e^0\#$,

$[e_i' \to e_{i+1}']_e^+, \quad [e_i'' \to e_{i+1}']_e^+$, *for* $i = 1, \ldots, n-2$,

$[e_i' \to e_i'']_e^0$, $[e_i'' \to \lambda]_e^0$, *for* $i = 1, \ldots, n-1$,

$[e_{n-1}' \to e_n]_e^+, \quad [e_{n-1}'' \to e_n]_e^+$.

In this new approach, we do not produce any useless membrane, so the dissolving rules are no longer needed. Furthermore, only *two* electrical charges are used. Although the sequence of electrical charges of a membrane is still meaningful, the end of the stage is not marked anymore now by getting a negative charge, but by having the neutral charge for two consecutive evolution steps (see [1] for an example of using active membranes with only two charges). This condition is controlled by the "witness-objects" e_i' and e_i'', that show whether in the previous step the charge was neutral (e_i'') or positive (e_i').

If we want to use these rules instead of the rules in (a), then the checking stage needs also to be adapted, and this could be done as follows:

Weight calculation stage

$$[x_1 \to a_0]_e^0, \quad [x_1 \to p_0]_e^+,$$
$$[x'_1 \to a_0]_e^0, \quad [x'_1 \to p_0]_e^+,$$
$$[x_i \to x_{i-1}]_e^+, \quad \text{for} \quad i = 2, \ldots, n,$$
$$[x'_i \to x_{i-1}]_e^+, \quad \text{for} \quad i = 2, \ldots, n,$$
$$[x_i \to x'_i]_e^0, \quad \text{for} \quad i = 2, \ldots, n,$$
$$[x'_i \to p_0]_e^0, \quad \text{for} \quad i = 2, \ldots, n.$$

This stage is almost the same as it was in the former designs, but again it is necessary to introduce "witness-objects" to detect when the generation stage finishes, because in this moment the calculation stage must also conclude.

6 Final remarks

The approach proposed here tries to be as general as possible, and at the same time tries to be *uniform*, in the sense that the design of a family of P systems that solves a problem is not made looking for one P system for each instance of the problem. Instead, each P system of the family can deal with a set of instances of the *same size* (in this case, with the same value of n, independently of the values of the weight function), receiving at the beginning of the computation an input that encodes the concrete instance. It is also important that the number of steps of the computations is polynomial (preferably linear) with respect to the input given; in the case of Partition problem the number of steps is of a linear order.

Several numerical problems have already been solved with similar techniques: the Subset-Sum problem ([9]), the Knapsack problem ([10]) and the Partition problem (in this paper), among others. This fact gives rise to the following question: is it possible to formalize a procedure of "reusing rules"? This question is addressed in [3].

Some first attempts in this direction have already been made, in the framework of a simulator of P systems in Prolog (see [2]). Several files have been created, containing the instructions to generate the evolution rules that deal with the different problems (following the schemes given in the corresponding designs), and now we are working to put these files together and reuse somehow the information.

Another research topic related with these ideas is trying to formalize what means polynomial reduction performed by P systems. It will be nice to have a definition of a complexity class that only depends on P systems parameters, without including a polynomial-time precomputing process (this is somehow unnatural, as one mixes computing devices of essentially different types, Turing machines and P systems).

Acknowledgements The support of 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. Alhazov A, Freund R, Păun Gh (2004) P systems with active membranes and two polarizations. 7:20–36
2. Cordón-Franco A, Gutiérrez-Naranjo MA, Pérez-Jiménez MJ, Sancho-Caparrini F (in press) A Prolog simulator for deterministic P systems with active membranes. New Generation Computing
3. Gutiérrez-Naranjo MA, Pérez-Jiménez MJ, Riscos-Núñez A (2004) Towards a programming language in cellular computing. 7: 247–257
4. Păun Gh (2000) Computing with membranes. Journal of Computer and System Sciences 61(1):108–143
5. Păun Gh (2001) P Systems with active membranes: Attacking NP-complete problems. J Automata, Languages and Combinatorics 6(1): 75–90
6. Păun Gh (2002) Membrane computing. An introduction. Springer, Berlin Heidelberg New York
7. Păun Gh, Riscos-Núñez A, Romero-Jiménez A, Sancho-Caparrini A (eds) (2004) Second Brainstorming Week on Membrane Computing, Sevilla, February 2004. TR 01/2004, Research Group on Natural Computing, Sevilla University
8. Păun Gh, Rozenberg G (2002) A guide to membrane computing. Theoretical Computer Science 287: 73–100
9. Pérez-Jiménez MJ, Riscos-Núñez A (in press) Solving the Subset-Sum problem by active membranes. New Generation Computing
10. Pérez-Jiménez MJ, Riscos-Núñez A (2004) A linear solution for the Knapsack problem using active membranes. In: Martín-Vide C, Mauri G, Păun Gh, Rozenberg G, Salomaa A (eds) (2004) Membrane Computing. International Workshop WMC2003, Tarragona, July 2003, Revised papers. Lecture Notes in Computer Science 2933, Springer, Berlin, 250–268
11. Pérez-Jiménez MJ, Romero-Jiménez A, Sancho-Caparrini F (2003) A polynomial complexity class in P systems using membrane division. In: Csuhaj-Varjú E, Kintala C, Wotschke D, Vaszil G (eds) (2003) Proceedings of the Fifth International Workshop on Descriptional Complexity of Formal Systems, Budapest, Hungary, July 12–14, 284–294
12. Zandron C (2001) A model for molecular computing: Membrane systems. Ph.D. Thesis, Università degli Studi di Milano.