Journal of Universal Computer Science, vol. 10, no. 5 (2004), 650-670
submitted: 1/4/04, accepted: 10/4/04, appeared: 28/5/04 © *J.UCS*

An Efficient Family of P Systems for Packing Items into Bins

Mario J. Pérez-Jiménez, Francisco José Romero-Campero
(University of Sevilla, Spain
{Mario.Perez, Francisco-Jose.Romero}@cs.us.es)

Abstract: In this paper we present an effective solution to the Bin Paching problem using a family of recognizer P systems with active membranes. The analysis of the solution presented here will be done from the point of view of complexity classes. A CLIPS simulator for recognizer P systems is used to describe a session for an instance of Bin Packing, using a P system from the designed family.

Key Words: Membrane computing, Recognizer P systems, Complexity classes, Bin Packing problem, CLIPS.

Category: F.1.1, F.1.3, F.2.1.

1 Introduction

Membrane Computing is an emergent branch of Natural Computing which investigated various distributed parallel computing models based upon the observation that the processes which take place in the complex structure of a living cell can be considered as computations.

Since Gh. Păun introduced this unconventional model of computation in [Păun 2000], several variants have been considered from different approaches. A fairly complete presentation of the domain can be found in [Păun 2002]. Many of the proposed types of P systems have been proved to be computationally complete, that is, equivalent in power to Turing machines. Furthermore, some types of P systems have been proved to be *computationally efficient*, able to solve **NP**-complete problems in polynomial time (see [Păun 2002], Chapter 7).

One more **NP**-complete problem is addressed in this paper, namely the Bin Packing problem.

The solution presented here is designed using a family of recognizer P systems with active membranes. This variant of P systems is studied in many places (see [Pérez et al. 2003b] and [Păun 2002]). In our approach we have followed several ideas and techniques used to solve other numerical **NP**-problems, such as the Subset–Sum in [Pérez and Riscos 2004] and the Knapsack problem in [Pérez and Riscos 2002]. Due to the strong similarities of the design of these solutions the idea of a *cellular programming language* seems possible as it is suggested in [Gutiérrez et al. 2004].

The analysis of the presented solution will be done from the point of view of the complexity classes, specifically, within the framework of the *complexity classes in P systems* studied in [Pérez et al. 2002] and [Pérez et al. 2003b]. A *complexity class* for a model of computation is a collection of problems that can be solved (or languages that can be decided) by devices of the considered type, using *similar* computational resources.

The paper is organized as follows: Section 2 recalls recognizer P systems with active membranes. In section 3 the complexity classes for P systems are briefly introduced. Sections 4, 5, and 6 present a cellular solution to the Bin Packing problem. In section 7 we illustrate the use of a CLIPS simulator for recognizer P systems with active membranes in solving an instance of Bin Packing problem. Conclusions are given in section 8.

2 Recognizer P Systems with Active Membranes

Let us recall that a decision problem, X, is a pair (I_X, θ_X) such that I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X.

In what follows, we assume the reader to be familiar with basic elements of membrane computing, e.g., from [Păun 2002].

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.

The computations of a P system with input m, a multiset 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.

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 $(\mu_0, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots, \mathcal{M}_p)$.

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 change. Namely, we assume that it is not possible to observe the internal processes from the P system, but we can only know that the computation has halted when some distinguished objects are sent out of the skin membrane.

We can formalize these ideas in the following way.

Definition 3. A *recognizer P system* is a P system with input, (Π, Σ, i_Π), and with external output such that:

1. The working alphabet contains two distinguished elements YES, NO.
2. All computations in Π halt.
3. If $\mathcal{C}$ is a computation of Π, then either the object YES or the object NO (but not both) is released into the environment, and only in the last step of the computation. We say that $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object YES (respectively, NO) appears in the environment associated to the corresponding halting configuration of $\mathcal{C}$.

These recognizer systems are especially suitable when trying to solve decision problems.

In this paper we will deal with recognizer P systems with active membranes. Such a system is a tuple

$$\Pi = (\Gamma, H, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_p, R),$$

where:

1. $p \geq 1$, is the initial degree of the system;
2. Γ is the alphabet of symbol-objects;
3. H is a finite set of labels for membranes;
4. μ is a membrane structure, of p membranes, labelled (not necessarily in a one-to-one manner) with elements of H;
5. $\mathcal{M}_1, \ldots, \mathcal{M}_p$ are strings over Γ, describing the initial multisets of objects placed in the p regions of μ;
6. R is a finite set of evolution rules, of the following forms:
   (a) $[\, a \to \omega \,]_h^\alpha$, for $h \in H$,$\alpha \in \{+, -, 0\}$, $a \in \Gamma$, $\omega \in \Gamma^*$.
   Object evolution rules: such a rule is associated with a membrane labelled with h and depends on the polarity of that membrane, but not directly involve the membrane.

   (b) $a\,[\ ]_h^{\alpha_1} \to [\, b \,]_h^{\alpha_2}$, for $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in \Gamma$.
   Communication rules (send in rules): an object from the region immediately outside a membrane labelled with h is introduced in this membrane, possibly transformed into another object; simultaneously, the polarity of the membrane can be changed.

(c) $[\,a\,]_h^{\alpha_1} \rightarrow b\,[\;\,]_h^{\alpha_2}$, for $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in \Gamma$.
Communication rules (send out rules): an object is sent out from a membrane labelled with h to the region immediately outside, possibly transformed into another object; simultaneously, the polarity of the membrane can be changed.

(d) $[\,a\,]_h^{\alpha} \rightarrow b$, for $h \in H$, $\alpha \in \{+, -, 0\}$, $a, b \in \Gamma$.
Dissolving rules: a membrane labelled with h is dissolved in reaction with an object. The skin is never dissolved.

(e) $[\,a\,]_h^{\alpha_1} \rightarrow [\,b\,]_h^{\alpha_2}\,[\,c\,]_h^{\alpha_3}$, for $h \in H$, $\alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}$, $a, b, c \in \Gamma$.
Division rules for elementary membranes: an elementary membrane can be divided into two membranes with the same label, possibly transforming some objects and their polarities.

These rules are applied according to the following principles:

- All the rules are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non deterministic way), but any object which can evolve by one rule of any form, should evolve.
- If a membrane is dissolved, then its content (multiset and internal membranes) is left free in the surrounding region.
- If at the same time a membrane h is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, and then the division is performed. Of course, this process takes only one step.
- The rules associated with membranes labelled with h are used for all copies of this membrane. At one step, a membrane labelled with h can be the subject of *only one* rule of types (b)–(e).

Let us denote by $\mathcal{AM}$ the class of language recognizer P systems with active membranes using 2-division.

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

Roughly speaking, a computational complexity study of a solution for a problem is an estimation of the resources (time, space, etc.) that are required during all 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, hence a system constructed to solve a concrete instance is useless when trying to solve another instance. On the other hand, if we consider P systems with input, then the same system can solve different instances of the problem, provided that the corresponding input multisets are introduced in the input membrane.

Definition 4. Let $\mathcal{F}$ be a class of recognizer P systems. We say that a decision problem $X = (I_X, \theta_X)$ is *solvable in polynomial time* by a family $\Pi = (\Pi(n))_{n\in\mathbf{N}}$, of $\mathcal{F}$, and we denote this by $X \in \mathbf{PMC}_{\mathcal{F}}$, if the following is true:

- The family Π is *polynomially uniform by Turing machines*; that is, there exists a deterministic Turing machine constructing $\Pi(n)$ from $n \in \mathbf{N}$ in polynomial time.
- There exists a pair (g, h) of polynomial-time computable functions $g : L \to \bigcup_{n\in\mathbf{N}} I_{\Pi(n)}$ and $h : L \to \mathbf{N}$ such that for every $u \in L$ we have $g(u) \in I_{\Pi(h(u))}$, and:
  - the family Π is *polynomially bounded* with regard to (g, h); that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(h(u))$ with input $g(u)$ is halting and, moreover, it performs at most, $p(|u|)$ steps;
  - the family Π is *sound*, with regard to (X, g, h); that is, for each $u \in I_X$ it is verified that if there exists an accepting computation of $\Pi(h(u))$ with input $g(u)$, then $\theta_X(u) = 1$;
  - the family Π is *complete*, with regard to (X, g, h); that is, for each $u \in I_X$ it is verified that if $\theta_X(u) = 1$, then every computation of $\Pi(h(u))$ with input $g(u)$ is an accepting one.

In the above definition we have imposed to every P system $\Pi(n)$ to be *confluent*, in the following sense: every computation with the *same* input produces the *same* output.

It is easy to prove that the class $\mathbf{PMC}_{\mathcal{F}}$ is closed under polynomial–time reduction and complement.

4 The Bin Paching Problem

The Bin Packing problem can be stated as follows: *Given a finite set* $A = \{s_1, \dots, s_n\}$*, a weight function* $\omega : A \to \mathbf{N}$ *and two constants* $b \in \mathbf{N}$*,* $c \in \mathbf{N}$*, decide whether or not there exists a partition of* A *into* b *subsets such that their weights do not exceed* c*.*

This problem can be viewed as the situation when given a set of n items and b bins of capacity c, we want to pack the items into bins.

It is well known that Bin Paching is a *strongly* **NP**–complete problem (that is, a problem remaining **NP**–complete even if any instance of length n is restricted to contain integers of size at most $p(n)$, where $p(n)$ is a polynomial). Let us recall that if there exist a *pseudopolynomial* (classical) *algorithm* solving a strongly **NP**–complete problem, then **P**=**NP** (the Knapsack problem is not a strongly **NP**–complete problem).

Bin Packing (especially the corresponding optimization problem) arises in a wide variety of packaging and manufacturing problems.

We will represent the instances of the problem using tuples of the form $(n, (\omega_1, \ldots, \omega_n), b, c)$, where n is the number of items, $(\omega_1, \ldots, \omega_n)$ are the weights, b is the number of bins, and c their capacity.

We will address this 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 be the following.

– For each bin we proceed through the following stages:
  - *Generation stage:* Membrane division is used until a specific membrane for each subset S of the *remaining* items is obtained.
  - *Calculation stage:* In each membrane the weight of its associated subset is computed.
  - *Checking stage :* The condition $\omega(S) \leq c$ is checked for every subset $S \subseteq A$.
  - *Transition stage:* If the associated subset satisfies $\omega(S) \leq c$, then the system continues the computation for the next bin; otherwise the membrane is dissolved.

– *Output stage:* The answer is sent out to the environment according to the results in the checking stage for each bin.

Now we construct a family of recognizer P systems with active membranes using 2-division solving the Bin Packing problem.

Let us consider a polynomial bijection, $\langle\ \rangle$, between $\mathbf{N}^3$ and $\mathbf{N}$ (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 the following:

$$\varPi = \{(\varPi(\langle n, b, c\rangle), \varSigma(n, b, c), i(n, b, c)) \ : \ (n, b, c) \in \mathbf{N}^3\}.$$

For each element of the family, the input alphabet is

$$\varSigma(n, b, c) = \{s_1, \ldots, s_n, r_1, \ldots, r_n\},$$

the input membrane is $i(n,b,c) = 2$, and the P system

$$\Pi(\langle n,b,c\rangle) = (\Gamma(n,b,c),\{1,2\},\mu,\mathcal{M}_1,\mathcal{M}_2,R)$$

is defined as follows:

- Working alphabet:
$$\begin{aligned}\Gamma(n,b,c) = \Sigma(n,b,c) &\cup \{e_{i,j,k}, z_{i,j,k} : 1 \le i \le b-1, -1 \le j \le n, 1 \le k \le n\}\\ &\cup \{E_{i,j,k}, Z_{i,j,k} : 1 \le i \le b-1, -1 \le j \le n, 1 \le k \le n\}\\ &\cup \{g_i : 0 \le i \le 2n+1\} \cup \{ch_j, Ch_j : 0 \le j \le 2c+1\}\\ &\cup \{y_i : 0 \le i \le 2c+3\} \cup \{n_i : 0 \le i \le 2c+6\}\\ &\cup \{w, W, D, \overline{D}, \hat{D}, G, z_0, z, neg, \sharp, YES, NO\}\end{aligned}$$
- Membrane structure: $\mu = [_1\ [_2\ \ ]_2\ ]_1$ (we will say that every membrane with label 2 is an *internal membrane*).
- Initial Multisets: $\mathcal{M}_1 = \emptyset$, $\mathcal{M}_2 = \{g_0, D^c\}$.
- The set of evolution rules, R, consists of the following rules:
  1. $[\ s_k \to e_{1,k,k}\ ]_2^0$, $[\ r_k \to z_{1,k,k}\ ]_2^0$, for $1 \le k \le n$.

     The multiplicity of the objects s_k represents in the initial configuration the weight of the item number k. The object r_k represents the item number k. The objects e_{ijk} and z_{ijk} encode the fact that the item number k occupies the position number j in the order in which the items are considered to be introduced in the bin number i.
  2. $[\ z_{i,1,k}\ ]_2^0 \to [\ z\ ]_2^+\ \ [\ z_{i,-1,k}\ ]_2^0$, for $1 \le k \le n$, $1 \le i \le b-1$.

     The goal of these rules is to generate one membrane for each subset of the remaining items that can be introduced in the current bin. When the object $z_{i,1,k}$ is present in a neutrally charged internal membrane the system produces two membranes: one (positively charged) where the item number k is added to the subset associated with the membrane – the object z appears in this membrane; and another one (neutrally charged) where the item number k is not added to the subset associated with the membrane – the object $z_{i,-1,k}$ appears in this membrane.
  3. $[\ e_{i,j,k}\ \to\ e_{i,j-1,k}]_2^0$,

     $[\ z_{i,j,k}\ \to\ z_{i,j-1,k}]_2^0$, for $0 \le j \le n$, $1 \le k \le n$, $1 \le i \le b-1$.

     These rules decrease the second subscript of the objects e and z. This subscript encodes the order in which the items are considered to be added to the subset associated with the internal membranes.

4. $[\, e_{i,0,k} \;\rightarrow\; w \,]_2^+$, for $1 \leq k \leq n$, $1 \leq i \leq b-1$.

   The multiplicity of the object $e_{i,0,k}$ encodes the weight of the item k and the multiplicity of the object w encodes the weight of the subset associated with the membrane. Thus, when an item is added to the subset associated with a membrane the objects $s_{i,0,k}$ evolve to w.

5. $[\, z \,]_2^+ \;\rightarrow\; \sharp\, [\;\,]_2^0$.

   The object z is used to change the polarization of the internal membranes from positive to neutral.

6. $[\, g_i \;\rightarrow\; g_{i+1} \,]_2^0$, $[\, g_i \;\rightarrow\; g_{i+1} \,]_2^+$, for $0 \leq i \leq 2n-1$,

   $[\, g_{2n} \;\rightarrow\; g_{2n+1}\, ch_0 \,]_2^0$, $[\, g_{2n} \;\rightarrow\; g_{2n+1}\, ch_0 \,]_2^+$, $[\, g_{2n+1} \,]_2^0 \;\rightarrow\; \sharp\, [\;\,]_2^-$.

   The objects g_i are counters used in the generation stage. The object g_{2n+1} changes the polarization of the internal membranes from neutral to negative and the *checking stage* starts.

7. $[\, e_{i,-1,k} \;\rightarrow\; E_{i,k,k} \,]_2^-$,

   $[\, z_{i,-1,k} \;\rightarrow\; Z_{i,k,k} \,]_2^-$, for $1 \leq i \leq b-1$, $1 \leq k \leq n$.

   The objects $e_{i,-1,k}$ and $z_{i,-1,k}$ are renamed to $E_{i,k,k}$ and $Z_{i,k,k}$ at the beginning of the *checking stage* in order to avoid conflicts with the previous stage. Moreover, the position in which the items are considered to be added is restarted by these rules.

8. $[\, w \;\rightarrow\; W \,]_2^-$.

   In the transition to the *checking stage* the objects w are renamed to W in order to avoid conflicts with the previous stage.

9. $[\, D \;\rightarrow\; \overline{D}\,,\, \hat{D} \,]_2^-$.

   The multiplicity of the object D represents the capacity of the bins. In the *checking stage* we have to check if the weight of the subset introduced in the current bin exceeds or not its capacity. At the beginning of this stage the objects D evolve to the objects $\overline{D}$ and $\hat{D}$. The objects $\hat{D}$ are used in the *checking stage* of the current bin, while the objects $\overline{D}$ encode the capacity of the bins so that this information can be used later in the computation.

10. $[\, \hat{D} \,]_2^- \;\rightarrow\; \sharp\, [\;\,]_2^0$, $[\, W \,]_2^0 \rightarrow \sharp\, [\;\,]_2^-$.

    These rules are used to compare the multiplicity of the objects W and $\hat{D}$.

11. $[\, ch_i \;\rightarrow\; ch_{i+1} \,]_2^-$, $[\, ch_i \;\rightarrow\; ch_{i+1} \,]_2^0$, for $0 \leq i \leq 2c-1$,
    $[\, ch_{2c} \;\rightarrow\; ch_{2c+1}\, G\, z_0 \,]_2^-$, $[\, ch_{2c} \;\rightarrow\; ch_{2c+1}\, G\, z_0 \,]_2^0$,
    $[\, ch_{2c+1} \,]_2^- \;\rightarrow\; \sharp\, [\;\,]_2^+$, $[\, ch_{2c+1} \,]_2^0 \;\rightarrow\; \sharp\, [\;\,]_2^+$.

    The objects ch_i are counters used in the *checking stage*. The object ch_{2c+1} changes the polarization of the internal membranes to positive and the *transition stage* starts.

12. $[\, W \,]_2^+ \;\rightarrow\; \sharp$.

    If there are objects W when the *checking stage* has finished, then the multiplicity of the object W exceeded the multiplicity of the object $\hat{D}$. Thus, the weight of the subset introduced in the bin exceeds its capacity, therefore this is not a solution to the problem and the membrane is dissolved.

13. $[\, \hat{D} \;\rightarrow\; \sharp \,]_2^+$, $[\, \overline{D} \;\rightarrow\; D]_2^+$.

    The remaining objects $\hat{D}$ are "erased" in the *transition stage* and the objects $\overline{D}$ are renamed to D so that they can be used in the computation for the next bin.

14. $[\, E_{i,k,k} \;\rightarrow\; e_{i+1,k,k} \,]_2^+$,
    $[\, Z_{i,k,k} \;\rightarrow\; z_{i+1,k,k} \,]_2^+$, for $1 \leq i \leq b-2, 1 \leq k \leq n$.

    The objects $E_{i,k,k}$ and $Z_{i,k,k}$ are renamed to $e_{i+1,k,k}$ and $z_{i+1,k,k}$ so that they can be used in the computation for the next bin.

15. $[\, z_0 \;\rightarrow\; z \,]_2^+$, $[\, G \;\rightarrow\; g_0]_2^+$.

    According to these rules the objects z_0 and G evolve to the objects z and g_0, respectively.

16. $[\, E_{b-1,k,k} \;\rightarrow\; w \,]_2^+$, $[\, Z_{b-1,k,k} \;\rightarrow\; neg \,]_2^+$, for $1 \leq k \leq n$.

    According to these rules, all the remaining objects that represent the weight of the items that can be introduced in the last bin evolve to w, so that these rules introduce all the remaining items in the last bin. The objects $Z_{b-1,k,k}$ evolve to the object neg and this object will force the system to skip the *generation stage* for the last bin.

17. $[\, neg \,]_2^0 \;\rightarrow\; n_0\, [\;\,]_2^-$, $[\, g_1 \;\rightarrow Ch_0\, y_0 \,]_2^-$, $[\, neg \;\rightarrow\; \sharp \,]_2^-$.

    The object neg changes the polarization of the internal membranes from neutral to negative and the object g_1 evolves to the objects Ch_0 (counter in the *checking stage* for the last bin) and y_0. The remaining objects neg are "erased" when the internal membranes become negatively charged.

18. $[\, Ch_i \;\rightarrow\; Ch_{i+1}\,]_2^-$, $[\, Ch_i \;\rightarrow\; Ch_{i+1}\,]_2^0$, for $0 \leq i \leq 2c$,

    $[\, Ch_{2c+1}\,]_2^- \;\rightarrow\; \sharp\,[\;]_2^+$, $[\, Ch_{2c+1}\,]_2^0 \;\rightarrow\; \sharp\,[\;]_2^+$.

    The objects Ch_i are counters used in the *checking stage* of the last bin. The object Ch_{2c+1} changes the polarization of the internal membranes to positive and the *output stage* starts.

19. $[\, y_i \;\rightarrow\; y_{i+1}]_2^0$, $[\, y_i \;\rightarrow\; y_{i+1}]_2^-$, for $0 \leq i \leq 2c+1$,

    $[\, y_{2c+2} \;\rightarrow\; y_{2c+3}\,]_2^+$, $[\, y_{2c+3}\,]_2^+ \rightarrow YES\,[\;]_2^0$.

    The objects y_i are counters in the internal membranes that will eventually send to the skin the object YES.

20. $[\, n_i \;\rightarrow\; n_{i+1}\,]_1^0$, for $0 \leq i \leq 2c+4$,

    $[\, n_{2c+5} \;\rightarrow\; NO\,]_1^0$.

    The objects n_i are counters in the skin that will eventually evolve to the object NO.

21. $[\, YES\,]_1^0 \;\rightarrow\; YES\,[\;]_1^+$, $[\, NO\,]_1^0 \;\rightarrow\; NO\,[\;]_1^-$.

    These rules send out the answer to the environment.

5 An Overview of the Computation

First of all, we must define a suitable pair of polynomial-time computable functions associated with the family Π in order to study its computational complexity. Given an instance $u = (n, (\omega_1, \ldots, \omega_n), b, c)$ of the Bin Packing problem, we define $h(u) = \langle n, b, c\rangle$ (recall the bijection mentioned in the previous section) and $g(u) = \{r_1, \ldots r_n, s_1^{\omega_1}, \ldots, s_n^{\omega_n}\}$. Next, we will informally describe how the system $\Pi(h(u))$ with input $g(u)$ works.

In the initial configuration the multiset from the skin is empty but the multiset from the initial internal membrane (the input membrane) contains the input $g(u)$, one object g_0 (a counter for the *generation stage*) and objects D. The multiplicity of the object D in the initial configuration encodes the capacity of the bins.

In the first step of the computation, the objects s and r evolve, according to the rules in (1), to the objects e and z. The objects e encode the weight of the items and the objects z encode the items. The objects e_{ijk} and z_{ijk} encode the fact that the item number k occupies the position number j in the order in which the items are considered to be introduced in the bin number i.

For each bin i, for $1 \leq i \leq b-1$, the generation and calculation stages take place in parallel, following the instructions from the rules in groups (2) - (5). The system generates every subset of the remaining items, associating each of them with a single internal membrane. Let us describe the evolution of the *subsets associated with internal membranes* during the *generation and calculation stages.*

At the beginning of the *generation stage* of each bin the *subsets associated* with the internal membranes are empty.

When the object $z_{i,1,k}$ appears in a neutrally charged internal membrane, using the rule in (2) the system produces two membranes. One of them is positively charged, and the item number k is added to its *associated subset*; the object z appears in this membrane. The other membrane is neutrally charged, and the item number k is not added to its *associated subset*; the object $z_{i,-1,k}$ appears in this membrane, indicating the fact that this object remains outside the bin so it can be introduced in the next bins.

These two new membranes behave in a different way:

- In the positively charged membrane the weight of its associated subset is updated using the rules in (4) (*calculation stage*). The multiplicity of the object $e_{i,0,k}$ encodes the weight of the item k and the multiplicity of the object w encodes the weight of the subset associated with the membrane. Thus, as the item number k has been added to the associated subset, the objects $s_{i,0,k}$ evolve to w. To carry on with the *generation stage* following the rule in (5), the object z changes the polarization of the membranes from positive to neutral.

- In the neutrally charged membrane, according to the rules in (3) the system decreases the second subscript of the objects e and z that represent the order in which the items are studied. Here the object $z_{i,-1,k}$ represents the fact that the item number k has been left out of the bin number i, hence it can be introduced in the next bins.

The *generation and calculation stages* end when the object g_{2n+1}, according to the last rule in (6), changes the polarization of the internal membranes to negative and the *checking stage* starts. In the previous step, the object g_{2n} evolves following the rules in (6) to g_{2n+1} and ch_0 (a counter used in the *checking stage*).

In the transition to the *checking stage* the system renames the objects e, z and w, following the rules in (7) and (8), in order to avoid conflicts with the previous stages. Moreover, the second subscript of the objects e and z are set equal to their third subscript restarting the position in which the items are studied. Finally, in the transition to the *checking stage* the objects D evolve, using the rule in (9), to the objects $\overline{D}$ and $\hat{D}$. The objects $\hat{D}$ are used in the *checking stage* of the current bin, while the objects $\overline{D}$ encode the capacity of the bins so that this information can be used later in the computation.

The purpose of the *checking stage* is to compare the multiplicities of objects W and $\hat{D}$. This task is carried out by the rules in (10). The objects ch_i are counters used in this stage; the stage ends when the object ch_{2c+1}, according to the last rules in (11), changes the polarization of the internal membranes to positive. At the end of the *checking stage*, if there are objects W in an internal membrane, then the multiplicity of the object W exceeded the multiplicity of the object $\hat{D}$. This means that the weight of the subset associated with this membrane exceeds the capacity of the bins, therefore this is not a solution to the problem and the membrane is dissolved by the rule in (12).

In the last steps of the *checking stage* the object ch_{2c}, following the rules in (11), evolve to the objects ch_{2c+1}, G, and z_0. The last two objects are used in the *transition stage* to the next bin. Using the rules in (15) the object G evolves to the object g_0 (a counter for the *generation stage*) and the object z_0 to the object z (this object changes the polarization to neutral). At the same time, the system, applying the rules in (14), increases the first subscript of the objects e and z. This subscript represents the bin that is being studied. Now, the *generation stage* for the next bin starts, and we continue in this way until the system considers the last bin.

The computation for the last bin is quite different from the previous bins. The system, using the first rule in (16), introduces all the remaining items in the last bin. When the second rule in (16) is applied the object neg appears in the internal membranes. This object changes the polarization of the membranes to negative and so the system skips the *generation stage* for the last bin and the *checking stage* starts.

According to the rules in (17), the object n_0 (an object that eventually produces the object NO) is sent to the skin and the objects Ch_0 (a counter in the *checking stage*) and y_0 (an object that eventually produces the object YES) appear in the internal membranes. The *checking stage* for the last bin ends when the object Ch_{2c+1}, following the rules in (18), changes the polarization of the internal membranes to positive and the system, using the rule in (12), checks whether or not the weights of the associated subsets exceed the capacity of the bins.

At the end of the *checking stage* of the last bin, all the internal membranes that have not been dissolved codify solutions to the instance of the Bin Packing problem that was introduced as input. Thus, these membranes, according to the rules in (19), send to the skin the object YES. On the other hand, if at the end of the *checking stage* there are no internal membranes, then the object NO appears in the skin following the rules in (20).

In the last step of the computation, the rules in (21) send out the answer to the environment. Note that the object NO appears a step later than the object YES in order to send out the right answer.

6 Required Resources

The presented family of recognizer P systems solving the Bin Packing problem is polynomially uniform by Turing machines. It can be observed that the definition of the family is done in a recursive manner, starting from a given instance, in particular from the constants n, b, and c. Furthermore the required resources to build an element of the family are:

– Size of the alphabet: $4n^2b - 8nb - 4n^2 - 6n + 8c + 28 \in O(\max\{n, b, c\}^3)$.

– Number of membranes: $2 \in \Theta(1)$.

– $|\mathcal{M}_1| + |\mathcal{M}_2| = c + 1 \in O(c)$.

– Sum of the rules' lengths: $28n^2b + 115nb - 28n^2 - 63n + 140c + 404 \in O(\max\{n, b, c\}^3)$.

Note that the instance $u = (n, (\omega_1, \ldots, \omega_n), b, c)$ is introduced in the initial configuration as an input multiset, that is, encoded in an unary representation, and thus we have $|u| \in O(n + \omega_1 + \cdots + \omega_n)$.

The number of steps in each stage are the following:

1. First stage: 1 step.
2. For each bin from 1 to $b - 1$:
   – Generation and calculation stages: $2n + 1$ steps.
   – Transition to the checking stage: 2 steps.
   – Checking stage: $2c + 1$ steps.
   – Transition to the next bin: 3 steps.
3. For the bin number b: $2c + 8$ steps.
4. In the output stage: 1 step.

Therefore, the overall number of steps is: $2nb + 2bc - 2n + 7b \in O(\max\{n, b, c\}^2)$. From this analysis we obtain the following results:

1. $\mathtt{BINPACKING} \in \mathbf{PMC}_{\mathcal{AM}}$.
2. $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$.

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

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

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

7 A CLIPS Session for $n = b = c = 2$

In this section we present a session with the CLIPS simulator presented in [Pérez and Romero 2004] for an instance of the Bin Packing problem, namely, for $u = (2, (1, 2), 2, 2)$.

```
CLIPS (V6.10 07/01/98)
CLIPS> (load "SIMULATOR4.clp")
CLIPS> (reset)
CLIPS> (run)

Write the path and the file where the P-system is written:
/home/Fran/binpacking.clp

                    *******************************
                    * P-SYSTEM SUCESSFULLY LOADED *
                    *******************************

Write the value of the parameter n : 2
Write the value of the parameter c : 2
Write the value of the parameter b : 2

Write the input multiset following the instructions given above:
r 1 , s 1 , r 2 , s 2 , s 2

Configuration number: 0

[environment [multiset ]]

[skin
   [children 2]
   [label 1] [polarity 0]
   [multiset , ,]]

[membrane
   [number 2] [children ] [father 1]
   [label 2] [polarity 0]
   [multiset , g 0 , D , D , r 1 , s 1 , r 2 , s 2 , s 2 ,]]


Configuration number: 1

[environment [multiset ]]
```

```
[skin
   [children 2]
   [label 1] [polarity 0]
   [multiset , ,]]

[membrane
   [number 2] [children ] [father 1]
   [label 2] [polarity 0]
   [multiset , g 1 , D , D , z 1 1 1 , e 1 1 1 , z 1 2 2 ,
                                    e 1 2 2 , e 1 2 2 ,]]
```

The initial configuration and the first step of the computation are illustrated above.

Next, the *generation stage* takes place. In the first configuration of this stage it can be seen how it works. The system produces two membranes, one where the item number 1 is placed in the first bin and another one where the item number 1 is left out of the first bin.

```
Configuration number: 2

[environment [multiset ]]

[skin
   [children 3 4]
   [label 1] [polarity 0]
   [multiset , ,]]

[membrane
   [number 4] [children ] [father 1]
   [label 2] [polarity 0]
   [multiset , g 2 , D , D ,
               z 1 -1 1 , e 1 0 1 , z 1 1 2 , e 1 1 2 , e 1 1 2 ,]]

[membrane
   [number 3] [children ] [father 1]
   [label 2] [polarity +]
   [multiset , g 2 , D , D , z ,
               e 1 0 1 , z 1 1 2 , e 1 1 2 , e 1 1 2 ,]]
```

The *checking stage* begins when the objects g_5 and ch_0 appear in the internal membranes. In our example, in this moment the system has four internal membranes representing every subset of the set of two items.

```
Configuration number: 5

[environment [multiset ]]

[skin
   [children 7 8 5 6]
   [label 1] [polarity 0]
   [multiset , , # ,]]
```

```
[membrane
   [number 8] [children ] [father 1]
   [label 2] [polarity 0]
   [multiset , g 5 , ch 0 , D , D , w ,
              z 1 -1 2 , e 1 -1 2 , e 1 -1 2 ,]]

[membrane
   [number 5] [children ] [father 1]
   [label 2] [polarity 0]
   [multiset , g 5 , ch 0 , D , D ,
              z 1 -1 1 , e 1 -1 1 , w , w ,]]

[membrane
   [number 6] [children ] [father 1]
   [label 2] [polarity 0]
   [multiset , g 5 , ch 0 , D , D ,
              z 1 -1 1 , e 1 -1 1 , z 1 -1 2 , e 1 -1 2 , e 1 -1 2 ,]]

[membrane
   [number 7] [children ] [father 1]
   [label 2] [polarity 0]
   [multiset , g 5 , ch 0 , D , D , w , w , w ,]]
```

The *checking stage* ends when the objects ch_5, G, and z_0 appear in the internal membranes. Here we can see that only the subsets associated with three of the four internal membranes satisfy the condition that their weights do not exceed the capacity of the bins. Therefore, the membrane number 7 will be dissolved.

```
Configuration number: 11

[environment [multiset ]]

[skin
   [children 7 8 5 6]
   [label 1] [polarity 0]
   [multiset , # , # ,]]

[membrane
   [number 8] [children ] [father 1]
   [label 2] [polarity 0]
   [multiset , ch 5 , G , z 0 , D- , D- ,
              Z 1 2 2 , E 1 2 2 , E 1 2 2 ,]]

[membrane
   [number 6] [children ] [father 1]
   [label 2] [polarity 0]
   [multiset , ch 5 , G , z 0 , D- , D^ , D- ,
              Z 1 1 1 , E 1 1 1 , Z 1 2 2 , E 1 2 2 , E 1 2 2 ,]]

[membrane
   [number 5] [children ] [father 1]
   [label 2] [polarity -]
   [multiset , ch 5 , G , z 0 , D- , D- , Z 1 1 1 , E 1 1 1 ,]]
```

```
[membrane
   [number 7] [children ] [father 1]
   [label 2] [polarity -]
   [multiset , ch 5 , G , z 0 , D- , D- , W ,]]
```

There is no *generation stage* for the last bin, because all the remaining items are introduced in this bin. The *checking stage* for the last bin starts when the object Ch_0 appears in the internal membranes.

```
Configuration number: 16

[environment [multiset ]]

[skin
   [children 8 5 6]
   [label 1] [polarity 0]
   [multiset , n 1 , n 1 , n 1 ,]]

[membrane
   [number 5] [children ] [father 1]
   [label 2] [polarity -]
   [multiset , Ch 0 , y 0 , D- , D^ , D- , D^ , W ,]]

[membrane
   [number 6] [children ] [father 1]
   [label 2] [polarity -]
   [multiset , Ch 0 , y 0 , D- , D^ , D- , D^ , W , W , W ,]]

[membrane
   [number 8] [children ] [father 1]
   [label 2] [polarity -]
   [multiset , Ch 0 , y 0 , D- , D^ , D- , D^ , W , W ,]]
```

At the end of the *checking stage* for the last bin, we can see that there are only two possible solutions to this instance of the Bin Packing problems codified in the membranes number 5 and 8.

```
Configuration number: 21

[environment [multiset ]]

[skin
   [children 8 5 6]
   [label 1] [polarity 0]
   [multiset , n 6 , n 6 , n 6 ,]]

[membrane
   [number 5] [children ] [father 1]
   [label 2] [polarity 0]
   [multiset , Ch 5 , y 5 , D- , D- ,]]

[membrane
```

```
   [number 6] [children ] [father 1]
   [label 2] [polarity -]
   [multiset , Ch 5 , y 5 , D- , D- , W ,]]

[membrane
   [number 8] [children ] [father 1]
   [label 2] [polarity -]
   [multiset , Ch 5 , y 5 , D- , D- ,]]
```

In the next three steps, the system sends out the answer to the environment.

```
Configuration number: 23

[environment [multiset ]]

[skin
   [children 8 5]
   [label 1] [polarity 0]
   [multiset , n 8 , ... , # ,]]

[membrane
   [number 8] [children ] [father 1]
   [label 2] [polarity +]
   [multiset , y 7 , D , D ,]]

[membrane
   [number 5] [children ] [father 1]
   [label 2] [polarity +]
   [multiset , y 7 , D , D ,]]

Configuration number: 24

[environment [multiset ]]

[skin
   [children 8 5]
   [label 1] [polarity 0]
   [multiset , n 9 , ... , YES , YES ,]]

[membrane
   [number 5] [children ] [father 1]
   [label 2] [polarity +]
   [multiset , D , D ,]]

[membrane
   [number 8] [children ] [father 1]
   [label 2] [polarity +]
   [multiset , D , D ,]]

Configuration number: 25

[environment [multiset , YES ,]]

[skin
   [children 8 5]
```

```
   [label 1] [polarity +]
   [multiset , NO , ... , YES ,]]

[membrane
   [number 5] [children ] [father 1]
   [label 2] [polarity +]
   [multiset , D , D ,]]

[membrane
   [number 8] [children ] [father 1]
   [label 2] [polarity +]
   [multiset , D , D ,]]

The system has reached a halting configuration in the step number 25
and the element YES has been sent out to the environment.
```

8 Conclusions

In this paper we have presented an effective solution for the Bin Packing problem using a family of recognizer P systems with active membranes. This has been done in the framework of complexity classes in cellular computing with membranes.

The design presented here is very similar to the solutions to numerical **NP**-complete problems studied in [Pérez and Riscos 2004], [Pérez and Riscos 2002] and [Gutiérrez et al. 2004]. The strong similarities of the solutions to these problems show that the idea of a *cellular programming language is possible*, indicating some "subroutines" that can be used in a variety of situations and therefore could be useful for addressing new problems in the future. As an example of the usefulness of the subroutines outlined in [Gutiérrez et al. 2004], let us see how the design of the solutions for the Bin Packing would look like:

BINPACKING
for $i = 1, \ldots, b-1$ do
   $gen-subsets(n_i)$
   $calc-weight(n_i)$
   $rename$
   $check-weight$
   $marker-leq$
   $counter(n)$
   $clean-dissolve$
end for.
$calc-weight(n_b)$

$$\begin{array}{l} rename \\ check - weight \\ marker - leq \\ counter(n) \\ clean - dissolve \\ detector \\ answer \end{array}$$

The CLIPS simulator for P systems presented in [Pérez and Romero 2004] is a very useful tool that has helped to debug the design and to understand better how the P systems from the family Π work.

Acknowledgement

This work is supported by the Ministerio de Ciencia y Tecnología of Spain, by the *Plan Nacional de I+D+I (2000–2003)* (TIC2002-04220-C03-01), cofinanced by FEDER funds, and, in the case of the second author, by a FPI fellowship from the University of Seville.

References

[CLIPS web] CLIPS Web Page: `http://www.ghg.net/clips/CLIPS.html`

[Cordón et al. 2004] 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, in press

[Gutiérrez et al. 2004] Gutiérrez-Naranjo M.A., Pérez-Jiménez M.J., Riscos-Núñez A.: "Towards a programming language in cellular computing"; 11th Workshop on Logic, Language, Information and Computation (WoLLIC'2004), July 19-22, 2004, Campus de Univ. Paris 12, Paris, France

[Păun 2000] Păun Gh.: "Computing with membranes"; Journal of Computer and Systems Sciences, 61, 1 (2000), 108–143

[Păun et al. 2000] Păun Gh., Rozenberg G., Salomaa A.: "Membrane computing with external output"; Fundamenta Informaticae, 41, 3 (2000), 313–340

[Păun 2002] Păun Gh.: "Membrane Computing. An Introduction"; Springer, Berlin (2002)

[Păun and Rozenberg 2002] Păun Gh., Rozenberg G.: "A guide to membrane computing"; Theoretical Computer Science, 287 (2002), 73–100

[Pérez et al. 2002] Pérez-Jiménez M.J., Romero-Jiménez A., Sancho-Caparrini F.: "Teora de la complejidad en modelos de computacion celular con membranas"; Editorial Kronos, 2002

[Pérez et al. 2003a] Pérez-Jiménez M.J., Romero-Jiménez A., Sancho-Caparrini F.: "Solving VALIDITY problem by active membranes with input"; "Proceedings of the Brainstorming Week on Membrane Computing" (M. Cavaliere, C. Martín-Vide, Gh. Păun, eds.), Report GRLMC 26/03, University of Tarragona (2003), 279–290

[Pérez et al. 2003b] 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" (E. Csuhaj-Varjú, C. Kintala, D. Wotschke, Gy. Vaszyl, eds.), Budapest (2003), 284–294

[Pérez and Romero 2004] Pérez-Jiménez M.J., Romero-Campero F.J.: "A CLIPS Simulator for Recognizer P Systems with Active Membranes"; "Proceedings of the Second Brainstorming Week on Membrane Computing" (Gh. Păun, A. Romero, A. Riscos, F. Sancho-Caparrini, eds.), Report 01/04, Research Group on Natural Computing, University of Seville (2004), 387–413

[Pérez and Riscos 2002] Pérez-Jiménez M.J., Riscos-Núñez,, A.: "A linear-time solution for the Knapsack problem using active membranes"; Lecture Notes in Computer Science 2933, Springer, Berlin (2004), 140–152

[Pérez and Riscos 2004] Pérez-Jiménez M.J., Riscos-Núñez A.: "Solving the Subset-Sum problem by active membranes"; New Generation Computing, in press

[P systems web] The P Systems Web Page: `http://psytems.disco.unimib.it/`