# Solving Multidimensional 0-1 Knapsack Problem by P Systems with Input and Active Membranes

**Linqiang PAN**[1,2], **Carlos MARTIN-VIDE**[2]

[1]Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, People's Republic of China
E-mail: `lqpan@mail.hust.edu.cn`

[2]Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain

**Abstract.** P systems are parallel molecular computing models based on processing multisets of objects in cell-like membrane structures. In this paper we give a membrane algorithm to multidimensional 0–1 knapsack problem in linear time by recognizer P systems with input and with active membranes using 2-division. This algorithm can also be modified to solve general 0–1 integer programming problem.

## 1 Introduction

The P systems are a class of distributed parallel computing devices of a biochemical type, introduced in [4], which can be seen as a general computing architecture where various types of objects can be processed by various operations. It comes from the observation that certain processes which take place in the complex structure of living organisms can be considered as computations. Since Gh. Păun introduced it, computer scientists and biologists, et al. have contributed enriching the field with their different points of view. For a motivation and detailed description of various P system models, please refer to [4, 6].

Membrane division – inspired from cell division well-known in biology – is the most investigated way for obtaining an exponential working space in a linear time, and solving on this basis hard problems, typically **NP**-complete problems, in polynomial (often, linear) time. Details can be found in [5, 6, 11]. Recently, **PSPACE**-complete problems were also attacked in this way (see [13, 1]).

In [8] Pérez-Jiménez et al. solve satisfiability problem in linear time with respect to the number of variables and clauses of propositional formula by recognizer P systems with input and with active membranes using 2-division. Thus the multidimensional 0–1 knapsack problem belonging to the class of **NP**-complete problems can also be solved in a polynomial time by P systems with input and with active membranes using 2-division. One can get this kind of solution by the reduction of multidimensional 0–1 knapsack problem to satisfiability problem in order to apply those P systems which solve satisfiability problem in a linear time. But the process of reduction is usually cumbersome and time consuming (polynomial time). On the other hand, it still remains open that how one can reduce an **NP** problem to another **NP**-complete problem by P systems. So, in this paper, we directly give a membrane algorithm to solve multidimensional 0–1 knapsack problem in linear time by recognizer P systems with input and with active membranes using 2-division. Here, we focus in the design of a family of P systems that solves multidimensional 0–1 knapsack problem, not in the formal verification of the membrane algorithm. As discussed in section 4, this algorithm is not difficult to be modified to solve the general 0–1 integer programming problem.

The paper is organized as follows: in section 2 the notion of recognizer P system is introduced, which is the model of computation to solve multidimensional 0–1 knapsack problem, and the polynomial complexity class in computing with membranes is recalled; section 3 gives a membrane algorithm to solve multidimensional 0–1 knapsack problem in linear time by recognizer P systems with active membranes using 2-division; in section 4 some discussion is presented.

## 2 P Systems

We start by introducing P systems with active membranes due to [5], where more details can also be found.

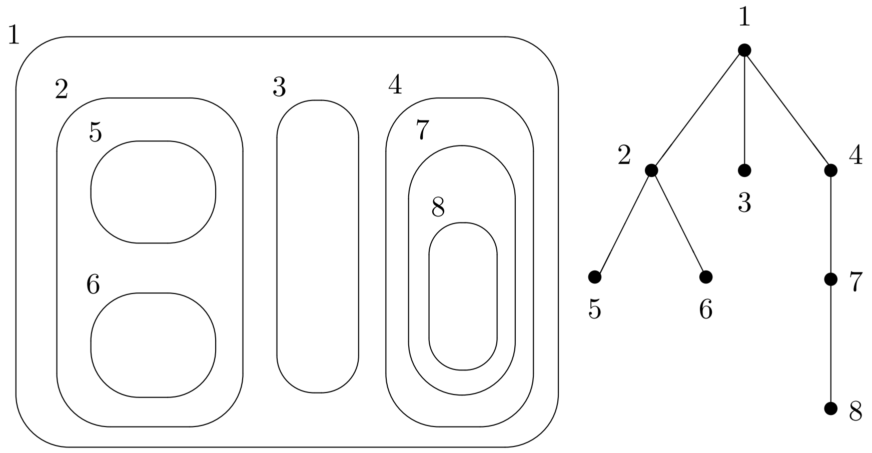

Figure 1: A membrane structure and its associated tree

A *membrane structure* is represented by a Venn diagram and is identified by a string of correctly matching parentheses, with a unique external pair of parentheses; this external pair of parentheses corresponds to the external membrane, called the *skin*. A membrane without any other membrane inside is said to be *elementary*. For instance, the structure in Figure 1 contains 8 membranes; membranes 3, 5, 6 and 8 are elementary. The string of parentheses identifying this structure is

$$\mu = [_1[_2[_5\ ]_5[_6\ ]_6]_2[_3\ ]_3[_4[_7[_8\ ]_8]_7]_4]_1.$$

All membranes are labeled; here we have used the numbers from 1 to 8. We say that the number of membranes is the *degree* of the membrane structure, while the height of the tree associated in the usual way with the structure is its *depth*. In the example above we have a membrane structure of degree 8 and of depth 4.

In what follows, the membranes can be marked with $+$ or $-$, and this is interpreted as an "electrical charge", or with 0, and this means "neutral charge". We will write $[_i\ ]_i^+, [_i\ ]_i^-, [_i\ ]_i^0$ in the three cases, respectively.

The membranes delimit *regions*, precisely identified by the membranes (the region of a membrane is delimited by the membrane and all membranes placed immediately inside it, if any such a membrane exists). In these regions we place *objects*, which are represented by symbols of an alphabet. Several copies of the same object can be present in a region, so we work with *multisets* of objects. A multiset over an alphabet $V$ is represented by a string over $V$: the number of occurrences of a symbol $a \in V$ in a string $x \in V^*$ ($V^*$ is the set of all strings over $V$; the empty string is denoted by $\lambda$) is denoted by $|x|_a$ and it represents the multiplicity of the object $a$ in the multiset represented by $x$.

A *P system with active membranes and 2-division* is a construct

$$\Pi = (O, H, \mu, w_1, \ldots, w_m, R),$$

where:

(i) $m \geq 1$ (the initial degree of the system);

(ii) $O$ is the alphabet of *objects*;

(iii) $H$ is a finite set of *labels* for membranes;

(iv) $\mu$ is a *membrane structure*, consisting of $m$ membranes, labelled (not necessarily in a one-to-one manner) with elements of $H$;

(v) $w_1, \ldots, w_m$ are strings over $O$, describing the *multisets of objects* placed in the $m$ regions of $\mu$;

(vi) $R$ is a finite set of *developmental rules*, of the following forms:

(a) $[_h a \rightarrow v]_h^{\alpha}$,
for $h \in H, \alpha \in \{+, -, 0\}, a \in O, v \in O^*$
(object evolution rules, associated with membranes and depending on the label and the charge of the membranes, but not directly involving the membranes, in the sense that the membranes are neither taking part in the application of these rules nor are they modified by them);

(b) $a[_h\ ]_h^{\alpha_1} \rightarrow [_h b]_h^{\alpha_2}$,
for $h \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in O$
(communication rules; an object is introduced in the membrane, possibly modified during this process; also the polarization of the membrane can be modified, but not its label);

(c) $[_h a\ ]_h^{\alpha_1} \rightarrow [_h\ ]_h^{\alpha_2} b$,
for $h \in H, \alpha_1, \alpha_2 \in \{+, -, 0\}, a, b \in O$
(communication rules; an object is sent out of the membrane, possibly modified during this process; also the polarization of the membrane can be modified, but not its label);

(d) $[_h a\ ]_h^\alpha \rightarrow b$,
for $h \in H, \alpha \in \{+,-,0\}, a, b \in O$
(dissolving rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

(e) $[_h a\ ]_h^{\alpha_1} \rightarrow [_h b\ ]_h^{\alpha_2} [_h c\ ]_h^{\alpha_3}$,
for $h \in H, \alpha_1, \alpha_2, \alpha_3 \in \{+,-,0\}, a, b, c \in O$
(division rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label, possibly of different polarizations; the object specified in the rule is replaced in the two new membranes by possibly new objects);

(f) $[_{h_0} [_{h_1}\ ]_{h_1}^{\alpha_1} \ldots [_{h_k}\ ]_{h_k}^{\alpha_1} [_{h_{k+1}}\ ]_{h_{k+1}}^{\alpha_2} \ldots [_{h_n}\ ]_{h_n}^{\alpha_2} ]_{h_0}^{\alpha_0}$
$\rightarrow [_{h_0} [_{h_1}\ ]_{h_1}^{\alpha_3} \ldots [_{h_k}\ ]_{h_k}^{\alpha_3} ]_{h_0}^{\alpha_5} [_{h_0} [_{h_{k+1}}\ ]_{h_{k+1}}^{\alpha_4} \ldots [_{h_n}\ ]_{h_n}^{\alpha_4} ]_{h_0}^{\alpha_6}$,
for $k \geq 1, n > k, h_i \in H, 0 \leq i \leq n$, and $\alpha_0, \ldots, \alpha_6 \in \{+,-,0\}$ with $\{\alpha_1, \alpha_2\} = \{+,-\}$; if the membrane with the label $h_0$ contains other membranes than those with the labels $h_1, \ldots, h_n$ specified above, then they must have neutral charges in order to make this rule applicable; these membranes are duplicated and then are part of the contents of both new copies of the membrane $h_0$
(division of non-elementary membranes; this is possible only if a membrane contains two immediately lower membranes of opposite polarization, + and −; the membranes of opposite polarizations are separated in the two new membranes, but their polarization can change; always, all membranes of opposite polarizations are separated by applying this rule).

For a detailed description of using these rules we refer to [5, 6]. Here we only mention that the rules are used in the non-deterministic maximally parallel manner customary in membrane computing in the bottom-up manner: in any given step, one uses first the evolution rules of type (a), then the other rules which also involve a membrane; moreover, one uses first the rules of types (b), (c), (d), (e), and then those of type (f). It is important to note that at one step a membrane $h$ can be subject of only one rule of types (b)-(f). In this way, we get transition from a configuration of the system to the next configuration. A sequence of transitions is a computation. A computation is halting if no other rules can be applied in its last configuration.

To understand what it means that a problem can be solved in polynomial time by P systems, it is necessary to recall some complexity measure for P systems as described in [9].

Consider a decision problem $A$ and denote by $A(n)$ an instance of $A$ of size $n$. Given a class $X$ of membrane systems and a total function $f : \mathbb{N} \rightarrow \mathbb{N}$ (for example, linear and polynomial functions), we say that problem $A$ belongs to $\mathbf{MC}_X(f)$ if a family of membrane systems $\Pi_A = (\Pi_A(1), \Pi_A(2), \ldots)$ of type $X$ exists such that:

1. $\Pi_A$ is a *uniform* family: there is a Turing machine which constructs $\Pi_A(n)$ in polynomial time starting from $n$.

2. Each $\Pi_A(n)$ is *confluent*: there is a distinguished object `yes` such that either in every computation of $\Pi_A(n)$ the object `yes` is send out from the system, or this happens in no computation.

3. $\Pi_A(n)$ is *sound*: that is, $\Pi_A(n)$ sends out the object yes if and only if the answer to $\Pi_A(n)$ is "yes".

4. $\Pi_A$ is *f-efficient*: that is, $\Pi_A$ always halts in at most $f(n)$ steps.

The polynomial complexity classes associated with a family of membrane systems, $X$, are defined as follows:

$$\mathbf{PMC}_X = \bigcup_{f \text{ polynomial}} \mathbf{MC}_X(f).$$

In [6], the definition of these complexity classes is based on a *semi-uniform* construction of P systems solving a problem $A$: one starts not from $n$, but from an instance $A(n)$. For a clearer description of the difference between uniform P systems and semi-uniform P systems, please refer to [7].

In what follows, we use *recognizer P systems*. First of all, following [7, 9] we consider P systems with input. Such a device is a tuple $(\Pi, \Sigma, i_0)$, where:

– $\Pi$ is a P system, with the alphabet of objects $\Gamma$ and initial multisets $w_1, \cdots, w_m$ (associated with membranes labelled by $1, \cdots, m$, respectively).

– $\Sigma$ is an (input) alphabet strictly contained in $\Gamma$ and such that $w_1, \cdots, w_m$ are multisets over $\Gamma - \Sigma$.

– $i_0$ is the label of a distinguished membrane (of input).

If $w$ is a multiset over $\Sigma$, then the initial configuration of $(\Pi, \Sigma, i_0)$ with input $w$ is $(\mu, w'_1, \cdots, w'_m)$, where $w'_i = w_i$ for $i \neq i_0$, and $w'_{i_0} = w_{i_0} \cup w$.

The computations of a P system with input are defined in a natural way. Note that the initial configuration is obtained by adding the input multiset $w$ over $\Sigma$ to the initial configuration of the system $\Pi$.

Now, a recognizer P system is a P system with input, $(\Pi, \Sigma, i_0)$, such that:

1. The alphabet of objects contains two distinguished elements yes, no.

2. All computations of the system halt.

3. If $\mathcal{C}$ is a computation of $\Pi$, then either the object yes or the object no (but not both) have to be sent out to the environment, and only in the last step of the computation.

We say that $\mathcal{C}$ is an accepting (respectively, rejecting) computation if the object yes (respectively, no appears in the environment in the halting configuration of $\mathcal{C}$.

## 3 Solving Multidimensional 0–1 Knapsack Problem by Recognizer P Systems with Active Membranes

### 3.1 Problem Formulation

The *0–1 Multidimensional Knapsack Problem* (MKP) is a well known **NP**-complete combinatorial problem [2]. The decision MKP can be formulated as follows: given an integer $k$, an objective function $f(x_1, \cdots, x_n) = \sum_{j=1}^{n} c_j x_j$, and constraints $\sum_{j=1}^{n} w_{ij} x_j \leq b_i$, for $i = 1, \cdots, m$, $x_j \in \{0, 1\}$, for $j = 1, \cdots, n$, where $c_j$, $w_{i,j}$ and $b_i$ are nonnegative integers, decide whether or not there exists an assignment of variables $x_j$ such that it satisfies the constraints and the objective function is greater than or equal to $k$.

MKP is an important combinatorial optimization problem both from a theoretic and practical point of view, which can formulate many practical problems such as capital budgeting where project $j$ has profit $c_j$ and consume $(w_{ij})$ units of resource $i$. The goal is to determine a subset of the $n$ projects such that the total profit is maximized and all resource constrains are satisfied. Other important applications include cargo loading [12] cutting stock problems, and processor allocation in distributed systems [3].

The special case of MKP with $m = 1$ is the classical knapsack problem (KP). It is well known that the KP is not strongly NP-hard because there are polynomial approximation algorithms to solve it. This is not the case for the general MKP. In the framework of cellular computing, a membrane algorithm to solve KP is developed [10]. In the next subsection, we will give membrane algorithm for the general MKP.

## 3.2 Membrane Algorithm for Multidimensional 0–1 Knapsack Problem

We present a solution of MKP via a brute force algorithm, in the framework of recognizer P systems with active membranes using 2-division.

Given an instance $u$ of MKP as shown in the above subsection, for convenience, we call $\sum\limits_{j=1}^{n} w_{ij}x_j \leq b_i$ $(1 \leq i \leq m)$ the $i$th constraint inequality, and $\sum\limits_{j=1}^{n} c_j x_j \geq k$ the $(m+1)$th inequality.

Let us consider a polynomial bijection, $\langle\ \rangle$, between $\mathbb{N}^{*l}$ $(l \geq 2)$ and $\mathbb{N}^*$, defined as follows: $\langle y_1, y_2\rangle = (y_1 + y_2)(y_1 + y_2 + 1)/2 + y_1$, $\langle y_1, y_2, y_3\rangle = \langle\langle y_1, y_2\rangle, y_3\rangle$, and $\langle y_1, \cdots, y_{l-1}, y_l\rangle = \langle\langle y_1, \cdots, y_{l-1}\rangle, y_l\rangle$, where $\mathbb{N}^*$ denotes the set of nonnegative integers.

We define the size function $h(u) = \langle n, k, b_1, \cdots, b_m\rangle$, and the input function $g(u) = x_{1,1}^{w_{11}} x_{2,1}^{w_{21}} \cdots x_{m,1}^{w_{m1}} x_{m+1,1}^{c_1} \cdots x_{1,n}^{w_{1n}} x_{2,n}^{w_{2n}} \cdots x_{m,n}^{w_{mn}} x_{m+1,n}^{c_n}$, where the first subscript $i$ of $x_{i,j}$ denotes the $i$th inequality, the second subscript $j$ of $x_{i,j}$ corresponds to the variable $x_j$.

For each $\langle n, k, b_1, \cdots, b_m\rangle$, we consider the recognizer P system $(\Pi(\langle n, k, b_1, \cdots, b_m\rangle), \Sigma(\langle n, k, b_1, \cdots, b_m\rangle), i_0)$, where $\Sigma(\langle n, k, b_1, \cdots, b_m\rangle) = \{x_{i,j} \mid 1 \leq i \leq m + 1, 1 \leq j \leq n\}$, $i(\langle n, k, b_1, \cdots, b_m\rangle) = 2$ and $\Pi(\langle n, k, b_1, \cdots, b_m\rangle) = (\Gamma(n, k, b_1, \cdots, b_m), \{1, 2\}, [_1[_2\ ]_2]_1, w_1, w_2, R)$, $\Gamma(n, k, b_1, \cdots, b_m)$ is defined as follows:

$$\begin{aligned}
\Gamma(m, n) &= \Sigma(\langle n, k, b_1, \cdots, b_m\rangle) \cup \{d_i \mid 1 \leq i \leq 3n + 2\sum_{i=1}^{m} b_i + 3m + 2k + 1\} \\
&\cup \ \{r_{i,j}, s_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq 2n\} \cup \{a_{i,j}, t_{i,j} \mid 1 \leq i \leq m, 0 \leq j \leq 2n\} \\
&\cup \ \{q_{i,j} \mid 0 \leq i \leq 2\max\{b_1, \cdots, b_m\} + 1, 1 \leq j \leq m\} \\
&\cup \ \{q_{i,m+1} \mid 0 \leq i \leq 2k + 1\} \cup \{d_+, d_-, e_0, \lambda, \mathtt{yes}, \mathtt{no}\}.
\end{aligned}$$

The initial content of each membrane is: $w_1 = \emptyset$ and $w_2 = d_1 a_{1,0}^{b_1} a_{2,0}^{b_2} \cdots a_{m,0}^{b_m} t_{m+1,0}^{k}$. The set of rules, $R$, is given by (we also give explanation about the use of these rules during the computations):

1. $[_2 d_i]_2^0 \rightarrow [_2 d_i]_2^+ [_2 d_i]_2^-$, $1 \leq i \leq n$.
   By using a rule of (1), a membrane with label 2 is divided into two membranes with the same label, but with different polarizations. These rules allow us to have exponential workspace in linear time.

2. $[_2 x_{i,1} \rightarrow r_{i,1}]_2^+$, $1 \leq i \leq m$.
   $[_2 x_{m+1,1} \rightarrow s_{m+1,1}]_2^+$.
   $[_2 x_{i,1} \rightarrow \lambda]_2^-$, $1 \leq i \leq m+1$.
   The rules of (2) try to implement a process allowing membranes with label 2 to encode the assignment of the variable $x_1$, in such a way that if the variable $x_1$ takes value 1, then the objects $x_{i,1}$ $(1 \leq i \leq m)$ evolve to objects $r_{i,1}$, and the objects $x_{m+1,1}$ evolve to objects $s_{m+1,1}$, in the corresponding membranes with label 2 and positive charge; otherwise, the objects $x_{i,1}$ will disappear in the corresponding membranes with label 2 and negative charge.

3. $[_2 x_{i,j} \rightarrow x_{i,j-1}]_2^+$, $1 \leq i \leq m+1$, $2 \leq j \leq n$.
   $[_2 x_{i,j} \rightarrow x_{i,j-1}]_2^-$, $1 \leq i \leq m+1$, $2 \leq j \leq n$.
   The evolving process described previously is always made with respect to the variable $x_1$. Hence, the rules of (3) take charge of making a cyclic path through all the variables to get that, initially, the first variable is $x_1$, then $x_2$, and so on.

4. $[_2 d_i]_2^+ \rightarrow [_2\ ]_2^0 d_i$, $1 \leq i \leq n$.
   $[_2 d_i]_2^- \rightarrow [_2\ ]_2^0 d_i$, $1 \leq i \leq n$.
   $d_i [_2\ ]_2^0 \rightarrow [_2 d_{i+1}]_2^0$, $1 \leq i \leq n-1$.
   The rules of (4) are used as controllers of the generating process of the assignments of all variables and the encoding of the coefficients: the objects $d_i$ are sent to the membrane with label 1 at the same time the assignments are made, and they come back to the membranes with label 2 to start the division of these membranes.

5. $[_2 r_{i,j} \rightarrow r_{i,j+1}]_2^0$, $1 \leq i \leq m$, $1 \leq j \leq 2n-1$.
   $[_2 s_{m+1,j} \rightarrow s_{m+1,j+1}]_2^0$, $1 \leq j \leq 2n-1$.
   $[_2 a_{i,j} \rightarrow a_{i,j+1}]_2^0$, $1 \leq i \leq m$, $0 \leq j \leq 2n-1$.
   $[_2 t_{m+1,j} \rightarrow t_{m+1,j+1}]_2^0$, $0 \leq j \leq 2n-1$.
   The use of objects $r_{1,2n}, a_{1,2n}, s_{m+1,2n}, t_{m+1,2n}$ in the rules (8), (11) and (14) makes necessary to perform a rotation of these objects. This is the mission of the rules of (5).

6. $[_1 d_i \rightarrow d_{i+1}]_1^0$, $n \leq i \leq 3n-3$.
   $[_1 d_{3n-2} \rightarrow d_{3n-1} e_0]_1^0$.
   Through the counter-objects $d_i$, the rules of (6) control the rotation of the objects $r_{i,j}$, $s_{m+1,j}$, $a_{i,j}$ and $t_{m+1,j}$ in the membranes with label 2, so that their second subscripts are unified to $2n$.

7. $e_0 [_2]_2^0 \rightarrow [_2 q_{0,1}]_2^-$.
   $[_1 d_{3n-1} \rightarrow d_{3n}]_1^0$.
   The application of the rules of (7) will show that the system is ready to check whether the constraint inequalities are satisfied by the assignment of variables encoded by an internal membrane.

8. $[_2 r_{1,2n}]_2^- \rightarrow [_2\ ]_2^0 \lambda$.
   $[_2 a_{1,2n}]_2^0 \rightarrow [_2\ ]_2^- \lambda$.
   These rules implement the comparison (that is, they check whether the constraint inequality holds or not). They work as a loop that erases objects $r_{1,2n}$ and $a_{1,2n}$ one by one alternatively, changing the charge of the membrane in each step. We

will see that if the checking had an affirmative result, then the membrane would get positively charged, and the checking of the next inequality will be activated.

9. $[_2 q_{2j,i} \to q_{2j+1,i}]_2^-$, for $j = 0, \cdots, \max\{b_1, \cdots, b_m\}$, $1 \le i \le m$.
$[_2 q_{2j+1,i} \to q_{2j+2,i}]_2^0$, for $j = 0, \cdots, \max\{b_1, \cdots, b_m\} - 1$, $1 \le i \le m$.
A counter that controls the previous loop is described here. The subscript of $q_{j,i}$ and the electric charge of the membrane give enough information to point out if the number of objects $r_{i,2n}$ is not greater than (less than or equal to) the number of objects $a_{i,2n}$.

10. $[_2 q_{2j+1,i}]_2^- \to [_2\ ]_2^+ q_{0,i+1}$, for $j = 0, \cdots, \max\{b_1, \cdots, b_m\}$, $1 \le i \le m$.
If an assignment verifies the $i$th constraint inequality, then inside the corresponding membrane that encodes it there will not be more objects $r_{i,2n}$ than $a_{i,2n}$. This forces the loop described in (9) to stop: the moment will come when there are no objects $r_{i,2n}$ left, and then the rule $[_2 q_{2j,i} \to q_{2j+1,i}]_2^-$ will be applied but it will not be possible to apply the rule $[_2 r_{1,2n}]_2^- \to [_2\ ]_2^0 \#$ at the same time. Thus, an object $q_{2w(i)+1}$ will be present in the membrane with negative charge, where $w(i) = \sum_{j=1}^{n} w_{ij}$, so the rule (10) will be applied.

11. $[_2 r_{i,2n} \to r_{i-1,2n}]_2^+$, for $2 \le i \le m$.
$[_2 a_{i,2n} \to a_{i-1,2n}]_2^+$, for $2 \le i \le m$.
$[_2 s_{i,2n} \to s_{i-1,2n}]_2^+$, for $2 \le i \le m+1$.
$[_2 t_{i,2n} \to t_{i-1,2n}]_2^+$, for $2 \le i \le m+1$.
The comparison process described in the rules of (8) is always made with respect to the first constraint inequality. Hence the rules of (11) take charge of making a cyclic path through all the constraint inequality. (for the objective function inequality, we have different comparison process, as shown later in rules of (14), (15) and (16).)

12. $q_{0,i+1}[_2\ ]_2^+ \to [_2 q_{0,i+1}]_2^-$, for $1 \le i \le m-1$.
The membranes with positive charge mean that the assignments encoded by them have verify the first $i$ constraint inequality. The objects $q_{0,i+1}$ enter the membranes with the label 2 and positive charge, changing the charge to negative to allow the checking for next inequality to begin (using rules of types (8), (9), (10)).

13. $q_{0,m+1}[_2\ ]_2^+ \to [_2 q_{0,m+1}]_2^+$.
If there is at least one assignment which verify all the constraint inequalities, then objects $q_{0,m+1}$ appear in the skin membrane. In the next step, by the rule of (13), these objects enter the membranes with label 2 and positive charge. At the same time, the objects $s_{1,2n}$ and $t_{1,2n}$ appear in the membranes encoding assignments which verify all the constraint inequalities (using rules of (11)). Now the system is ready for checking the objective function inequality.

14. $[_2 s_{1,2n}]_2^+ \to [_2\ ]_2^0 \lambda$.
$[_2 t_{1,2n}]_2^0 \to [_2\ ]_2^+ \lambda$.
The checking loop is designed for objective function as rules of (8) for constraint inequalities, but the electric charges involved are now neutral and positive.

15. $[_2 q_{2j,m+1} \to q_{2j+1,m+1}]_2^+$, for $j = 0, \cdots, k$.
$[_2 q_{2j+1,m+1} \to q_{2j+2,m+1}]_2^0$, for $j = 0, \cdots, k-1$.
The counter $q_{j,m+1}$ controls the previous loop described in the rules of (14).

16. $[_2 q_{2k+1,m+1}]_2^+ \rightarrow [_2\ ]_2^0 \texttt{yes}$.
$[_2 q_{2k+1,m+1}]_2^0 \rightarrow [_2\ ]_2^0 \texttt{yes}$.
If an assignment verifies the objective function inequality, then in the membrane encoding this assignment there are not less objects $s_{1,2n}$ than objects $t_{1,2n}$. This causes the rules from (15) to apply $k$ times each, and after that the first subscript of $q_{j,2n}$ will be $j = 2k$. Then the rule $[_2 q_{2j,m+1} \rightarrow q_{2j+1,m+1}]_2^+$ will be applied (possibly together with $[_2 s_{1,2n}]_2^+ \rightarrow [_2\ ]_2^0 \lambda$) and one of the rules from (16) will produce the object `yes` reporting that this assignment verifies the objective function inequality.

17. $[_1 d_i \rightarrow d_{i+1}]_1^0$, for $i = 3n, \cdots, 3n + 2\sum_{i=1}^{m} b_i + 3m + 2k + 1$.
$[_1 d_l \rightarrow d_+ d_-]_1^0$, where $l = 3n + 2\sum_{i=1}^{m} b_i + 3m + 2k + 1$.
Before the answer is sent out to the environment, all of the membrane should have either ended their checking stage successfully (checking the constraint inequalities and objective function inequality) or got to a blocking state otherwise. The counter $d_j$ gives enough time to deal with the worst case. The condition for worst case (i.e. the case that the checking stage will last longer) is that the assignment with all variables taking value 1 is a solution (i.e. it verifies the constraint inequalities and the objective function inequality). The number of computational steps will be maximum if $\sum_{j=1}^{n} w_{ij} = b_i$, for $i = 1, \cdots, n$.

18. $[_1 d_+]_1^0 \rightarrow [_1\ ]_1^+ d_+$.
$[_1 d_- \rightarrow \texttt{no}]_1^+$.
$[_1 \texttt{yes}]_1^+ \rightarrow [_1\ ]_1^0 \texttt{yes}$.
$[_1 \texttt{no}]_1^+ \rightarrow [_1\ ]_1^0 \texttt{no}$.
The output process is activated. The skin membrane needs to be positively charged before the answer is send out to the environment. Object $d_+$ takes charge of this, and, then, if the answer is affirmative, an object `yes` will be sent out changing the charge of the skin to neutral. Otherwise, an object `no` will be sent out changing the charge of the skin to neutral.

From the previous explanation of the use of rules, one can easily see how this P system works. It is easy to prove that the designed P system is deterministic, confluent, and sound.

The family is polynomially uniform by Turing machines. It can be observed that the above description of the evolution rules is computable in an uniform way, in particular from the constants $n$, $m$, $k$, and $b_i$, $i = 1, \cdots, m$.

Now, we prove that the family $\mathbf{\Pi} = (\Pi(t))_{t \in \mathbb{N}}$ solves the MKP in linear time.

The computational process of the designed P system with input $g(u)$ can be structured in four stages: a stage of *generation* of all assignments of variables; a stage of *synchronization*; a stage of *checking* whether there is an assignment which satisfies the constraint inequalities and the objective function inequality; and a stage of *output*.

The *generation* is controlled by the objects $d_i$, with $1 \le i \le n$.

– The presence in the skin of one object $d_i$, with $1 \le i \le n$, will show that all possible partial assignments associated with $\{x_1, \cdots, x_i\}$ have been generated.

- In this stage, we simultaneously encode in every internal membrane the coefficient from the constraint inequalities and the objective function inequality (through the objects $r_{i,j}$ and $\lambda$)
- The object $d_1$ appears in the skin after 2 steps of the computation. From the appearance of $d_i$ in the skin to the appearance of $d_{i+1}$, with $1 \leq i \leq n-1$, 3 steps have been executed.
- This stage ends when the object $d_n$ appears in the skin.

Hence, the total number of steps in the generation stage is $3n-1$.

The *synchronization* stage has the goal of unifying the second subscripts of the objects $r_{i,j}$ , to make them equal to $2n$.

- This stage starts with the evolution of the object $d_n$ in the skin.
- In every step of this stage the object $d_i$, with $n \leq i \leq 3n-1$, in the skin evolves to $d_{i+1}$.
- In every step of this stage, the second subscripts of objects $r_{i,j}$ will increase.
- This stage ends as soon as the object $d_{3n}$ appears in the skin, that is the moment when each membrane with label 2 has negative charge and contains one object $q_{0,1}$ (by using the first rule of (7)).

Therefore, the synchronization stage needs a total of $2n$ steps.

The *checking* stage has the goal to determine whether there is an assignment which satisfy the constraint inequalities and the objective function inequality. This stage is controlled by the objects $d_i$, where $3n \leq i \leq 2\sum_{j=1}^{m} b_j + 3m + 2k + 1$.

- The presence of an object $q_{0,j}$ with $1 \leq j \leq m+1$ in a membrane with label 2 shows that the first $j-1$ inequalities are satisfied by the assignment represented by such membrane. The presence of object `yes` in the skin membrane shows that all inequalities are satisfied by an assignment.
- From every $q_{0,j}$ with $1 \leq i \leq m$, the object $q_{0,j+1}$ is obtained in some membranes after at most $2b_j + 3$ steps. From every $q_{0,m+1}$, the object `yes` is obtained in skin membrane after at most $2k+2$ steps.
- The checking stage ends as soon as the objects $d_+, d_-$ appear in the skin. The checking of inequalities finishes before or at this moment.

Therefore, the total number of steps of this stage is $2\sum_{j=1}^{m} b_j + 3m + 2k + 2$.

The *output* stage starts immediately after the appearance of the objects $d_+, d_-$ in the skin.

- If there is an assignment which satisfies the constraint inequalities and objective function inequality, then after 2 steps the system sends `yes` to the environment.
- If there is no assignment which satisfies the constraint inequalities and objective function inequality, then after 3 steps the system sends `no` to the environment.

Therefore, the total number of steps in the output stage is at most 3.

Let us see that the family $\mathbf{\Pi}$ is *linearly bounded.* For that, it is enough to note that the time of the stages of the execution of $\Pi(h(G))$ with input $g(u)$ is: (a) generation stage, $3n-1$ steps; (b) synchronization stage, $2n$ steps; (c) checking stage, $2\sum_{j=1}^{m} b_j + 3m + 2k + 2$ steps; and (d) output stage, at most 3 steps. Hence, the total execution time of $\Pi(h(G))$ with input $g(u)$ is $5n + 2\sum_{j=1}^{m} b_j + 3m + 2k + 4 \in O(n, m, k, b_1, \cdots, b_m)$.

From all the above we have the following result.

**Theorem 3.1** *The MKP can be solved in linear time by uniform recognizer P systems with active membranes using 2-division.*

## 4 Conclusions

We have shown that MKP can be solved in linear time by uniform recognizer P systems with active membranes using 2-division, in this sense: all instance of the problem that have the *same size* are processed by the same P system (on which an appropriate input, that depends on the concrete instance, is supplied). In our algorithm, the definition of the size function contains $d_1, d_2, \cdots, d_m$. It is a natural question to define a size function without $d_1, d_2, \cdots, d_m$ (i.e., the size function does not depend on $d_1, d_2, \cdots, d_m$, in this sense the P systems will be "more uniform".) Under this definition of size function, it is open how to design P systems to solve MKP.

The general 0–1 integer programming problem has the same form with the MKP shown in subsection 3.1, the only difference is that the constraint inequalities of general 0–1 integer programming problem involve both positive and negative integer coefficients. The MKP can be considered as a general 0–1 integer programming problem with nonnegative coefficients. It is not difficult to modify our algorithm (design similar comparison process) to solve the general 0–1 integer programming problem, because we can write this problem in the following form: given an integer $k$, an objective function $f(x_1, \cdots, x_n) = \sum_{j=1}^{n} c_j x_j$, and constraints $\sum_{j=1}^{n} w_{ij} x_j \leq \sum_{j=1}^{n} u_{ij} x_j$, for $i = 1, \cdots, m$, $x_j \in \{0, 1\}$, for $j = 1, \cdots, n$, where $c_j$, $w_{i,j}$ and $u_{i,j}$ are nonnegative integers, decide whether or not there exists an assignment of variables $x_j$ such that it satisfies the constraints and the objective function is greater than or equal to $k$.

Let us denote by $\mathcal{AM}$ the class of recognizer P systems with active membranes using 2-division. Then from Theorem 3.1, we have MKP $\in \mathbf{PMC}_{\mathcal{AM}}$. Because the class $\mathbf{PMC}_{\mathcal{AM}}$ is stable under polynomial time reduction, we have $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$. It remains open whether or not the inclusion is strict or not.

**Acknowledgements.** Many thanks to Agustín Riscos-Núñez for his comments and corrections. The first author (he is also the corresponding author) is supported by grant DGU-SB2001-0092 from Spanish Ministry for Education, Culture, and Sport, National Natural Science Foundation of China (Grant No. 60373089).

## References

[1] A. Alhazov, C. Martín-Vide, L. Pan, Solving a **PSPACE**-Complete Problem by P Systems with Restricted Active Membranes, *Fundamenta Informaticae*, 58(2)(2003), 66–77.

[2] M.R. Garey, D.J. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman, San Francisco, 1979.

[3] B. Gavish and H. Pirkul, Allocation of Data Bases and Processors in a Distributed Computing System, *Managment of Distributed Data Processing*, 31 (1982), 215–231.

[4] Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and TUCS Research Report 208, 1998 (http://www.tucs.fi).

[5] Gh. Păun, P Systems with Active Membranes: Attacking NP-Complete Problems, *Journal of Automata, Languages and Combinatorics*, 6, 1 (2001), 75–90.

[6] Gh. Păun, *Membrane Computing: An Introduction*, Springer, Berlin, 2002.

[7] M.J. Pérez-Jiménez, A. Romero Jiménez, F. Sancho-Caparrini, Complexity Classes in Models of Cellular Computing with Membranes, *Natural Computing*, 2, 3 (2003), 265–285.

[8] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, A Polynomial Complexity Class in P systems Using Membrane Division, in *the 5th Workshop on Descriptional Complexity of Formal Systems*, Budapest, Hunagary, July 12–14, 2003.

[9] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, Solving Validity Problem by Active Membranes with Input, in *Proceedings of the Brainstorming Week on Membrane Computing* (M. Cavaliere, C. Martín-Vide, Gh. Păun, Eds.), Report GRLMC 26/03, 2003, 270–278.

[10] M.J. Pérez-Jiménez, A. Riscos-Núñez,: A Linear-time Solution to the Knapsack Problem Using Active Membranes, in *Preproceedings of the Workshop on Membrane Computing* (A. Alhazov, C. Martín-Vide, Gh. Păun, Eds.), Tarragona, Spain, July 17-22, 2003.

[11] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, *Teoría de la Complejidad en Modelos de Computatión Celular con Membranas*, Editorial Kronos, Sevilla, 2002.

[12] W. Shih, A Branch & Bound Method for the Multiconstraint Zero-one Knapsack Problem, *Journal of the Operational Research Society*, 30(1979), 369–378.

[13] P. Sosík, Solving a **PSPACE**-Complete Problem by P Systems with Active Membranes, *Proceedings of the Brainstorming Week on Membrane Computing* (M. Cavaliere, C. Martín-Vide, Gh. Păun, Eds.), Report GRLMC 26/03, 2003, 305–312.