ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **12**, Number 2, 2009, 157–173

New Strategies of Using the Rules of a P System in a Maximal Way: Power and Complexity

Gabriel CIOBANU[1], Solomon MARCUS[2], Gheorghe PĂUN[2]

[1] Institute of Computer Science, Romanian Academy
and *A. I. Cuza* University of Iaşi
Blvd. Carol I no.8, 700505 Iaşi, Romania
E-mail: gabriel@iit.tuiasi.ro

[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
E-mail: solomon.marcus@imar.ro, gpaun@us.es

Abstract. We examine the computing power and complexity of P systems which use two strategies of applying the rules. One strategy is to maximize the number of objects used in rules and the other is to maximize the number of rules applied in each membrane. For P systems with cooperative multiset rewriting rules, P systems with active membranes, and P systems with symport/antiport rules we prove the computational universality for both these types of parallelism. The computational complexity of the maximum consuming systems is studied for systems with cooperative rules of two types, by using two known combinatorial **NP**-complete problems, namely the knapsack problem and the integer linear programming.

1. Introduction

Membrane computing is a branch of natural computing, the new field of computer science dealing with models and paradigms inspired by the way nature "computes", where computation is mainly given by bio-information processing. Membrane computing abstracts computing models, called P systems, from the structure and the functioning of the living cell and from the way the cells cooperate in multi-cell structures – see [8], [9], [11]. In this way, P systems combine the power of distributed parallel rewriting systems with the power of contextual evolution to achieve computational universality. From a computational point of view, several results show that

the number of membranes sufficient for obtaining the power of a Turing machine is pretty small.

The cell is the main inspiration of membrane computing. Inside the cell, several membranes define compartments where specific biochemical processes take place. In particular, a membrane encloses the nucleus where the genetic material is placed. Through vesicles enclosed by membranes, several molecules can be transported from a compartment of the cell (e.g., from the Golgi apparatus) to other compartments of the cell. Each compartment contains substances (ions, small molecules, macromolecules) and specific reactions. The substances are abstracted by multisets of objects, and the reactions by rules of form $u \to v$, where u and v are multisets of objects. (The multisets are represented by strings, with the convention that all permutations of a string represent the same multiset.) When such a system is evolving, the objects and the rules are chosen in a nondeterministic manner, and the rules are applied in parallel. There are two basic types of rules: multiset-rewriting rules (evolution rules), and communication rules (symport/antiport rules).

The most investigated way of using the rules in a P system is the maximal parallelism: in each membrane a multiset of rules is chosen which can be applied to the objects from that membrane and is maximal in the sense of inclusion, i.e., no further rule can be added such that the enlarged multiset is still applicable. In case of rules which compete for objects from adjacent membranes (as in P systems with symport/antiport rules) or for membranes (as in P systems with active membranes), specific variations are necessary to this basic idea. Another natural idea is to apply the rules in such a way to have a maximal number of objects evolved in each membrane. Recently, this strategy was explicitly considered in [2], where it is proved that, for P systems with rules of the form $a^n \to u$, where a is an object and u a multiset, the problem of finding the rules which consume a maximal number of objects is **NP**-complete.

In this paper, we consider the maximum consuming strategies, defining the transitions in a P system in such a way to maximize the number of objects evolved in each membrane, as well as the related case of applying a multiset of rules of a maximal cardinality. First we investigate the computing power of three basic classes of P systems for these strategies. Specifically, we consider P systems with multiset rewriting rules, with symport/antiport rules, and with active membranes. In all cases (with cooperative rules in the first one) we prove the universality of the considered systems. (The proofs are rather standard in this area: simulations of register machines.) All these results need improvements. For instance, we know that for the usual maximal parallelism (in the multiset inclusion sense), also P systems with catalytic rules are universal [4], and a similar result is expected also for the two types of maximal parallelism considered here. Then, for symport/antiport P systems, our proof uses antiport rules of weight 2 (two objects together are moved inside or outside a membrane), but, for the "standard maximal parallelism", also systems with symport/antiport rules of weight 1 or with symport rules of weight 2 are universal, [1]; similar results remains to be proved also for the new forms of maximality. In turn, in the case of P systems with active membranes, we use here (two) polarizations, but it is known that universality results can be obtained also for non-polarized membranes in the case of usual

maximal parallelism; this topic remains to be considered also for the new maximality modes.

2. Types of Maximal Parallelism

The reader is supposed to be familiar with basic elements of membrane computing presented in [9]. However we will briefly introduce in the next sections the P systems we work with; further details can be found in the cited papers and at the P systems website [11].

In this section we introduce some terminology concerning the variants of maximal parallelism we consider here, starting from multiset rewriting rules. Take an alphabet O of objects and a set R_i of rules, those associated with a region i where a multiset $w_i \in O^*$ of objects is present. In the standard maximally parallel way of using the rules (we denote this case with *maxP*), the objects of w_i are non-deterministically assigned to rules in R_i until no further object can be associated with a rule, and those objects are evolved which were assigned (clearly, the objects evolve through the respective rules with which they were associated). Otherwise stated, we choose a multiset of rules in R_i (we choose a subset $R'_i \subseteq R_i$ and we associate multiplicities to rules in R'_i); this multiset (hence, having the support R'_i) is applied only if it has the following two properties: (1) its rules can be applied to the objects in w_i, and (2) no rule can be added to R'_i and no multiplicity of rules in R'_i can be increased so that the obtained multiset is still applicable to w_i. A simple example will be discussed immediately. Thus, the applied multiset of rules should be maximal, in the sense of inclusion, among the multisets of rules applicable to the objects in w_i. This maximality does not ensure that the number of evolved objects or the number of used rules are maximal.

Indeed, let us assume that $w_i = abcde$ and that R_i consists of the following rules (the multisets $u_1 - u_4$ in the right hand of rules are not relevant):

$$r_1 : ab \to u_1,\ r_2 : c \to u_2,\ r_3 : bc \to u_3,\ r_4 : acde \to u_4.$$

In the table below we indicate all multisets (in this case, the multiplicity of used rules is always 1) of applicable rules, together with the number of objects evolved and the number of rules used:

Case	Rules used	No. obj.	No. rules	Types of maximality
1	r_1	2	1	none
2	r_2	1	1	none
3	r_3	2	1	maxP
4	r_4	4	1	maxO, maxP
5	r_1, r_2	3	2	maxR, maxP

No rule can be added in cases 3, 4, 5, hence for them we have found a multiset of rules which can be applied in the *maxP* mode.

Note that the number of objects is bigger in case 4 than in all others and that the number of rules is the biggest in case 5. This suggests defining two new strategies of

using the rules: in such a way to maximize the number of evolving objects (we denote it with *maxO*) and in such a way to apply a maximal number of rules (denoted by *maxR*). Of course, if several multisets of rules lead to the same maximal number of objects or of rules, then one multiset is non-deterministically chosen, as usual also for *maxP*. The previous table contains such cases – and it is important to note that they do not coincide, but, clearly, a transition which is of type *maxO* or *maxR* is also of type *maxP* (adding a rule to the multiset would increase the number of evolved objects and, trivially, of applied rules).

For non-cooperative rules, the three types of maximal parallelism coincide. This is not the case for the catalytic rules: for $w_i = cab$ (c is the catalyst) and

$$R_i = \{r_1 : ca \to u_1,\ r_2 : cb \to u_2,\ r_3 : a \to u_3\},$$

both r_1 and r_2r_3 are *maxP*, but only the second multiset is also *maxO* and *maxR*.

The previous example can be transferred to a symport/antiport system: assume that c is outside membrane i, ab are inside, and the rules

$$r_1 : (c, in; a, out),\ r_2 : (c, in; b, out),\ r_3 : (a, out)$$

are associated with membrane i. As above, both r_1 and r_2r_3 are *maxP*, but only r_2r_3 is also *maxO* and *maxR*. Using symport or antiport rules with more objects in their multisets (of a higher weight) will lead to still more interesting cases, like in the first example considered above.

A similar situation appears for rules with active membranes: consider a membrane i with objects ab inside and subject to the following rules

$$r_1 : [\ a\ ]_i \to [\quad]_i a,\ r_2 : [\ b\ ]_i \to [\quad]_i b,\ r_3 : [\ a \to u\ ]_i.$$

Like in the symport/antiport case, both r_1 and r_2r_3 are *maxP*, but only r_2r_3 is also *maxO* and *maxR*.

The fact that the three modes of maximal parallelism are different for the three types of P systems mentioned above makes interesting the study of the computing power of these systems for the new types of maximality, *maxO* and *maxR*, and this will be done in the next section.

3. Computational Power

3.1. Universality for Multiset Rewriting Rules

For the sake of readability, we start by recalling the formal notations for (cell-like) P systems with multiset rewriting rules. Such a system is a construct $\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_o)$, where O is the alphabet of objects, μ is the membrane structure (with m membranes), $w_1, \ldots, w_m$ are (strings over O representing) multisets of objects present in the m regions of μ at the beginning of a computation, $R_1, \ldots, R_m$ are finite sets of evolution rules, associated with the regions of μ, and i_o is the label of a membrane, used as the output membrane. The general form

of rules is $u \to v$, where $u \in O^+$ and $v \in (O \cup (O \times \{out, in\})^*$; when applying such a rule, objects of u are "consumed" and those of v are "produced" and immediately transported to the region indicated by the targets *out, in*, i.e., outside the membrane where the rule was used in the case of *out*, or to any of its directly inner membranes, non-deterministically chosen, in the case of *in*. We denote by $N_{maxX}(\Pi)$ the set of numbers computed by Π in the *maxX* mode, with $X \in \{P, O, R\}$, and by $NOP_m^{maxX}(\textit{type-of-rules})$ the family of sets $N_{maxX}(\Pi)$ computed by systems Π with at most m membranes and rules as indicated by *type-of-rules*; as types of types we consider here only *coo*, indicating cooperative rules.

It is known that $NOP_1^{maxP}(coo) = NRE$, where NRE is the family of Turing computable sets of numbers, and a similar result is true for catalytic rules. Such a result is also true for *maxO* and *maxR*. All proofs are based on simulating register machines; as it is known (see, e.g., [6]), the register machines (even with a small number of registers, three) compute all sets of numbers which are Turing computable, hence they characterize NRE. A register machine M computes a number n in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach the halt instruction, then the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$.

Theorem 3.1. *$NOP_1^{maxX}(coo) = NRE$, for all $X \in \{P, O, R\}$.*

Proof. The case of $X = P$ is known, we have mentioned it only for the sake of uniformity.

Consider a register machine $M = (m, H, l_0, l_h, I)$ as above (the number of registers is not relevant), and construct the P system

$$\Pi = (O, [\quad]_1, l_0, R_1, 1),$$

with the alphabet

$$O = \{a_r \mid 1 \leq r \leq m\} \cup \{l, l', l'', l''', l^{iv} \mid l \in H\} \cup \{\#\}$$

and the set of rules obtained as follows (the value of register r is represented in Π by the number of copies of object a_r present in the unique membrane of Π):

1. For each ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ of M we introduce in R_1 the rules

$$l_i \to l_j a_r,$$
$$l_i \to l_k a_r.$$

This is an obvious simulation of the ADD instruction.

2. For each SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ of M we introduce in R_1 the rules

$$\begin{aligned} &l_i \to l_i' l_i'', \\ &l_i' a_r \to l_i''', \\ &l_i'' \to l_i^{iv}, \\ &l_i^{iv} l_i''' \to l_j, \\ &l_i^{iv} l_i' \to l_k. \end{aligned}$$

If any copy of a_r is present, the "checker" l_i^{iv} finds in the membrane the object l_i''' and introduces the label l_j, otherwise the label l_k is introduced, which means that the SUB instruction is correctly simulated.
Note that in both cases, the three maximality modes coincide.

3. In order to correctly simulate a computation of M, we have to remove all objects different from a_1. In the case of *maxR*, this can be done by means of the following rules (also working for the *maxP* case):

$$\begin{aligned} &l_h a_r \to l_h, \ r \in \{2, 3, \ldots, m\}, \\ &l_h \to l_h' l_h'', \\ &l_h' a_r \to \#, \ r \in \{2, 3, \ldots, m\}, \\ &\# \to \#, \\ &l_h'' \to l_h''', \\ &l_h''' l_h' \to \lambda. \end{aligned}$$

After removing objects $a_r, r \neq 1$ (one rule is used in each step), one passes to checking whether any object $a_r, r \neq 1$, is still present and only in the negative case one also remove the primed versions of l_h.
In the case of *maxO*, only the first rule above, together with

$$l_h \to \lambda$$

are enough: this last rule cannot be used as long as any rule $l_h a_r \to l_h, r \neq 1$, can be used.

Of course, this last group of rules can be avoided if we start from a register machine which ends the computations with all registers empty excepting the first one (then, the rule $l_h \to \lambda$ is sufficient in all cases), but we wanted to make an explicit use of the restriction imposed by the *maxO* mode in ensuring the correct simulation of the register machine. □

3.2. Universality for Symport/Antiport Rules

A symport/antiport P system is a construct $\Pi = (O, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, i_o)$, where O, μ, $w_1, \ldots, w_m$, and i_o are as above, $E \subseteq O$ is the set of objects appearing in the environment, in arbitrarily many copies, and $R_1, \ldots, R_m$ are finite

sets of symport/antiport rules associated with the membranes of μ. The rules are of the forms (u, in) or (u, out) (symport rules) and $(u, out; v, in)$ (antiport rules), where $u, v \in O^+$. Using such a rule from R_i means moving across membrane i the objects specified by u and v, in the indicated directions. The maximal length of u, v in all rules of the system defines the *weight* of the system. We denote by $N_{maxX}(\Pi)$ the set of numbers computed by Π in the *maxX* mode, with $X \in \{P, O, R\}$, and by $NOP_m^{maxX}(sym_p, anti_q)$ the family of sets $N_{maxX}(\Pi)$ computed by systems Π with at most m membranes, symport rules of weight at most p, and antiport rules of weight at most q. It is known that $NOP_3^{mP}(sym_1, anti_1) = NOP_3^{mP}(sym_2, anti_0) = NRE$. A similar result is also true for *maxO* and *maxR*.

Theorem 3.2. *$NOP_1^{maxX}(sym_1, anti_2) = NRE$, for all $X \in \{P, O, R\}$.*

Proof. Consider a register machine $M = (m, H, l_0, l_h, I)$ as above, this time making the assumption (which does not restrict the generality) that in the end of the computations only the first register may be non-empty. We construct the P system

$$\Pi = (O, [\quad]_1, l_0, O, R_1, 1),$$

with the alphabet

$$O = \{a_r \mid 1 \leq r \leq m\} \cup \{l, l', l'', l''', l^{iv} \mid l \in H\}$$

and the set of rules obtained as follows:

1. For each ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ of M we introduce in R_1 the rules

$$(l_i, out; l_j a_r, in),$$
$$(l_i, out; l_k a_r, in).$$

2. For each SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ of M we introduce in R_1 the rules

$$(l_i, out; l_i' l_i'', in),$$
$$(l_i' a_r, out; l_i''', in),$$
$$(l_i'', out; l_i^{iv}, in),$$
$$(l_i^{iv} l_i''', out; l_j, in),$$
$$(l_i^{iv} l_i', out; l_k, in).$$

   Like in the previous proof, the maximal parallelism of any type ensures the correct simulation of the instructions of M.

3. In order to finish the simulation we just add the rule

$$(l_h, out).$$

The three maximality modes *maxX*, $X \in \{P, O, R\}$, coincide, and this concludes the proof. □

3.3. Universality for Rules with Active Membranes

The definition of a P system with active membranes is the same as for systems with multiset rewriting rules, but the membranes are explicit parts of rules. In what follows we only use three types of rules (evolution, send-in, and send-out), but we use two polarizations of membranes. Specifically, we use rules of the following forms:

(a) $[\ a \to u\ ]_i^e$, with $a \in O$, $u \in O^*$, and $e \in \{0, +\}$,

(b) $[\ a\ ]_i^e \to [\ \ ]_i^f b$, with $a, b \in O$ and $e, f \in \{0, +\}$,

(c) $a[\ \ ]_i^e \to [\ b\ ]_i^f$, with $a, b \in O$ and $e, f \in \{0, +\}$.

Note that rules of types (b) and (c) are symbol-to-symbol and that they can change the polarization of the membrane. When specifying a P system with active membranes, we have to define all components as in a system with multiset rewriting, with only one set of rules: the rules precisely identify the membranes where they are applied.

We denote by $NOP_m^{maxX}((a),(b),(c))$ the family of sets $N_{maxX}(\Pi)$ computed by systems Π with active membranes, using at most m membranes and rules of the three types defined above.

The counterpart of the previous results for the case of rules with active membranes leads to the next result, which is of some interest also for the *maxP* case, because the proof is done here starting from register machines, not from matrix grammars with appearance checking, as usual in the literature for this case.

Theorem 3.3. *$NOP_m^{maxX}((a),(b),(c)) = NRE$, for all $m \geq 3$ and $X \in \{P, O, R\}$.*

Proof. Consider a register machine $M = (m, H, l_0, l_h, I)$ as above, again with the assumption that in the end of the computations only the first register may be non-empty. Because three registers ensure the universality, we assume that $m = 3$. We construct the P system

$$\Pi = (O, B, [\ [\ \ ]_1^0 [\ \ ]_2^0\ ]_3^0, \lambda, \lambda, l_0, R, 1),$$

with

$$\begin{aligned} O &= \{a_r \mid r = 1, 2, 3\} \cup \{l, l', l'', l''', l^{iv}, l^v, l^{vi} \mid l \in H\}, \\ B &= \{1, 2, 3\}, \text{ with 3 being the skin membrane,} \end{aligned}$$

and the set of rules obtained as follows; with each register $r = 1, 2, 3$ of M we associate a membrane with label r and also an object a_r:

1. For each ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k), r = 1, 2$, of M we introduce in R the rules

$$\begin{aligned} &l_i[\ \ ]_r^0 \to [\ l_i'\ ]_r^0, \\ &[\ l_i' \to l_j a_r\ ]_i^0, \\ &[\ l_i' \to l_k a_r\ ]_i^0, \\ &[\ l_g\ ]_r \to [\ \ ]_r l_g,\ g \in \{j, k\}, \end{aligned}$$

and for $r = 3$ we consider the rules

$$[\ l_i \to l_g a_3\ ]_3^0,\ g \in \{j, k\}.$$

The simulation of the ADD instruction is obvious.

2. For each SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k), r = 1, 2$, of M we introduce in R the rules

$$\begin{aligned}
&l_i[\quad]_r^0 \to [\ l_i'\ ]_r^+,\\
&[\ a_r\ ]_r^+ \to [\quad]_r^0,\\
&[\ l_i' \to l_i''\ ]_r^+,\\
&[\ l_i''\ ]_r^0 \to [\quad]_r^0 l_j,\\
&[\ l_i''\ ]_r^+ \to [\quad]_r^0 l_k,
\end{aligned}$$

while for $r = 3$ we consider the following rules:

$$\begin{aligned}
1.\quad &[\ l_i \to l_i' l_i''\ ]_3^0,\\
2.\quad &[\ l_i' \to l_i'''\ ]_3^0,\\
&[\ l_i''\ ]_3^0 \to [\quad]_3^+ l_i^{iv},\\
3.\quad &[\ a_3\ ]_3^+ \to [\quad]_3^0 a_3,\\
&[\ l_i''' \to l_i^v\ ]_3^+,\\
4.\quad &[\ l_i^v \to l_j\ ]_3^0,\\
&[\ l_i^v\ ]_3^+ \to [\quad]_3^+ l_i^{vi},\\
5.\quad &l_i^{vi}[\quad]_3^+ \to [\ l_k\ ]_3^0.
\end{aligned}$$

The interplay of primed versions of l_i and the polarizations ensures now the correct simulation of the SUB instruction in all modes *max*X, $X \in \{O, P, R\}$.

The simulation of the SUB instructions for register 3 is slightly more difficult than for other registers: In the first step we introduce two objects, one for producing the "checker" l_i^v, and one for changing the polarization of the skin membrane. With the polarization changed to positive, an object a_3 can be removed (due to the maximal parallelism, it must be removed if it exists), and this is recorded in changing back the polarization to neutral. Now, the "checker" l_i^v introduces the correct object l_j or l_k, depending on the membrane polarization (returning this polarization to neutral in the case the register was empty – and this requires two steps, for sending out and bringing back primed version of l_i.

The three maximality modes *max*X, $X \in \{P, O, R\}$, coincide, and the end of a computation in M coincides with the halting of a computation in Π and $N(M) = N(Pi)$, which concludes the proof. □

The optimality of this result in what concerns the number of membranes and the use of polarizations remains to be investigated.

4. Computational Complexity

We consider two variants of membrane systems called simple P systems and maximum cooperative P systems, respectively. They evolve at each step by consuming the maximum number of objects. We study the problem of distributing objects to rules in order to achieve a maximum consuming and non-deterministic evolution of simple P systems. In a previous paper, [3], it is proved that (using the knapsack problem) the decision version of the resource mapping problem for simple P systems and (using the integer linear programming) the resource mapping problem for maximum cooperative P systems are **NP**-complete. Here we prove that the evolution of a simple P system can be simulated stage by stage by using a pseudo-polynomial algorithm for the knapsack problem, and similarly for maximum cooperative P systems.

4.1. Simple P Systems

First we consider a class of transition P systems called simple P systems, where the left side of the rules can contain only a single object (having various multiplicities) and the rules are applied in the *maxO* sense. We look at the "inner" complexity of the maximum consuming evolution of simple P systems. A resource allocator distributes the available objects to rules such that the evolution is then done in a maximum consuming way. A resource allocator can be associated to each membrane of a simple P system. Given the parallel evolution of membranes, each resource allocator resolves a particular instance of the knapsack problem associated with its membrane, and each one operates independently of the others. A resource allocator can be formally defined as a mapping $RA : O^* \to (O^*)^{|R|+1}$, where O is the alphabet of objects and R is the set of rules associated with a membrane. Considering $w \in O^*$ as the content of a membrane, $w_1, w_2, \ldots, w_{|R|}$ as the multisets allocated to rules (determined by their left sides), and w' as the multiset of objects that are not consumed, we have

$$RA(w) = \{(w_1, w_2, \ldots, w_{|R|}, w') \mid w_i \not\subseteq w', i = \overline{1, |R|} \text{ and } \sum_{i=1}^{|R|} w_i + w' = w\}. \quad (1)$$

Example 4.1. To clarify the previous definition, we give an example. Suppose that the resource allocator RA has to distribute the multiset a^{10} to the rules $a^4 \to b$, $a^3 \to c$, and $a^2 \to d$. According to our definition, the resources can be distributed in multiple ways. We show only a few:

$$a^{10} \Rightarrow 2 \cdot a^4 + 0 \cdot a^3 + 1 \cdot a^2$$
$$a^{10} \Rightarrow 1 \cdot a^4 + 2 \cdot a^3 + 0 \cdot a^2$$
$$a^{10} \Rightarrow 0 \cdot a^4 + 2 \cdot a^3 + 2 \cdot a^2$$

This example illustrates again the fact that maximizing the number of consumed objects is different than applying the rules in a maximally parallel way; for instance, we cannot distribute a^{10} as $3 \cdot a^3$ which is maximally parallel, but it is not maximum consuming.

The *resource mapping problem* for simple P systems (shortly $\mathbf{RMP_s}$) can be formulated as follows: given a resource allocator RA for a membrane, construct a multiset which is maximum consuming. Thus $\mathbf{RMP_s}$ can be viewed as an implementation of a resource allocator RA.

Definition 1. Formally, *an instance of* $\mathbf{RMP_s}$ is given by (i, w_i, R_i) where:

1. i is a membrane of a simple P system,
2. w_i is the multiset of objects associated with membrane i,
3. R_i is the set of rules associated with membrane i,

such that for

$$M = \left\{ \{u_l \to v_l\}^{k_l} \mid u_l \to v_l \in R_i \right\}, \text{ and}$$
$$\sum_{\{u_l \to v_l\}^{k_l} \in M} u_l^{k_l} \subseteq w_i$$

we obtain a maximum $\sum_{\{u_l \to v_l\}^{k_l} \in M} |u_l|\, k_l$.

We investigate the computational complexity of this problem, and show that the decision version of the resource mapping problem in simple P systems is **NP**-complete. Without loss of generality, we consider the decision version of $\mathbf{RMP_s}$ where we have as input a single membrane of a simple P system with a resource allocator, and a number T which represents the number of objects we intend to rewrite; we ask whether there exists a multiset of rules which consumes at least that number T of objects. Formally, using the framework of Definition 1, we ask whether there exists $\overline{M} = \left\{ \{u_j \to v_j\}^{k_j} \mid u_j \to v_j \in R_i \right\}$ such that $\sum_{\{u_j \to v_j\}^{k_j} \in \overline{M}} u_j^{k_j} \subseteq w_i$ and $\sum_{\{u_j \to v_j\}^{k_j} \in \overline{M}} |u_j| k_j \geq T$, where $|u_j|$ denotes the cardinality of the multiset u_j. Using a classical approach, [3] it is shown first that $\mathbf{RMP_s}$ is in **NP**, and then that the knapsack problem can be polynomial-time reduced to $\mathbf{RMP_s}$. For the latter, a version of the knapsack problem **KNAP** is considered, where the weight of an item is equal to its profit and we can choose an item multiple times.

Theorem 4.1. *The decision version of* $\mathbf{RMP_s}$ *is* ***NP****-complete.*

We consider now a reverse reduction, of the resource allocation problem to the knapsack problem.

Each evolution step of a simple P system is composed of three stages: the first stage consists of the assignment of objects to rules according to the resource allocator, the second stage represents the distribution of the results obtained after applying the selected rules, and the final stage consists of the dissolution of certain membranes. We create an instance of the discrete knapsack problem based on the multiset of objects, then we solve this instance obtaining the corresponding rules. After applying the rules, the resulting objects are moved according to their tags *here*, *out*, and in_j.

The membranes containing the special symbol δ are dissolved, and their contents are transferred to their parents (the δ symbols are not transferred).

For the first stage of the process we present a function that transforms an instance of the resource allocation problem into a knapsack instance such that we can obtain a solution of the $\mathbf{RMP_s}$ instance by solving the transformed instance.

Given a membrane of a simple P system, we define the capacity c of the knapsack, the number n of items, the weight g_i and profit p_i for each item i, as follows:

$$\begin{aligned} &c = |w|; \\ &g_{i_k} = k \cdot h, \text{ where } R = \{r_1, \ldots, r_m\},\ r_i = u_i^h \to v_i,\ u_i \in O \ \wedge \\ &\quad k = \overline{1,m} \text{ where } m = \max\{j \in \mathbb{N} \mid w' \subseteq w \ \wedge \ w' = j \cdot u_i\};\ i = \overline{1,|R|}; \\ &n = |\{g_{i_k}\}|; \\ &p_{i_k} = g_{i_k}. \end{aligned} \tag{2}$$

The transformation f is defined by (2). In the knapsack problem, an item can be used only once; however in membrane systems, a rule can be applied several times. Thus for every rule we define a "class" of items such that each item represents a rule application. We denote by W such a set of items; thus we have $|W| \leq c \cdot |R|$, because a rule cannot be applied more than c times, and there exist $|R|$ rules. Since we are interested in consuming as many objects as possible, the profit of an item is defined as the number of objects it consumes.

The transformation f can be computed in polynomial time with respect to $|w|$.

In the first stage we transform an instance of the resource allocation problem to an instance of the knapsack problem by using the function f. Then we solve the new created instance, and obtain the rules which can be applied in parallel, together with the multiplicity of each rule. We can express the computational complexity of each stage with respect to the input represented by the multiset of objects w of the membrane.

For the first stage we can define a relation between the number of the created items and the size of the input multiset. According to equations (2), each rule $u_i \to v_i$ can introduce a maximum of $\frac{|w|}{|u_j|}$ corresponding items in the knapsack instance. Summing these quantities, for each rule we have that

$$n \leq \sum_{i=1}^{|R|} \frac{|w|}{|u_i|} = |w| \sum_{i=1}^{|R|} \frac{1}{|u_i|}.$$

Note that $\sum_{i=1}^{|R|} \frac{1}{|u_i|}$ is a constant associated to the membrane, because the rules of a membrane do not change in the process of evolution. We denote this constant with S, and so $n \leq |w| \cdot S$. Thus the complexity of this stage is $O(n) = O(|w| \cdot S)$.

For the second stage we use a pseudo-polynomial algorithm for the knapsack problem having a complexity of $O(n \cdot c)$, where n is the number of items in the knapsack instance, and c is the capacity of the knapsack. By using the estimated number of

items expressed in the first stage and the fact that $c = |w|$, the complexity of the second stage is $O(|w| \cdot S \cdot |w|) = O(|w|^2 \cdot S)$.

The third stage consists of applying the rules according to the solution of the knapsack problem. It is worth to note that we can have different rules which give the same profit. The pseudo-polynomial algorithm for knapsack can retrieve the selected items in $O(n)$ steps, and so the complexity of this step is $O(|w| \cdot S)$. During the backtracking process used for obtaining the selected items we nondeterministically choose a rule corresponding to each item. Thus we obtain a multiset of rules which corresponds to a maximum consuming evolution, and the evolution is nondeterministic because of the way the rules are chosen. Thus a complexity of an evolution step for a membrane of a simple P system is

$$O(|w| \cdot S) + O(|w|^2 \cdot S) + O(|w| \cdot S) = O(|w|^2 \cdot S).$$

An example and more details are presented in [2].

4.2. Maximum Cooperative P Systems

Maximum cooperative P systems are transition P systems with cooperative rules that evolve at each step by consuming the maximum number of objects. The computational complexity of the maximum cooperative P systems is connected to the classical optimization problem of *integer linear programming* problem. Moreover, the decision problem related to the allocation of resources for maximum cooperative P systems is **NP**-complete.

We use the same resource allocator for a membrane which can be formally defined as a mapping $RA : O^* \to (O^*)^{n+1}$, where O is the alphabet of objects, and $R = \{u_j \to v_j \mid j = \overline{1,n}\}$ is the set of applicable rules contained in the membrane. Considering $w \in O^*$ as the membrane contents, and w' the multiset of objects that are not consumed, we have

$$RA(w) = \{(w_1, w_2, \ldots, w_n, w') \mid w_j \not\subseteq w' \wedge w_j = {u_j}^{k_j}, k_j \in \mathbb{N}, j = \overline{1,n} \text{ and } \sum_{j=1}^{n} w_j + w' = w\}. \tag{3}$$

To study the computational complexity systems evolution, we define the *resource mapping problem* for maximum cooperative P systems (shortly $\mathbf{RMP_c}$) used to compute the distribution of objects to rules in order to achieve a maximum consuming evolution.

Definition 2. *An instance of* $\mathbf{RMP_c}$ is given by (i, w_i, R_i) where:

1. i is a membrane of μ, where μ is a membrane structure of a maximum cooperative P system,
2. w_i is the multiset of objects in membrane i,
3. R_i is the set of applicable rules within membrane i,

such that for

$$H^i = \left\{ \{u_{il} \to v_{il}\}^{k_{il}} \mid u_{il} \to v_{il} \in R_i,\, k_{il} \in \mathbb{N},\, l = \overline{1, |R_i'|} \right\} \text{ and } \sum_{l=1}^{|R_i|} u_{il}^{k_{il}} \subseteq w_i$$

we obtain a maximum value for $\sum_{l=1}^{|R_i|} |u_{il}|\, k_{il}$.

The version of the resource mapping problem defined for maximum cooperative P systems is different from the version defined for simple P systems. The difference consists in the rules which compose the multiset: in the version for simple P systems we allow only rules having the left-hand side consisting of a single object with a certain multiplicity; in maximum cooperative P systems we allow rules having several objects on the left-hand side. This difference is emphasized by the notations we use: $\mathbf{RMP_s}$ is the version for simple P systems, and $\mathbf{RMP_c}$ is the version for maximum cooperative P systems.

In [3] is is shown that the decision version of the resource mapping problem $\mathbf{RMP_c}$ is **NP**-complete. Without loss of generality, one considers the decision version of $\mathbf{RMP_c}$ where we have as input a single membrane and a number T which represents the number of objects we intend to rewrite, and ask whether there exists a multiset of rules which consumes at least T objects. Formally, using the framework of Definition 2, we ask whether there exists a multiset of rules

$$\overline{H^i} = \left\{ \{u_{ij} \to v_{ij}\}^{k_{ij}} \mid u_{ij} \to v_{ij} \in R_i \wedge k_{ij} \in \mathbb{N},\, j = \overline{1, |R_i|} \right\} \text{ such that}$$

$\sum_{j=1}^{|R_i|} u_{ij}^{k_{ij}} \subseteq w_i$ and $\sum_{j=1}^{|R_i|} |u_{ij}|\, k_{ij} \geq T$, where $|u_{ij}|$ denotes the length of the string u_{ij}.

Theorem 4.2. *The decision version of* $\mathbf{RMP_c}$ *is* ***NP****-complete.*

The proof in [3] consists of two parts: first it is proved that $\mathbf{RMP_c}$ is in **NP**, and then that $\mathbf{RMP_c}$ is **NP**-complete. To show that $\mathbf{RMP_c}$ is **NP**-complete we use the fact that it contains $\mathbf{RMP_s}$ as a subproblem. It is easy to see that every instance of $\mathbf{RMP_s}$ is an instance of $\mathbf{RMP_c}$, because the rules of simple P systems are special instances of those in maximum cooperative P systems. This can be viewed as a reduction where the identity function transforms an instance of $\mathbf{RMP_s}$ into an $\mathbf{RMP_c}$ instance. Using this in conjunction with the fact that $\mathbf{RMP_s}$ is **NP**-complete, it follows that $\mathbf{RMP_c}$ is **NP**-complete. The technical details of the proof are presented in [3].

The evolution of a maximum cooperative P system can be described by using the integer linear programming (**ILP**) to establish the dependencies between the objects from the left-hand side of the rules and those available in each membrane. Each evolution step is composed of three stages: the assignment of objects to rules according to the resource allocator, the distribution of the results obtained after applying the selected rules, and the dissolution of certain membranes. Such an evolution step can be simulated as follows. We create an instance of **ILP** based on the multiset of available objects, and then we solve it, obtaining the corresponding rules. After applying the rules, the resulting objects are moved according to their tags (*here*, *out*,

and in_j). The membranes containing the special symbol δ are dissolved, and their contents are transferred to their parents (the δ symbols are not transferred).

For the first stage of the process we present a function that transforms an instance of the resource allocation problem into an integer linear programming instance such that a solution of the $\mathbf{RMP_c}$ instance is obtained by solving the **ILP** instance.

Given a membrane of a maximum cooperative P system with the available object multiset w and the rule set R, we construct a corresponding **ILP** instance. We consider the objects alphabet $O = \{o_1, o_2, \ldots\}$, and $w(o)$ represents the multiplicity of object o from a multiset w. For the **ILP** instance, we define the number n of variables, the number m of constraints, a $m \times n$ matrix A of positive coefficients together with a vector $b \in \mathbb{Z}^m$ representing the constraints, and the coefficients $c \in \mathbb{Z}^n$ of the objective function to maximize:

$$\begin{aligned}
&n = |R|,\\
&m = |\{o \in \overline{w} \mid \overline{w}(o) > 0\}|, \text{ where } \overline{w} = \left(\sum_{u_j \to v_j \in R} u_j\right) \cap w,\\
&c \in \mathbb{Z}^n,\ c_j = |u_j|,\ u_j \to v_j \in R \text{ for all } j = \overline{1,n},\\
&b \in \mathbb{Z}^m,\ b_i = w(o_i) \text{ for all } i = \overline{1,m},\\
&A = (a_{ij}),\ a_{ij} = u_j(o_i), \text{ where } R = \{u_j \to v_j \mid j = \overline{1,n}\} \wedge o_i \in O,\ \forall i = \overline{1,m}.
\end{aligned} \tag{4}$$

The transformation f is defined by (4). Each variable from the **ILP** instance corresponds to a certain rule (i.e., x_j corresponds to $u_j \to v_j \in R$). The solution $x \in \mathbb{Z}^n$ indicates the number of times each rule is applied. The constraints define how many available objects can be consumed.

A constraint is defined whenever a rule $u_j \to v_j$ involves the consumption of an available object o (i.e., $u_j(o) > 0 \ \wedge w(o) > 0$). Hence the number of constraints is equal to the number of distinct objects that can be consumed by applicable rules. This number is equal to the number of distinct objects of the multiset obtained from the intersection between the sum multiset of the rules left-hand side with the multiset of available objects. The constraint associated with object o_j is of the form $\sum_{i=1}^{n} a_{ij} \leq b_j$ for all $j = \overline{1,m}$, indicating the fact that we cannot consume more objects o_j than are available.

The transformation f can be computed in polynomial time with respect to $|w|$ and $|R|$. The number n of variables is computed in $O(|R|)$, the number m of constraints is computed in $O\left(\sum_{u_j \to v_j \in R} |u_j|\right) + O(|w|) = O(|w|)$, c is computed in $O(|R|)$, b is computed in $O(m)$, and A is computed in $O(nm) = O(|R| \cdot |w|)$. Thus f can be computed in $O(|R|) + O(|w|) + O(|R| \cdot |w|)$.

According to [7], the size L of an **ILP** instance is $nm + \log|P|$. Knowing that $n = |R|$, $m = O(|w|)$, and $|P| = O((nm)^2)$, we have $L = O(|R| \cdot |w|) + O(\log(|R| \cdot |w|))$.

In the first stage, an instance of the resource mapping problem is transformed into an instance of the integer linear programming problem by using the function f. Thus

the complexity of this stage is the complexity of f, namely $O(|R|)+O(|w|)+O(|R|\cdot|w|)$. The **ILP** instance is solved, obtaining the rules and their multiplicity which can be applied in parallel.

The complexity of the second stage is the complexity of the algorithm used to solve the **ILP** instance. All known algorithms for solving **ILP** are exponential in the size of the input; branch-and-bound and cutting plane are two such algorithms. An upper bound for the solution of **ILP** is presented in [7]; if an instance of **ILP** has a finite optimum, then it has an optimal solution x such that $|x_j| \leq n^3[(m+2)a_3]^{4m+12} = M$, where $a_3 = \max\{a_1, a_2\} \cup \{|c_j| \mid j = \overline{1,m}\}$, $a_1 = \max_{i,j}\{|a_{ij}|\}$, and $a_2 = \max_{i}\{|b_i|\}$ for all $i = \overline{1,n}$, $j = \overline{1,m}$. Thus the logarithm of the solution size is polynomial in the size of the input

$$\begin{aligned}\log M &= 3\log n + (4m+12)[\log(m+2) + \log a_3] = O(L^2) = \\ &= O((|R|\cdot|w| + \log(|R|\cdot|w|))^2) = \\ &= O((|R|\cdot|w|)^2).\end{aligned}$$

The third stage consists of applying the rules according to the solution of the **ILP** instance. The solution to **ILP** is the vector $x \in \mathbb{Z}^n$, where x_j represents the number of times we apply the rule $u_j \to v_j \in R$, for all $j = \overline{1,n}$. It follows that the complexity of this step is $O(|R|)$.

The complexity of an evolution step for a membrane of a maximum cooperative P system is the sum of the complexities of the three stages.

An example and more details are presented in [3].

5. Open Problems

As mentioned above, all previous results need further investigations, for improving them to the complexity level of systems used in universality proofs for the *maxP* case. We do not repeat these open problems here, but we point out further directions of research (most probably, several of them can be settled in a similarly easy way as the way the previous constructions are similar to constructions used in the case *maxP*).

We have mentioned that the main goal of [2] was to investigate the complexity of finding the multiset to use in each step of a computation in the case *maxO* for a special type of multiset rewriting P systems; what about *maxR*, what about classes of P systems other than the multiset rewriting ones?

The parallelism (together with the possibility of creating an exponential workspace in linear time) is the basic ingredient for devising efficient solutions to computationally hard problems, mainly in terms of P systems with active membranes. The parallelism is traditionally maximal in the sense of *maxP*. What about the other types of maximality, *maxO* and *maxR*?

Maximizing the number of objects or of rules can be related to the idea of using energy for controlling the computations, [10], where the rules to be used can be chosen in such a way to maximize or minimize the consumed energy. Can these two ideas be related (e.g., through mutual simulations)?

References

[1] ALHAZOV A., FREUND R., ROGOZHIN Y., *Computational power of symport/antiport: history, advances and open problems*, Lecture Notes in Computer Science, vol. **3850**, Springer, 2006, pp. 1–30.

[2] CIOBANU G., RESIOS A., *Computational complexity of simple P systems*, Fundamenta Informaticae, **87**, 2008, pp. 49–59.

[3] CIOBANU G., RESIOS A., *The evolution of maximum cooperative P systems is NP-complete*, Natural Computing, 2009, to appear.

[4] FREUND R., KARI L., OSWALD M., SOSIK P., *Computationally universal P systems without priorities: two catalysts are sufficient*, Theoretical Computer Science, **330**, 2005, pp. 251–266.

[5] IBARRA O. H., PĂUN A., RODRÍGUEZ-PATÓN A., *Sequentiality induced by spike numbers in SN P systems*, *Proceedings 14th DNA Computing*, Prague, 2008, pp. 36–46.

[6] MINSKY M., *Computation: Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, NJ, 1967.

[7] PAPADIMITRIOU C. H., STEIGLITZ K., *Combinatorial Optimization: Algorithms and Complexity*, Dover, 1998.

[8] PĂUN G., *Computing with membranes*, Journal of Computer and System Sciences, **61**, 2000, pp. 108–143.

[9] PĂUN G., *Computing with Membranes: An Introduction*, Springer, Berlin, 2002.

[10] PĂUN G., SUZUKI Y., TANAKA H., *P systems with energy accounting*, International Journal of Computer Mathematics, **78**, 2001, pp. 343–364.

[11] P Systems Web Site: `http://ppage.psystems.eu`.