Solving complex problems with a bioinspired model

Alberto Arteta [a,*], Nuria Gomez [b,1], Luis Fernando Mingo [a,2]

[a] Crtra Valencia km 7, 28034 Madrid, Spain
[b] Campus Montegancedo Boadilla del Monte, Madrid, Spain

ARTICLE INFO

Keywords:
P-systems
Membrane computing
Natural computing
Evolution rules application
Redundant rules
Fast linear algorithms

ABSTRACT

Membrane systems are parallel and bioinspired systems which simulate membranes behavior when processing information. As a part of unconventional computing, P-systems are proven to be effective in solving complex problems.

A software technique is presented here that obtain good results when dealing with such problems. The rules application phase is studied and updated accordingly to obtain the desired results. Certain rules are candidate to be eliminated which can make the model improving in terms of time.

1. Introduction

"*Membrane computing is a parallel and distributed computational model based on the membrane structure of living cells*" (Paun, 1998). This model has become, during these last years, a powerful framework for developing new ideas in theoretical computation. Membrane systems are commonly referred as Transition P-systems.

Most membrane systems are computationally universal: "*P-systems with simple ingredients (number of membranes, forms and sizes of rules, controls of using the rules) are Turing complete*" (Paun, 2005).

This framework is extremely general, flexible, and versatile. "*Several classes of P-systems with an enhanced parallelism are able to solve computationally difficult problems (typically, NP complete problems) in a feasible amount of time (Polynomial or linear)*". Solving multidimensional (2005) have been proven to be an efficient tool to deal with known and complex problems as the *multidimensional 0-1 knapsack one* (Solving multidimensional, 2005) or the *Boolean satisfiability problem* (Solving SAT, 2005).

In Transition P-systems, each evolution step is obtained through two consecutive phases within each membrane:

In the first stage the evolution rules are applied. In the second one, the communication between membranes is established. This work is focused on the first phase, the application of active rules.

This application has had the condition of using the rules in a maximally parallel way although recent findings consider the *minimal* parallelism of using the rules: "*if at least a rule from a set of rules associated with a membrane or a region can be used, then at least one rule from that membrane or region must be used, without any other restriction*" (Ciobanu et al., 2007). This condition also relaxes the requirements of developing new algorithms that implement the rules application phase.

This section describes the literature regarding membrane systems. There are conferences, workshops and symposiums specialized in membrane computing such as the *workshop on Membrane Computing, under the auspices of IEEE Computational Intelligence Society Emergent Technologies Technical Committee*. Proceedings from this workshop are published in *Springer Lecture Notes in Computer Science series*. Other important workshops are: *Workshop of Unconventional Computing* whose proceedings are published in the *LNCS series of Springer.*, *Brainstorming Week on Membrane Computing* whose proceedings are published in *Journal of unconventional computing (ISSN 1548-7202)*, *Journal of Universal Computer Science, Soft Computing and International Journal of Foundations of Computer Science*.

Other prestigious journals show related work such as the *Journal of Parallel and Distributed Systems*, *Theoretical Computer Science* (*ISSN: 0304-3975*) and *Engineering Applications of Artificial Intelligence* (*ISSN: 0952-1976*).

Foundations and organizations were aware of this new revolutionary way of computing and encouraged the research in that direction. Examples of these are *EMCC* (*European Molecular Computing Consortium*) and *The Consortium for Biomolecular Computing*.

Transition P-system technology has its own web which has recently been updated from http://psystems.disco.unimib.it to

* Corresponding author. Tel.: +34 913365104.
E-mail addresses: aarteta@eui.upm.es (A. Arteta), ngomez@eui.upm.es (N. Gomez), lfmingo@eui.upm.es (L. Fernando Mingo).
[1] Tel.: +3491337445.
[2] Tel.: +34913365104.

http://ppage.psystems.eu On this website, it is possible to find a large amount of information about membrane systems: Articles, papers, news and updates.

Users and researchers utilize forums to leave comments about membrane systems; (http://cantor.cs.us.es/~fsancho/foro/viewtopic.php?t=9) Membrane computing has often been addressed as a revolutionary way of computing by several scientific organizations. Furthermore, P-systems have been suffered a transformation themselves. This transformation comes from the desire of collecting more biological properties. Here below, there are some of the features that membrane systems have achieved:

In Ibarra (2006), a computational complexity analysis is performed in membrane systems. O. Ibarra, S. Worwood, H. Yen and Z. Zhang study the computational power of different variants of sequential P-systems. They show two types of results: there are sequential P-systems that are universal and sequential P-systems that are nonuniversal. "In particular, both communicating and cooperative P-systems are universal, even if restricted to 1-deterministic system with one membrane. However, the reachability problem for multi-membrane catalytic P-systems with prioritized rules is NP-complete and, hence, these systems are nonuniversal" '.

Zandron (Solving NP-Complete Problems, 2003) was able to create a model to Solve NP-Complete Problems by using Membranes systems. For example, he solves two different NP-complete problems in linear time: satisfiability problem (SAT) and Hamiltonian path problem. The authors arrive to this conclusion after proving two complex theorems.

Narayanan and Rama (2003) proposes a membrane system that is able to break a cryptosystem, DES. The DES key can be obtained in linear time with respect to the length of the key.

In 2005, Paun (Solving multidimensional, 2005) builds a membrane system model that solves multidimensional 0–1 knapsack problem.

Regarding the implementation of the rules application phase in the membrane systems, several algorithms have been implemented:

Step by Step (Arroyo et al., 2003). This algorithm involves the random selection of one of the active evolution rules for calculating a final set of active rules.

Applicability BenchMark (Fernandez et al., 2006a). Once an evolution rule has been selected in a non-deterministic manner, the rule is applied a random number of times between 1 and the maximal applicability benchmark, per iteration.

"The algorithm can apply simultaneously several rules several times in the same membrane. Moreover, if every needed condition is accomplished all the rules can be simultaneously applied in the whole P-system".

Algorithm of active rules elimination for evolution rules application (Tejedor et al., 2007). This one eliminates an active evolution rules for every evolutionary step. In each step of this algorithm two main actions are carried out eliminating, at least, an evolution rule to the set of active rules. Therefore, the number of operations executed is limited and it can be known a priori which its execution time is at worst.

Fast Linear algorithm (Gil et al., 2009). This algorithm is the best candidate for sequential devices. It obtains the fastest results. "The algorithm is based on the one by one elimination of rules: when a rule has been applied to its maximal applicability benchmark, this rule stops being active, and, therefore, it is eliminated. The algorithm finishes when all rules have been eliminated".

The *fast linear algorithm, FLA* (Gil et al., 2009) has been proven to obtain the best performance in general when implementing the rules application phase. By updating the models in Solving NP-Complete Problems (2003), Solving multidimensional (2005), Solving SAT (2005), Narayanan and Rama (2003) with the technique used in *FLA* we are certain that it achieves an optimal performance as the models reduce the operations.

Presently, there are some others sequential algorithms for rules application in P-systems (Ciobanu and Paraschiv, 2002; Fernandez et al., 2006b; Arteta et al., 2008) that use different techniques. In particular Arteta et al. (2008) use the resolution of linear system diophantine equations to obtain the number of times that rules should be applied. However, none of these improve the performance of FLA.

2. Definitions

Here are some necessary definitions to understand the membrane model. In this section, we formally define the membrane system or P-system here. We also define concepts such as multiset of objects, evolution rules and multiplicity of objects. These definitions create a better understanding of the strategy we are going to follow to improve the functionality of the membrane systems. These concepts are vastly described in Paun (1998).

2.1. Transition P-systems

A Transition P-system of degree n, $n > 1$ is a construct

$$\prod = (V, \mu, \omega_1, \ldots, \omega_n, (R_1, \rho_1), \ldots (R_n, \rho_n), i_0)$$

where

1. V is an alphabet; its elements are called objects;
2. μ is a membrane structure of degree n, with the membranes and the regions labeled in a one-to-one manner with elements in a given set ; in this section we always use the labels 1,2,...,n;
3. ω_i $1 \leq i \leq n$, are strings from V^* representing multisets over V associated with the regions 1,2,...,n of μ;
4. R_i $1 \leq i \leq n$, are finite set of evolution rules over V associated with the regions 1,2,...,n of μ; ρ_i is a partial order over R_i $1 \leq i \leq n$, specifying a priority relation among rules of R_i. An evolution rule is a pair (u,v) which we will usually write in the form $u \to v$ where u is a string over V and $v = v'$ or $v = v'\delta$ where v' is a string over $(V \times \{here, out\}) \bigcup (V \times \{in_j \ 1 \leq j \leq n\})$, and δ is a special symbol $\in \{dissolve, not\ dissolve\}$. The length of u is called the radius of the rule $u \to v$.
5. i_o is a number between 1 and n which specifies the output membrane of $\prod$.

2.2. Multiset of objects

Let U be a finite and not an empty set of objects and N the set of natural numbers. A *multiset of objects* is defined as a mapping:

$$M : U \to N$$

$$a_i \to u_1$$

where a_i is an object and u_i its multiplicity.

As it is well known, there are several representations for multisets of objects.

$$M = \{(a_1, u_1), (a_2, u_2), (a_3, u_3) \ldots\} = a_1^{u_1} \cdot a_2^{u_2} \cdot a_n^{u_n} \ldots$$

Multisets of objects are processed in the membrane system according to the rules.

Note. *Initial Multiset* is the multiset existing within a given region before evolving.

2.3. Evolution rule

Evolution rule with objects in U and targets in T is defined by $r=(m,c,\delta)$ where $m \in M(U), c \in M(UxT)$ *and* $\delta \in$ {*to dissolve,not to dissolve*}

From now on 'c' will be referred to the consequent

Note. The *set of evolution rules* with *objects* in U and targets in T is represented by $R\ (U,\ T)$.

The rules affect the consumption of objects. After applying them, the multiset of objects change accordingly.

2.4. Multiplicity of an object in a multiset of objects M(U)

Let $a_i \in U$ be an object and let $m \in M(U)$ be a multiset of objects. The multiplicity of an object is defined by a multiset of objects such as

$$\|_{a_i} : U \times M(U) \to N$$

$$(a_i,m) \to |m|_{a_i} = n | (a_i,n) \in m$$

2.5. Multiplicity of an object within a rule

Let $a_i \in U$ be an object and let $R(U,T)$ be a multiset of evolution rules. Let $r=(m,c,\delta) \in R(U,T)$ where $m \in M(U), c \in M(UxT)$ *and* $\delta \in$ {*to dissolve,not to dissolve*}

The multiplicity of an object in an evolution rule r is defined by an evolution rule such as

$$\|_{a_i} : U \times R(U,T) \to N$$

$$(a_i,r) \to |m|_{a_i} = n | (a_i,n) \in m$$

Let C_i be the *consequent* of *the rule* r_i. Thus,

Here is a representation:

$$r_1 : a_1^{u_{11}} a_2^{u_{12}} \ldots a_n^{u_{1n}} \to C_1$$
$$r_2 : a_1^{u_{21}} a_2^{u_{22}} \ldots a_n^{u_{2n}} \to C_2$$
$$\ldots\ldots\ldots\ldots \to \ldots\ldots\ldots\ldots$$
$$r_m : a_1^{u_{m1}} a_2^{u_{m2}} \ldots a_n^{u_{mn}} \to C_m$$

Observation: Let $k_i \in N$ be the number of times that the r_i is applied. Therefore, the number of symbols a_j which have been consumed after applying the evolution rules a specific number of times are

$$\sum_{i=1}^{m} k_i \cdot u_{ij}$$

2.6. Extinguished multisets

Given a region R, let U be an alphabet of objects $U = \{a_i | 0 < i \leq n\}$.

Let $M(U) = \{(a_i,u_i) | a_i \in U\ u_i \in \mathrm{N}\ 0 < i \leq n\}$ the set of all the multisets over U. Let $x \in M(U)$ be a multiset of objects over U within R. Let $R(U,T) = \{r_i\ | \exists m \in \mathrm{N}\ i \leq m,\ i \in \mathrm{N}\}$ be a *set of evolution rules.* $r_j = (x_j,c_j,\delta) \in R(U,T)$ *and* $x_j = \{(a_{ji},u_{ji})\ | \exists n,m \in \mathrm{N}\ i \leq n,\ j \leq m\ i,\ j \in \mathrm{N}\}$.

Let $k_j \in N$ the number of times that the rule $r_j \in R(U,T)$ is applied over x. we say x is an *maximal or extinguished multiset if* and only if

$$\bigcap_{j=1}^{m} \left[\bigcup_{i=1}^{n} \left(u_i - \sum_{j=1}^{m} (k_j \cdot u_{ji}) \leq u_{ji} \right) \right] \quad (1)$$

Observation: In other words, m is an extinguished *multiset* if and only if it is not possible to apply any more *rules to it.*

2.7. Competiveness

In a given region, Let $R(U,T)$ be the multiset of evolution rules within that region. Let $r_i, r_j \in R(U,T)\ i \neq j$. r_i competes with r_j if and only if their antecedents have objects in common.

2.8. Maximal applicability benchmark of a rule over a given multiset

In a given region, let m be a multiset of objects and r_i an evolution rule. The *maximal applicability benchmark* of r_i over m is the maximum number of times that r_i can be applied over m.

3. Evolution rules application phase

In this section we briefly describe the evolution rules application phase, which has been implemented following different techniques.

As previously mentioned, this phase is part of the evolution of the membrane systems. The membrane system evolves in relations to how the rules are applied. Non-determinism is part of the inner behavior of the P-system. Any valid implementation of this phase must respect the non-deterministic character of the P-system.

In every region within a P-system, the evolution rules application phase is described as follows:

Rules application to a multiset of objects in a region is a transforming process of information which has input, output and conditions for making the transformation.

Given a region within a P-system, let $U = \{a_i\ | 1 \leq i \leq n\}$ be the alphabet of objects, m a multiset of objects over U and $R(U,T)$ a multiset of evolution rules with antecedents in U and targets in T.

1. The input in the region is the initial multiset m,
2. The output is an extinguished multiset m',
3. The transformations have been made based on the application of the evolution rules over m until m' is obtained.

Application of evolution rules in each region of the P-systems involves subtracting objects from the initial multiset by using the antecedents of the rules. Rules used are chosen in a non-deterministic manner. This phase ends when no rule is applicable anymore.

The transformation only needs antecedents because the consequents of the rules are part of the communication phase. Membrane computing is a parallel model, therefore, all of these transformations must occur in a parallel manner. However, the condition of using the rules in a maximally parallel way, has recently been simplified. This has occurred due to the *minimal parallelism.* Ciobanu et al. (2007) defined in the rules application phase.

The parallelism found in this process is a subject that is commonly analyzed. Several authors have defined a *degree of parallelism* in P-systems *"for proving some results concerning the maximum number of applications of rules in a single step through the computation of a P-system"* (Gutiérrez-Naranjo et al., 2007).

All of these concepts create a more flexible idea in regards to building new algorithms that implement the evolution rules application phase.

Any algorithm that implements the rules application phase must respect the non-determinism and can return any possible extinguished multiset of object. This condition is very important

as if this did not occur, this implementation would not be correct as it would not respect Paun's model.

3.1. Fast linear algorithm (FLA)

The algorithm is based on elimination of rules one by one: When a rule has been applied to its maximal applicability benchmark, this rule is not active anymore and, therefore, it is eliminated. FLA finishes when all rules have been eliminated.

The algorithm is made up of two phases:

1. All rules belonging to the set of active rules -except one- are applied a random number of times between 0 and its maximal applicability benchmark. The unique rule that has not been applied a random number of times, it is applied the last one and the same number of times as its applicability benchmark.
2. In the second phase, all of the rules are applied to its maximal applicability benchmark.

Consequently, there are no applicable rules left, and the algorithm finishes, generating the following result:

1. A multiset of applied rules,
2. An extinguished multiset of objects.

This algorithm is sequential and it obtains the smallest execution times compared to other algorithms. Therefore, it is more appropriate for the implementation of Transition P-systems in sequential devices. We focus on improving this algorithm by using an universal technique that can be used to improve the performance in different models.

4. Bioinspired model

The idea of bypasssing evolving rules is revolutionary. Some rules can be ignored during the second phase of the application phase.

In the methods we have seen above, there is a phase in which a rule is selected randomly between all the active rules. That rule is applied a number of times between 0 and its maximal applicability benchmark except the last rule which is applied to its maximal applicability. This assures non-deterministic aspects of the process.

If we obtain an extinguished multiset, then the process is finished. When analyzing the inherent properties in the multiplicities of the evolution rules, a group of redundant evolution rules can be obtained. These evolution rules will be a subset of the multiset of evolution rules that we are working within our membrane. By detecting and removing the redundant evolution rules it is possible to achieve improved performance in terms of time.

Otherwise FLA applies all the rules a number of times equal to their maximal applicability benchmark.

This assures that we obtain an extinguished multiset at the end of the evolution rules application phase.

FLA achieves a computational complexity equal to $O(R)$. The number of iterations is $2 \cdot |R| - 1$ and this seems to be the optimal approach. However, we have seen that the method can be improved by detecting the set of redundant evolution rules.

Notation. Given that $R(U, T)$ is a multiset of evolution rules within a region, we denote the set of Redundant evolution rules as $US_{R(U,T)}$.

In order to define a set of redundant evolution rules, we must first define an operation over multisets of objects.

4.1. Order in rules

In a given region, Let $U = \{a_i \mid 1 \leq i \leq n\}$ be an alphabet of objects. Let m be a multiset of objects over U $m = \{(a_i, u_i) | a_i \in U\ u_i \in N\ 1 \leq i < n\}$ and m' another multiset of object over $U, m' = \{(a_i, u_i') | a_i \in U u_i' \in N\ 1 \leq i < n\}$. $m \geq_u m'$ if and only if $u_i \geq u_i'$ $\forall i 1 \leq i \leq n\ i \in N$.

4.2. Set of redundant evolution rules

In a given region, let U be an alphabet of objects and T a set of targets. Let $R(U, T)$ be a multiset of evolution rules with objects in U and targets in T.

$US_{R(U,T)} = \{r_i = (m, c, \delta) | \exists r_j = (m', c', \delta')\quad m \geq_u m'\ r_i, r_j \in R(U,T)\}$.

In other words, we say that a rule r is redundant if another rule r' exists whose objects multiplicities within its antecedent are less than or equal to the objects multiplicities within the antecedent of r

Example.

$U = \{a_1, a_2\}$

$R(U, T) = \{r_1, r_2, r_3\}$

$r_1 = a_1^5 a_2^3 \rightarrow C_1$

$r_2 = a_1^2 a_2^6 \rightarrow C_2$

$r_3 = a_1^3 a_2^3 \rightarrow C_3$

Thus, $m_1 = \{(a_1, 5), (a_2, 3)\}$ and $m_3 = \{(a_1, 3), (a_2, 3)\}$. In this scenario, r_1 is redundant because $5 > 3$ and $3 \geq 3$.

According to the definition of redundant rules, there will always be at least one rule which is not redundant in a multiset of evolution rules.

$n = |R(U, T)| \Rightarrow 0 \leq |US_{R(U,T)}| \leq n - 1$.

When the number of rules is high, obviously the chances of finding redundant rules are high too. On the contrary when the number of symbols increases, finding redundant rules is more difficult.

Let us select two natural numbers x, y randomly. Considering that the multiplicity values can be generated uniformly and randomly from continuous distribution, we do not have information of what numbers we can have.

Thus, in this case:

$P(x > y) = P(x < y)$

Assuming that most computers work with a range of natural numbers from $[0, 2^{32}]$ we can state for certain that

$P(x = y) = \frac{1}{2^{32}} \approx 0 x, y \in N$.

$P(x \geq y) = P(x > y) + P(x = y) \Rightarrow P(x \geq y) \approx P(x > y)$.

Moreover, for every pair of random numbers x, y we know $P(x > y) = P(x < y)$ and also

$P(x \geq y) + P(x < y) = 1$.

Thus, $P(x \geq y) \approx P(x > y) = 1/2$

$P(x < y) = 1/2$

Given a multiset of evolution rules $R(U,T)$, we are interested in calculating the probability of having redundant rules within $R(U, T)$.

Thus P (*having redundant rules in* $R(U, T)$) $= 1 - P$ (*not having any redundant rules in* $R(U, T)$) which is the same as

$P(US_{R(U,T)} \neq \phi) = 1 - P(US_{R(U,T)} = \phi)$.

Let us suppose we have two evolution rules as follows:

$r_1 = a_1^{u_1} a_2^{u_2} \dots a_n^{u_n} \to C_1$

$r_2 = a_1^{u'_1} a_2^{u'_2} \dots a_n^{u'_n} \to C_2$

In this scenario, the favorable cases for not having redundant rules are all of the possible cases except these two:

1. $u_i \le v_i \quad \forall i \le n$,
2. $u_i \ge v_i \quad \forall i \le n$.

Lemma 1. *Let r_1, r_2 be two evolution rules*

$r_1 : a_1^{u_1} a_2^{u_2} \dots a_n^{u_n} \to C_1$

$r_2 : a_1^{v_1} a_2^{v_2} \dots a_n^{v_n} \to C_2$

The probability of not having any redundant rules is

$$P(\neg\exists \textit{ red rule}\{r_1, r_2\}) = P(US_{\{r_1,r_2\}} = \phi) = \frac{2^n - 2}{2^n}$$

Proof. Clearly, a redundant rule exists if and only if

$u_1 \le v_1, u_2 \le v_2, \dots u_n \le v_n$

or

$u_1 \ge v_1, u_2 \ge v_2, \dots u_n \ge v_n$

This is the same as

$input(r_1) \subset input(r_2)$

or

$input(r_1) \subset input(r_2)$

Thus, $P(\exists \textit{ red rule}\{r_1,r_2\}) = P(input(r_1) \subset input(r_2)) + P(input(r_2) \subset input(r_1)) = P(u_1 \le v_1, u_2 \le v_2, \dots u_n \le v_n) + P(u_1 \ge v_1, u_2 \ge v_2, \dots u_n \ge v_n)$.

$P(u_1 \le v_1, u_2 \le v_2, \dots u_n \le v_n) = 1/2 \times 1/2 \times \dots \times 1/2 = (1/2)^n$

$P(u_1 \ge v_1, u_2 \ge v_2, \dots u_n \ge v_n) = 1/2 \times 1/2 \times \dots \times 1/2 = (1/2)^n$

$$(1/2)^n + (1/2)^n = \frac{1}{2^n} + \frac{1}{2^n} = \frac{2}{2^n}.$$

Furthermore,
$P(\neg\exists \textit{ red rule}\{r_1, r_2\}) = P(US_{\{r_1,r_2\}} = \phi) = 1 - P(\exists \textit{ red rule}\{r_1, r_2\})$,
therefore: $P(\neg\exists \textit{ redundant rule}) 1 - 2/2^n = (2^n - 2)/2^n$ and lemma is proved. □

Lemma 2. *In order to clarify the proof the notation. We establish the following*:

$$M = \binom{m}{2}$$

and

$$M + 1 = \binom{m+1}{2}$$

Let $r_1, r_2, \dots, r_m$, be m evolution rules:

$r_1 : a_1^{u_1} a_2^{u_2} \dots a_n^{u_n} \to C_1$
$r_2 : a_1^{v_1} a_2^{v_2} \dots a_n^{v_n} \to C_4$
$\dots$
$r_m : a_1^{z_1} a_2^{z_2} \dots a_n^{z_n} \to C_m$

$$P(\neg\exists \textit{ red rule}\{r_1, r_2, \dots, r_m\}) = \left(\frac{2^n - 2}{2^n}\right)^M$$

Proof. Following the Induction method for m (number of rules):
$m = 2 \xrightarrow{lemma1}$ It is proved.
Now we must prove:

$$P(\neg\exists \textit{ red rule}\{r_1, r_2, \dots, r_m\}) = \left(\frac{2^n - 2}{2^n}\right)^M$$

$$\Rightarrow P(\neg\exists \textit{ red rule}\{r_1, r_2, \dots, r_{m+1}\}) = \left(\frac{2^n - 2}{2^n}\right)^{M+1}$$

Given $m+1$ rules:

$r_1 : a_1^{u_1} a_2^{u_2} \dots a_n^{u_n} \to C_1$
$r_2 : a_1^{v_1} a_2^{v_2} \dots a_n^{v_n} \to C_2$
$\dots$
$r_m : a_1^{z_1} a_2^{z_2} \dots a_n^{z_n} \to C_m$
$r_{m+1} : a_1^{x_1} a_2^{x_2} \dots a_n^{x_n} \to C_{m+1}$

$$\begin{aligned}
&P(\neg\exists \textit{ red rule}\{r_1, r_2, \dots, r_m, r_{m+1}\}) \\
&\quad = P(\neg\exists \textit{ red rule}\{r_1, r_2, \dots, r_m\}) \times P(\neg\exists \textit{ red rule}\{r_1, r_{m+1}\}) \\
&\qquad \times P(\neg\exists \textit{ red rule}\{r_2, r_{m+1}\}) \times \dots \times P(\neg\exists \textit{ red rule}\{r_m, r_{m+1}\}) \\
&\quad = \left(\frac{2^n - 2}{2^n}\right)^M \times \left(\frac{2^n - 2}{2^n}\right) \times \overset{m}{\dots} \times \left(\frac{2^n - 2}{2^n}\right) \\
&\quad = \left(\frac{2^n - 2}{2^n}\right)^{M+m} = \left(\frac{2^n - 2}{2^n}\right)^{(m!/2!(m-2)!)+m} \\
&\quad = \left(\frac{2^n - 2}{2^n}\right)^{(m(m-1)/2)+m} \Rightarrow \left(\frac{2^n - 2}{2^n}\right)^{(2m+m^2-m)/2} \\
&\quad = \left(\frac{2^n - 2}{2^n}\right)^{(m^2+m)/2} = \left(\frac{2^n - 2}{2^n}\right)^{m(m+1)/2} \\
&\quad = \left(\frac{2^n - 2}{2^n}\right)^{((m+1)m(m-1)!)/(2(m-1)!)} \\
&\quad = \left(\frac{2^n - 2}{2^n}\right)^{(m+1)!/2(m-1)!} = \left(\frac{2^n - 2}{2^n}\right)^{M+1}
\end{aligned}$$

□

In a region of our P-system, given a set of evolution rules and an alphabet of objects, it is always possible to calculate the probability of having redundant rules. If the probability is high, it is worthwhile to use this method; It is highly probable that the computational complexity and number of operations are reduced in comparison to the *fast linear algorithm*.

Example. $U = \{a\}$, $R(U,T) = \{r_1, r_2, r_3\}$

$r_1 = a^5 \to C_1$

$r_2 = a^2 \to C_2$

$r_3 = a^3 \to C_3$

r_1 *and* r_3 are redundant rules. The only non-redundant rule is r_2. This happens because the multiplicity within its antecedent is the minimum of all the objects multiplicities within the antecedent of the remainder of rules.

When the number of objects is greater than 1, there is no guarantee that Redundant evolution rules exist. However, as the number of evolution rules increases, so do the chances of having redundant rules.

5. Paun's biological model

When we modify the methods by including new strategies, we have to make sure that this does not alter the biological model.

According to Paun's model, the election of the evolution rules to be applied must be non-deterministic. The evolution rules are applied in a parallel and non-deterministic manner (Paun, 1998).

Depending on the evolution rules we choose, a different extinguished or maximal multiset can be obtained. However, our method has to consider all possible extinguished multisets when returned. This means that, by following our method, any possible extinguished multiset can be returned.

The current methods assure this fact. By adding our own strategy, we must prove that this does not alter the model and, therefore, we are still able to return any possible extinguished multiset.

Lemma 3. *In a given region, Let* $U=\{a_i|1\leq i\leq n\}$ *be an alphabet of objects, let* $m=\{(a_i,u_i)|a_i\in U\ u_i\in N\ 0<i\leq n\}$ *be the existing multiset before applying any evolution rule and let* $R(U,T)$ *be a multiset of evolution rules within that region.* $|R(U,T)|=k$. *We can obtain any possible extinguished multiset by implementing the redundant rules method.*

Proof. All methods such as the fast linear algorithm, the elimination of active rules algorithm, the step by step algorithm etc, has been proved to generate any possible extinguished multiset of objects. Now, we are going to prove that implementing redundant rules method can generate any possible maximal multiset as well.

Let us suppose that this is not true. There is an extinguished multiset we cannot obtain by using the Redundant rules method.

Let $m_1=\{(a_i,u_i)|a_i\in U\ u_i\in N\ 0<i\leq n\}$ be the extinguished multiset generated by the fast linear algorithm and impossible to obtain through the *redundant rules method*.

In order to obtain m_1 it is necessary to apply the rules contained in $R(U,T)$ a certain number of times.

Let μ_i be the number that indicates the number of times every rule has been applied in order to obtain m_1. i.e. r_i has been applied μ_i times. We define μ_i as the number of times that rule r_i has been applied in order to obtain m_1 $0\leq i\leq k$.

$\mu_i=\mu'_i+\mu''_i$ where μ'_i is the number of times that the rule r_i is applied in the first round (a random number between 0 and its applicability benchmark) and μ_i'' is the number of times that r_i is applied in the second round which is the new applicability benchmark. The last application occurs when the extinguished multiset has not been found at the end of the first round. □

Redundant rules method: Let $m_2=\{(a_i,u'_i)|a_i\in U\ u'_i\in N\ 0<i\leq n\}$ be a maximal multiset. We define η_i as the number of times that rule r_i has been applied in order to obtain m_2 ($\forall i 0\leq i\leq k$.

If r_i is not redundant then $\eta_i=\eta'_i+\eta''_i$ where η'_i is the number of times that the rule r_i is applied in the first round (a random number between 0 and its applicability benchmark) and η''_i is the number of times that the rule is applied in the second round which is the new maximum applicability benchmark. If (1) is fullfilled then the extinguished multiset has been found at the end of the first round, and then η''_i is always 0.

If r_i is Redundant, then $\eta_i=\eta'_i+\eta''_i$ where η'_i is the number of times that the rule r_i is applied in the first round (a random number between 0 and its applicability benchmark) and $\eta''_i=0$ because we do not consider the rule r_i in the second round.

$\eta_i=\mu_i\Leftrightarrow\eta'_i+\eta''_i=\mu'_i+\mu''_i$.

1. r_i is not Redundant $\Rightarrow\eta'_i+\eta''_i=\mu'_i+\mu''_i\Rightarrow\eta'_i=u'_i$ and $\eta''_i=u''_i$ is a possible solution $\Rightarrow m_1=m_2$,
2. r_i is Redundant $\Rightarrow\eta'_i+\eta''_i=\mu'_i+\mu''_i\Rightarrow\eta'_i=\mu'_i+\mu''_i\Rightarrow m_1=m_2$. This can always happens as:
   (a) η'_i is a random number between 0 and applicability benchmark (r_i),
   (b) $0\leq\mu'_i+\mu''_i\leq$ applicability benchmark (r_i).

Thus, $m_1=m_2$ as long as the equations above are fulfilled, therefore, proving that any maximal multiset can be obtained by using the *redundant rules method*.

In this section we describe a detailed example. This example shows the steps taken to obtain extinguished multisets.

Example. $U=\{a_1,a_2\}$

We randomly reorder the rules within $R(U,T)$ to assure non-determinism in our rules selection.

$R(U,T)=\{r_1,r_2,r_3\}$

Two objects, three rules $\Rightarrow P(US_{R(U,T)}\neq\phi)=1-\frac{1}{8}=\frac{7}{8}$.

$r_1=a_1^4a_2^4\rightarrow C_1$

$r_2=a_1^2a_2^3\rightarrow C_2$

$r_3=a_1^5a_2^7\rightarrow C_3$

Initial Multiset $m=\{(a_1,14),(a_2,15)\}$.

Fast linear algorithm (FLA) First round: We apply r_1 a random number of times between 0 and its maximum applicability benchmark i.e. random {0,2} Let us say 0. We apply r_2 a random number of times between 0 and its maximum applicability benchmark i.e. random {0,3}. Let us say 1.

After applying r_2 once, the resulting multiset is $\{(a_1,12),(a_2,12)\}$. Finally, we apply the rule r_3. In this case we apply this rule a number of times equal to its maximal applicability benchmark=1.

Thus the resulting multiset is $\{(a_1,7),(a_2,5)\}$. This resulting multiset is not maximal because rules can still be applied. According to the fast linear algorithm, we must start a new round.

Second round: In this phase we must apply all of the rules to make sure that the resulting multiset is extinguished.

We check r_1: Its maximal applicability benchmark is 1. By applying this rule, we obtain a resulting multiset $\{(a_1,3),(a_2,1)\}$.

We check r_2: Its maximal applicability benchmark is 0; therefore, we do not apply it.

We check r_3: Its maximal applicability benchmark is 0 therefore, we do not apply it.

The resulting multiset $\{(a_1,3),(a_2,1)\}$ is an extinguished multiset. As a last step, we must add up the times we have applied r_1,r_2,r_3 in the two rounds:

Rule	Firstround	Secondround	Total
r_1	0	1	1
r_2	1	0	1
r_3	1	0	1

In this case the iterations are $2\times|R(U,T)|-1=5$.

Redundant rules method:

As there is a high probability of finding redundant rules, we decide to try this method.

Following the steps we propose, we will work with the same method until the new round is started.

In the second round, we ignore the redundant rules. In this case we skip checking r_1 and r_3 as they are redundant, according to our definition.

Thus, the following steps are taken:

First round: same as FLA, therefore, the resulting multiset is: $\{(a_1,7),(a_2,5)\}$. This is not maximal which means that a new round must be implemented.

Second round: We check r_2; its maximal applicability benchmark is 1, therefore, we apply r_2 once and we obtain $\{(a_1,5),(a_2,2)\}$ which is maximal.

At the end, the number of times we have applied every rule is:

Rule	First round	Second round	Total
r_1	0	0	0
r_2	1	1	2
r_3	1	0	1

In this case, the number of iterations is: $2n-1=5$, n=number of active evolution rules.

Thus the number of iterations is $2\times|R(U,T)|-1-n=3$, n=number of redundant evolution rules.

We have obtained an extinguished multiset without having to analyze all of the rules.

This simple example has shown that it is possible to obtain maximal multisets in a faster, more efficient way by passing the redundant rules.

6. Integration of redundant rules method in the current algorithms

In this section we are going to modify (FLA). We can always determine the redundant rules within the compiling time of the algorithm.

FLA works as follows:

1. Multiset of evolution rules exists.
2. FLA receives the multiplicities of an initial multiset as input.
3. It calculates the active rules based on the multiplicities of the initial multiset.
4. It applies the active rules in a non-deterministic manner.
5. All rules belonging to the set of active rules "except one" are applied a random number of times between 0 and its maximal applicability benchmark.
6. FLA returns the maximal multiset and the times that each rule has been applied.

The algorithm expects a multiset of objects as input. What we propose is: The calculation of the redundant rules occurs before processing this input. We can do this during the compilation time of the algorithm. This calculation is performed before the algorithm begins its execution and, therefore, does not damage efficiency of the execution.

In order to calculate the active rules to be applied, we need the multiplicities of the initial multiset included in a region. However, we do not need those multiplicities to detect the redundant rules. This is the reason why we can detect the redundant rules before FLA starts its execution.

The modification we propose is just the inclusion of the technique as the precondition of the current algorithms:

PRE NotRed(U,T)$\leftarrow R(U,T)$// Redundant calculation
(1) m′$\leftarrow$m//multiplicities of initial multiset
(2) $m_R\leftarrow$MR(U)
(3) **FOR i=1 TO** | R |-1 **DO** // Phase 1 same FLA
(4) **BEGIN**
(5) $Max\leftarrow\Delta R[i]\lceil m'\rceil$//calculus max applicability benchmark for the rule R[i]
(6) **IF** ($Max\neq 0$) **THEN**
(7) **BEGIN**
(8) $K\leftarrow random(0,Max)$
(9) $m_R\leftarrow m_R+\{R[i]^K\}$
(10) $m'\leftarrow m'-input(R[i])\cdot K$ //*Subtracting objects from m′k times the antecedent of* r_i
(11) $Active[i]=K<Max$
(12) **END**
(13) **ELSE** $Active[i]=false$
(14) **END**
(15) $Active[|R|]=true$
(16) **FOR i**=|NotRed(U,T) |**DOWN TO 1 DO** //Phase 2 Checks only not redundant rules
(17) **IF** ($Active[i]$) **THEN**
(18) **BEGIN**
(19) $Max\leftarrow\Delta R[i]\lceil\omega'\rceil$
(20) $m_R\leftarrow m_R+\{R[i]^{Max}\}$
(21) $m'\leftarrow m'-input(R[i])\cdot Max$//*Subtracting objects from m′ max benchmark times the antecedent of* r_i
(22) **END**

6.1. Algorithm explanation

The algorithm we propose works as follows: In the "PRE" step, it calculates the redundant rules, which occurs before execution time. R is the number of evolution rules existing in a given region. In other words, it is the cardinal of $R(U,T)$. From the set of rules $R(U,T)$ we obtain notRed(U,T). This step checks the multiplicity of objects within evolution rules and picks up the non-redundant ones.

In step (1) we select the multiplicities of the objects belonging to the initial multiset. This is the multiset ready for the rules to be applied to it. The multiplicities of the multiset are the numbers we work with.

In step (2) we store all the evolution rules existing in the membrane region in the array mR. This variable provides the evolution rules to work with.

From (3) to (15) it works exactly the same as *FLA* (Gil et al., 2009). Basically, for each rule (iterative step), in the first round: calculation of the applicability benchmark of the rule r[i] (5). If the benchmark >0 (6) then we pick a random number k between 0 and the benchmark (8). We add the rule r[i] and the value "k" to the output multiset of evaluation rules (9). The multiset of objects gets reduced after applying the rule r[i] k times (10).

In (16) the loop only works with the non-redundant rules that are. This means that there are application rules which are not considered and, therefore, the number of operations decreases.

In (17) and (18) we select the (non-redundant) active rules. In (19) we calculate the maximum applicability benchmark. In (20) we include the rule and its benchmark to the set m_R In (21) The multiset m' of the membrane system becomes extinguished after applying the rule the maximum number of times. In (22) the algorithm finishes and returns m_R and m' (The rules applied, the number of times that have been applied and the extinguished multiset).

6.2. Analysis of performance

There are two loops in the algorithm. considering R is the number of evolution rules of the region, Computational complexity in terms of time for both methods is clearly

$\Theta(R)$

Below we show an estimation of the operations performed by the algorithm.

$\#operations\_per\_iteration\approx 3\cdot objects\ number(m)$

The worst case with the fast linear algorithm occurs when the maximal multiset has not been found at the end of round 1. This means that round 2 must occur. In this case, the number of

iterations executed in the worst case is

$$\#iterations = (|R| - 1) + |R| = 2, \cdot |R| - 1$$

Therefore, the number of operations executed in the worst case by the algorithm is

$$\#operations = (2 \cdot |R| - 1) \cdot 3 \cdot objects\ number\ (m)$$

Following we carry out an analysis of the operations performed when compared FLA and FLA+RED. The analysis focuses in the number of operations that each method performs.

We split up the analysis into two phases (two rounds): (a) *First round*: It occurs in FLA and FLA+RED in the same way.

Let us define N_1 as the number of operations occurring in an iteration in the first loop (3). These operations are:

- Calculating the max applicability benchmark for the rule r_i (5).
- Calculating a random number between 0 and the benchmark (8).
- Including the rule in the output set of rules m_r with the number of times it has been applied (9).
- Subtracting objects from the initial multiset m, k times the antecedent of r_i (10).

The loops iterates for each rule r_i. The total number of operations in the first loop is

$$\sum_{j=1}^{R} N_1 = R \cdot N_1$$

The first round ends up having the same number of operations for both methods.

(b) *Second round*: Let us define N_2 as the number of operations occurring in an iteration in the second loop (3). These operations are:

- Calculating the max applicability benchmark for the rule r_i (19).
- Including the rule in the output set of rules m_r linked to the max benchmark (20).
- Subtracting objects maximally as possible from the multiset m (21). (Applying the rule r_i maximally).

Let us suppose that n is the number of redundant rules

$$(n \leq R-1).$$

Then, the operations are:

FLA: As it occurs in the first round, the number of operations occurring in FLA depends on R; that is to say:

$$\sum_{j=1}^{R} N_2 = R \cdot N_2$$

FLA+RED: As seen previously FLA+RED becomes stronger when the number of rules increases and so it does chances for finding redundant rules. In this case the number of operations that FLA+RED performs depends on the redundant rules.

Let us define M as an arbitrary big number, R the number of evolution rules and n the number of redundant rules.

$$R \to M \Rightarrow n \to R-1$$

Thus

$$\sum_{j=1}^{R-n} N_2 = \sum_{j=1}^{R} N_2 - \sum_{j=1}^{n} N_2 = R \cdot N_2 - n \cdot N_2$$

$$\lim_{n \to R-1} \sum_{j=1}^{R-n} N_2 = N_2$$

Total amount of operations. The total number of operations in FLA is

$$\sum_{j=1}^{R} N_1 + \sum_{j=1}^{R} N_2 = \sum_{j=1}^{R} (N_1 + N_2) = R \cdot (N_1 + N_2)$$

The total number of operations in FLA+RED is

$$\sum_{j=1}^{R} N_1 + \sum_{j=1}^{R-n} N_2 = \sum_{j=1}^{R} (N_1 + N_2) - \sum_{j=1}^{n} N_2 = R \cdot (N_1 + N_2) - n \cdot N_2$$

Conclusions of the analysis. As proved before, when the number of rules are high, the number of redundant rules increases.

$$n \geq 0 \Rightarrow R \cdot (N_1 + N_2) \geq R \cdot (N_1 + N_2) - n \cdot N_2 \Rightarrow$$

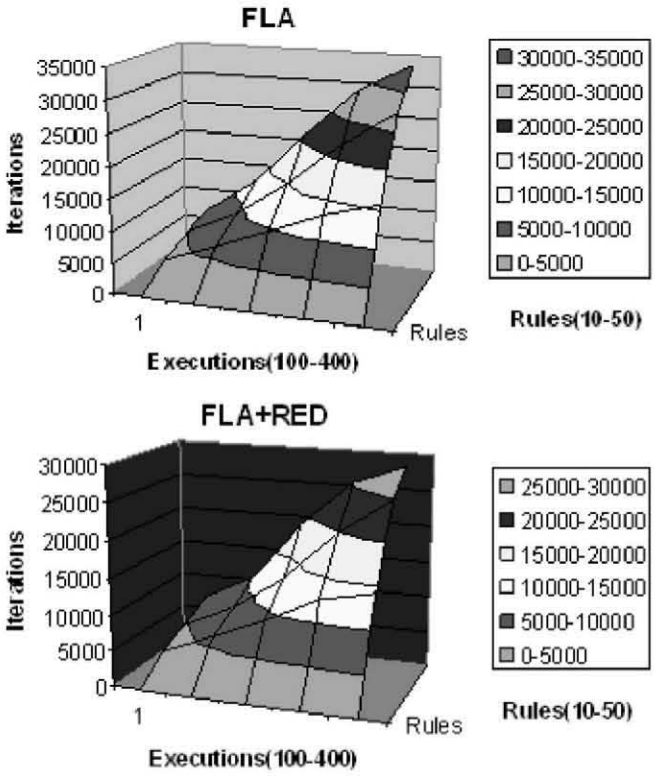
$$(FLAOperations) \geq (FLA + REDoperations).$$

It is obvious that the worst case has not improved when using FLA+RED. This occurs due to the fact that we are not always able to find redundant rules ($n=0$). In this case, the number of operations is the same in the two algorithms. In the best case ($n=R-1$) the number of operations performed in FLA+RED decreases dramatically.

$$n = R-1 \Rightarrow FLA + REDoperations = (R \cdot N_1) + N_2$$

This analysis shows that the number of operations in FLA+RED is less or equal to the number of operations in FLA in any case.

The average case clearly improves as the number of iterations is reduced. This implies that the number of total operations decreases as well. This is shown in the figures.



6.3. Comparative results

The results are compared with FLA (Gil et al., 2009), described in the literature as the fastest algorithm working on sequential devices. As the fast linear algorithm (FLA) has been proved to obtain the best results, we have analyzed the changes of the results when we have modified the FLA by adding the *redundant rules method* (FLA+RED). It is obvious that when removing certain rules from the algorithm,

the number of operations decrease and therefore the algorithm is even faster. In the first graph we focus on FLA and the second graph shows FLA updated with our method. The parameters are the number of rules and the executions of both algorithm. By looking the results, we certainly notice the reduction in the number of iterations when our technique is used.

The number of iterations for the fast linear algorithm has in the worst case a $2 \cdot |R| - 1$, being R the number of active evolution rules.

This means that the number of rules will determine how good the algorithm is in terms of time.

We have seen that, as the number of rules increases, so do the chances of finding redundant rules.

We have measured the number of iterations in both cases. As the number of iterations depends on the number of rules R, we consider R as one of the parameters of our study.

R has a range (0–50).

We have executed the algorithms a number of times within a range (100–400).

Based on the number of executions and the number of rules as parameters, we have obtained the number of iterations.

The first graph shows the number of iterations occurring for the fast linear algorithm (FLA). We have combined the number of active evolution rules and the number of times the algorithm is executed.

The second graph shows the number of iterations occurring on the fast linear algorithm modified by the *redundant rules method* (FLA+RED) when combining the number of active evolution rules and the number of times the algorithm is executed.

It is noticeable that the distance between the both graphs increases, corresponding to the increase of the number of rules. When there fewer rules, the difference is not as significant.

However, the difference becomes larger when we increase the number of evolution rules. This also occurs due to the increase of chances of finding redundant evolution rules. Although the best and worst cases have not been altered, the average case has clearly been improved.

7. Conclusions

Solving NP complete problems is presently recognized as a well-known challenge. Parallel and distributed models are obtaining promising results on this matter. In particular, Transition P-systems have been proven to be effective and efficient when solving these kinds of problems.

We propose a technique that performs some updates during the implementation of Transition P-systems. This technique is focused on the evolution rules application phase. We have achieved optimal results with this updated technique.

We have seen that when using this technique, the execution time of current algorithms is reduced; therefore, obtaining an improvement in the current implementations of Transition P-systems.

Furthermore, we have proved that the technique does not alter the nature of the Transition P-system and, therefore, Paun's model remains intact.

This is an important step in the process of solving complex problems. The inclusion of this technique with the current implementation ensures times reduction.

Obviously, execution times are affected by the election of hardware in which the Transition P-systems must be implemented. However, when the hardware implementations have been selected, using this technique ensures optimal results, as shown.

In this paper we focus on incorporating this method with the fast linear algorithm, as this is the one that has obtained the best results this far, in general terms. However, as this method bypasses the analysis of certain rules, it can be incorporated with other strategies in the same methodology presented in this paper.

Curiously enough, as the number of rules has been the key factor to damage performance in all known algorithms, it is also the key factor for taking advantage of this method.

We prove that, by increasing the number of rules, the chances for finding redundant rules also increases, therefore, improving performance throughout the entire implementation.

In conclusion, it is the technique proposal, not the algorithm, that is the key to this paper. This means that the technique can be used in P-systems technology to improve performance regardless of the hardware/software used for implementation.

References

Arroyo, F., Luengo, C., Baranda, A.V., Mingo, L.F., 2003. A software simulation of transition P-systems in Haskell. International Workshop Membrane Computing, Curtea de Arges (Romania), August 2002, vol. 2597. Springer-Verlag, Berlin, pp. 19–32.

Arteta, A., Fernandez, L., Gil, J., 2008. Algorithm for application of evolution rules based on linear diophantine equations Synasc. IEEE Timisoara Romania, September.

Ciobanu, G., Paraschiv, D., 2002. Membrane software. A P-system simulator. In: Pre-Proceedings of Workshop on Membrane Computing, Curtea de Arges, Romania, August 2001, Technical Report 17/01 of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, 2001, 45–50 and Fundamenta Informaticae, vol 49, 1–3, 2002, pp. 61–66.

Ciobanu, G., Pan, P.L., Paun, Gh., Pérez-Gimenez, M., P-systems with minimal parallelism, 2007.

Fernandez, L., Arroyo, F., Castellanos, J., et al., 2006. New Algorithms for application of evolution rules based on applicability benchmarks. In: BIOCOMP 06, Las Vegas, USA.

Fernandez, L., Arroyo, F., Tejedor, J., Castellanos, J., 2006. Massively parallel algorithm for evolution rules application in transition P-system. In: Seventh Workshop on Membrane Computing, WMC7, Leiden, The Netherlands, July.

Gil, F.J., Fernandez, L., Arroyo, F., et al., Fast linear algorithm for active rules application in transition P-systems. i.TECH-2009, Varna, Bulgaria; Theoretical Computer Science, ISSN: 0304-3975, vol. 378, issue (1), 3 June 2007, pp. 117–130.

Gutiérrez-Naranjo, M.A., Perez-Jiménez, M.J., Riscos-Nuñez, A., 2007. On the degree of parallelism in membrane systems, Theoretical Computer Science, ISSN: 0304-3975, vol. 372, Issues 2–3, 15 March 2007, pp. 183–195.

Ibarra, O.H., 2006. On the computational power of 1-deterministic and sequential P-systems. Fundamenta Informaticae—Special Issue on Trajectories of Language Theory, vol. 73, Issue 1,2, July 2006, table of contents.

Narayanan, S., Rama, R., 2003. Breaking DES using P-systems. Journal of Theoretical Computer Science Archive 299 (1–3).

Paun, G., 1998. Computing with membranes. Journal of Computer and System Sciences, 61(2000), and Turku Center of Computer Science-TUCS Report n 208.

Paun, G., 2005. Membrane computing. Basic ideas results applications. in: Pre-Proceedings of First International Workshop on Theory and Application of P-systems, Timisoara, Romania, September, pp. 1–8.

Solving multidimensional 0–1 knapsack problem by P-systems with input and active membranes, 2005. Source Journal of Parallel and Distributed Computing archive vol. 65, Issue 12, December, pp. 1578–1584, Year of Publication:2005, ISSN:0743-7315.

Solving SAT by Symport/Antiport P-systems with Membrane Division, 2005. In: Proceedings of the ESF Exploratory Workshop on Cellular Computing (Complexity Aspects), Sevilla, Spain, 2005, Fénix Editora, Sevilla, pp. 1–6.

Solving NP-Complete Problems using P-systems with Active Membranes, 2001. In: MC 2000 Proceedings of the Second International Conference on Unconventional Models of Computation, ISBN:1-85233-415-0.

Tejedor, J., Fernandez, L., Arroyo, F., Gutiérrez, A., 2007. Algorithm of active rules elimination for evolution rules application. In: 8th WSEAS International Conference on Automation and Information, Vancouver, Canada, June.