Optimizing Evolution Rules Application and Communication Times in Membrane Systems Implementation

Jorge A. Tejedor, Abraham Gutiérrez, Luis Fernández, Fernando Arroyo, Ginés Bravo, and Sandra Gómez

Natural Computing Group
Escuela Universitaria de Informática, Universidad Politécnica de Madrid
Crta. de Valencia Km. 7, 28031 Madrid, Spain
{jtejedor,abraham,setillo,farroyo,gines,sgomez}@eui.upm.es
http://www.eui.upm.es

Abstract. Several published time analyses in P systems implementation have proved that there is a very strong relationship between communication and evolution rules application time in membranes of the system. This work shows how to optimize the evolution rule application and communication times using two complementary techniques: the improvement of evolution rules algorithms and the usage of compression schema.

On the one hand, this work uses the concepts of competitiveness relationship among active rules and competitiveness graph. For this, it takes into account the fact that some active rules in a membrane can consume disjoint object sets. Based on these concepts, we present a new evolution rules application algorithm that improves throughput of active rules elimination algorithms (sequential and parallel).

On the other hand, this work presents an algorithm for compressing information related to multisets and evolution rules, based on the assumption that algorithmic complexity of the operations performed over multisets, in evolution rules application algorithms, is determined by the representation of multiset information of these rules. This representation also affects the communication phase among membranes phase.

1 Introduction

Computation with membranes was introduced by Gheorghe Păun in 1998 [14] through a definition of transition P systems. This new computational paradigm is based on the observation of biochemical processes. The region defined by a membrane contains chemical elements (multisets) which are subject to chemical reactions (evolution rules) to produce other elements. Transition P systems are hierarchical, as the region defined by a membrane may contain other membranes. Multisets generated by evolution rules can be moved to adjacent membranes (parent and children). This multiset transfer feeds back into the system so that new products are consumed by further chemical reactions in the membranes.

G. Eleftherakis et al. (Eds.): WMC8 2007, LNCS 4860, pp. 298–319, 2007.

These systems perform computations through transition between two consecutive configurations. Each transition or evolution step goes through two steps: rules application and objects communication. First, the evolution rules are applied simultaneously to the multiset in each membrane. This process is performed by all membranes at the same time. Then, also simultaneously, all membranes communicate with their destinations.

Nowadays, membrane systems have been sufficiently characterized from a theoretical point of view. Their computational power has been settled – many variants are computationally complete. Among their most relevant characteristics appears the fact that they can solve non polynomial time problems in polynomial time, but this is achieved by the consumption of an exponential number of resources, in particular, the number of membranes that evolve in parallel.

There are available several membrane systems simulators, [**?**]. An overview of membrane computing software can be found in [2]. However, the way in which these models can be implemented is a persistent problem today, because "the next generation of simulators may be oriented to solve (at least partially) the problems of information storage and massive parallelism by using parallel language programming or by using multiprocessor computers" [2]. In this sense, information storage in membrane computation implementation is an example of Parkinson's First Law [13]: "storage and transmission requirements grow double than storage and transmission improvements".

The objectives of this paper are: first to present an improvement of the algorithm of active rules elimination [20] used in the rules application step, and second to present a compression algorithm that allows us to compress information without penalizing evolution time in P systems implementation.

To achieve this, the paper is structured as follows: first, related works are presented; then, the basic ideas of the active rules elimination algorithm are summarized, which is followed by a definition of the concept of competition between rules and the optimization of the algorithm is specified. Next sections present requirements for information compression in membrane systems and the proposed compression schema; then we analyze the obtained results for a set of tests for a well known P system. Finally, some conclusions are presented.

2 Related Works

The first works over massively parallel implementation for P systems started with Syropoulos [18] and Ciobanu [3] who in their distributed implementations of P systems use Java Remote Method Invocation (RMI) and the Message Passing Interface (MPI) respectively, on a cluster of PC connected by Ethernet. These authors do not carry out a detailed analysis about the importance of the time used during communication phase in the total time of P system evolution; although Ciobanu stated that "the response time of the program has been acceptable. There are however executions that could take a rather long time due to unexpected network congestion".

Recently, in [19] and [1] one presents analyses for distributed architectures that are technology independent, based on: the allocation of several membranes in the same processor; the use of proxies for communication among processors; and, token passing in the communication. These solutions avoid communication collisions, and reduce the number and length for communication among membranes. All these allow to obtain a better step evolution time than in others suggested architectures congested quickly by the network collisions when the number of membranes grows. Table 1 summarizes minimum times (T_{min}), optimal amount of processors and membranes located in each processor (P_{opt} and K_{opt}) to reach those minimum times, and the throughput obtained with corresponding processors and communications (Th_{proc} and Th_{com}) for the architecture. This analysis considers the P system number of membranes (M) that would evolve, the maximum time used by the slowest membrane in applying its rules (T_{apl}), and the maximum time used by the slowest membrane for communication (T_{com}).

Table 1. Distributed architecture parameters depending on application rules time (T_{apl}), communication time (T_{com}) and number of membranes (M)

Distributed Architecture [19]	Distributed Architecture [1]
$T_{\min} = 2\sqrt{2\ M\ T_{apl}\ T_{com}} - 2\,T_{com}$	$T_{\min} = 2\ \sqrt{M\ T_{apl}\ T_{com}} + T_{com}$
$P_{opt} = \sqrt{\frac{M\ T_{apl}}{2\ T_{com}}}$	$P_{opt} = \sqrt{\frac{M\ T_{apl}}{T_{com}}}$
$K_{opt} = \sqrt{\frac{2\,M\,T_{com}}{T_{apl}}}$	$K_{opt} = \sqrt{\frac{M\ T_{com}}{T_{apl}}}$
$Th_{proc} \sim 50\%$	$Th_{proc} \sim 50\%$
$Th_{com} \sim 50\%$	$Th_{com} \sim 100\%$

From all these, we may conclude that to reach minimum times over distributed architectures, there should be a balance between the time dedicated to evolution rules application and the time used for communication among membranes. So, depending on the existing relation between both times, and on the number of membranes in the P system, it is possible to determine the number of processors and the number of membranes that will be located at each of them to obtain the evolution minimum time.

The difference between these architectures lies on the different topology for the processors net and the policy for token passing. Thus, [1] reaches a throughput near to a 100% of the communication line, an increment in the parallelism level by the increment of a 40% in the processors amount involved in the architecture and a reduction to reach the 70% of the evolution time. Both works conclude that, for a specific number of membranes M, if it is possible that:

1. For T_{apl} to be N times faster, the number of membranes that would be hosted in a processor would be multiplied by $\sqrt{N}$, the number of required processors would be divided by the same factor and the time required to perform an evolution step would improve approximately with the same factor $\sqrt{N}$.

2. For T_{com} to be N times faster, the number of required processors would be multiplied by $\sqrt{N}$, the number of membranes that would be hosted in a processor would be divided by the same factor and the time required to perform an evolution step would improve approximately by the same factor $\sqrt{N}$.

Table 2 summarizes the importance of reducing T_{apl} and T_{com} over the distributed architectures parameters (minimum evolution time, optimum number of processors and optimum number of membranes per processor).

Table 2. Repercussion on distributed architecture parameters depending on T_{apl} and T_{com}

Conditions	$T_{\min}$	P_{opt}	K_{opt}
T_{apl} be N faster and T_{com} be equal	$\frac{T_{\min}}{\sqrt{N}}$	$\frac{P_{opt}}{\sqrt{N}}$	$K_{opt} \cdot \sqrt{N}$
T_{apl} be equal and T_{com} be N' faster	$\frac{T_{\min}}{\sqrt{N'}}$	$P_{opt} \cdot \sqrt{N'}$	$\frac{K_{opt}}{\sqrt{N'}}$
T_{apl} be N faster and T_{com} be N' faster	$\frac{T_{\min}}{\sqrt{N'} \cdot \sqrt{N}}$	$\frac{P_{opt} \cdot \sqrt{N'}}{\sqrt{N}}$	$\frac{P_{opt} \cdot \sqrt{N}}{\sqrt{N'}}$

These architectures need to know the time required to perform rules application to be able to optimally distribute membranes among processors. Analysis of the rules application algorithms published to date shows that only the execution time of active rules elimination algorithm [20](and its parallel version [8]) can be known beforehand. These two algorithms enable prior determination of the maximum execution time, since this value depends on the number of rules rather than on the cardinality of the multiset to which they are applied, as it is reported in other algorithms [3], [6], [7]. In addition, these algorithms are the fastest in their category (sequential and parallel).

This paper describes how to optimize evolution rule application and communication times by means of two strategies. On one hand, the active rules elimination algorithm modification taking into account the fact that some active rules in a membrane can consume disjoint object sets will improve the evolution rule application time. On the other hand, the use of a compression schema for multisets and evolution rules presented in membranes will improve both times.

3 Optimization of Active Rules Elimination Algorithm

This section first presents the main ideas about active rules elimination algorithm, second it introduces the concepts of competitive rules and competitiveness graph and finally, three optimizations of the algorithm are carried out taking into account the competitiveness among rules and the features of the algorithm itself.

3.1 Active Rules Elimination Algorithm

The general idea of this algorithm is to eliminate, one by one, the rules from the set of active rules. Each step of rule elimination performs two consecutive actions:

1. Iteratively, any rule other than that which is to be eliminated is applied for a randomly selected number of times in an interval from 0 to the maximum applicability threshold. This action ensures the non-determinism inherent to P systems.
2. The rule to be eliminated is applied a number of times which is equal to its maximum applicability threshold, thus making it no longer applicable and resulting in its disappearance from the set of active rules.

We assume that:

1. The object multiset to which active rules are applied is ω.
2. The active rules set is transformed to an indexed sequence R in which the order of rules is not relevant.
3. The object multiset resulting from application of active rules is ω'.
4. The multiset of applied rules that constitute the algorithm output is ω_R.
5. Operation $|R|$ determines the number of rules in the indexed sequence R.
6. Operation $\Delta_{R[Ind]} \lceil \omega' \rceil$ calculates the maximum applicability threshold of the rule $R[Ind]$ over ω'.
7. The operation $input(R[Ind]) \cdot K$ performs the scalar product of the antecedent of rules by a natural number.

The algorithm is as follows:

(1) $\omega' \leftarrow \omega$
(2) $\omega_R \leftarrow \emptyset_{M_{R(O,T)}}$
(3) **FOR** $Last = |R|$ **DOWNTO** 1
(4) **BEGIN**
(5) **FOR** $Ind = 1$ **TO** $Last - 1$ **DO**
(6) **BEGIN**
(7) $Max \leftarrow \Delta_{R[Ind]} \lceil \omega' \rceil$
(8) $K \leftarrow random(0, Max)$
(9) $\omega_R \leftarrow \omega_R + \left\{ R[Ind]^K \right\}$
(10) $\omega' \leftarrow \omega' - input(R[Ind]) \cdot K$
(11) **END**
(12) $Max \leftarrow \Delta_{R[Last]} \lceil \omega' \rceil$
(13) $\omega_R \leftarrow \omega_R + \left\{ R[Last]^{Max} \right\}$
(14) $\omega' \leftarrow \omega' - input(R[Last]) \cdot Max$
(15) **END**

Remember that if rule $R[i]$ is no longer applicable in the elimination step for $R[j]$, it is no longer necessary to perform the elimination step for $R[i]$, and thus the algorithm is greatly improved, as shown in [20].

In each iteration of the algorithm of actives rules elimination, the maximum applicability threshold of a rule is calculated and then the rule is applied. The number of iterations executed at worst is:

$$\#iterations = \sum_{i=1}^{q} i = \frac{q \cdot (q+1)}{2}$$

Let q be the cardinality of the indexed sequence of active rules. Thus, this algorithm allows one to know how long it takes to be executed in the worst case, with knowledge of the rules set of a membrane.

It is important to note that, in general, it is essential to perform the first action in each elimination step of a rule. This action is necessary to ensure that any possible result of the rules application to the multiset is produced by the algorithm. In case the action is not performed, the eliminated rule (applied as many times as the value of its maximum applicability threshold) may consume the objects necessary so that any other rule can be applied. However, the latter does not always occur and the first action in each elimination step can be simplified. For the sake of illustration, let us assume that the antecedents of a set of active rules are shown in Figure 1.

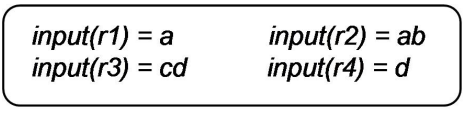

Fig. 1. Antecedent of the Active Rules Set

In this case, in the elimination step of the rule r_1 only the first action with the rule r_2 has to be taken, as r_1 and r_2 are the only rules with the object a in its antecedents. The same is the case with rules r_3 and r_4, as these two compete for the object d. Thus, taking into account the competition between rule antecedents, one can adjust the rule elimination algorithm to perform only 6 iterations in the worst case, rather than 10 (2 to eliminate r_1, 1 to eliminate r_2, 2 to eliminate r_3, 1 to eliminate r_4) as shown in Figure 2.

3.2 Definition of Competitiveness Between Rules

Let R be a set of active rules, $R = \{r_1, r_2, ..., r_q\}$ *with* $q > 0$, and let C be a binary relation defined over the set R such that

$$\forall x, y \in R,\ x \neq y \quad x\ C\ y \ \Leftrightarrow\ input(x) \cap input(y) \neq \emptyset$$

This binary relation can be represented by a non-directed graph $CG = (R, C)$ called a competitiveness graph, where the rules are related to each other if and

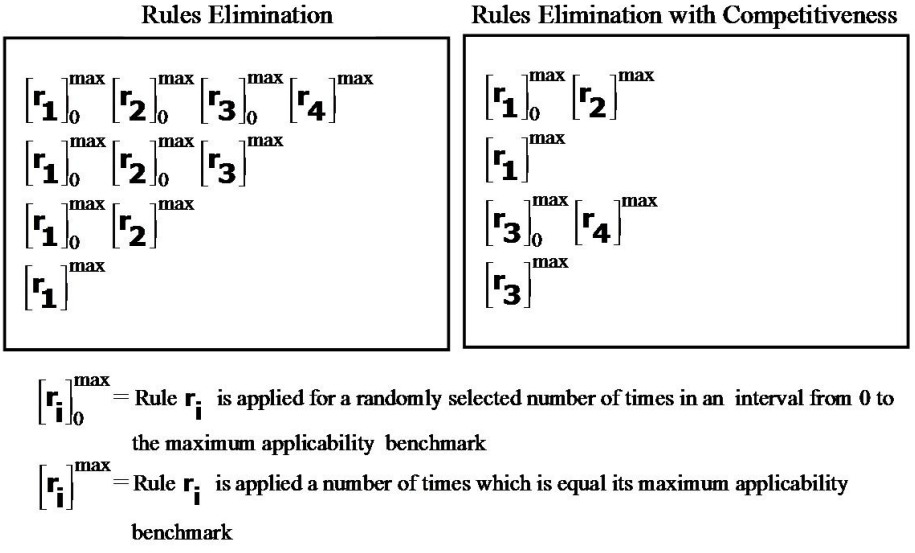

Fig. 2. Execution trace of Rules Elimination and Rules Elimination with competitiveness algorithms

only if their antecedents have an object in common. For example, given the rules inputs shown in Figure 3, the competitiveness graph generated by these rules taking into account the relation C will be as shown in Figure 4.

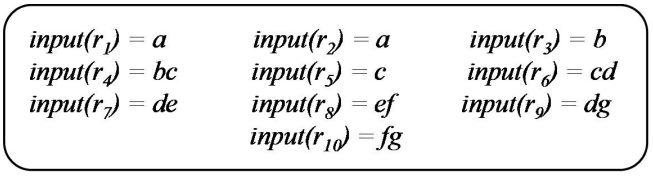

Fig. 3. Antecedents of an active rules set

Consider a competitiveness graph $CG = (R, C)$, a rule $x \in R$, and a set $R' \subseteq R$. The subgraph resulting from elimination of rule x is defined as

$$CSG = (R - \{x\}, C \cap R - \{x\} \times R - \{x\})$$

and the competitiveness subgraph induced by the subset R' is the graph

$$CSG = (R', C \cap R' \times R').$$

For a competitiveness graph $CG = (R, C)$, a competitiveness chain is defined as an ordered sequence of rules pertaining to R

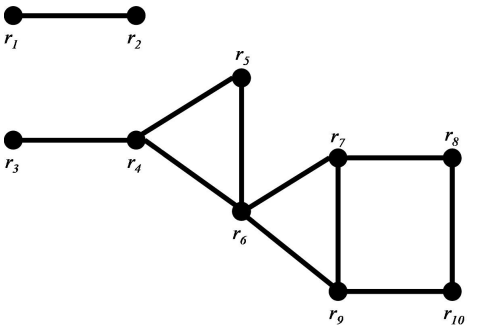

Fig. 4. Competitiveness graph

$$s_1,\ s_2,\ \ldots\ , s_n \quad s_i \in R,$$

satisfying:

$$s_i\, C\, s_{i+1} \quad \forall i \in \{1,\ \ldots,\ n-1\}$$

By definition, there is always a competitiveness chain composed of a single rule.

For a competitiveness graph $CG = (R, C)$, the accessible rule relation (A) is defined as:

$$x, y \in R \quad x\ A\ y \ \Leftrightarrow\ \exists \text{ a competitiveness chain } \ s_1, ..., s_n | s_1 = x \wedge s_n = y$$

This is an equivalence relation which divides the rule set R into equivalence classes.

Let E be an equivalence class produced by A. The connected component of CG is defined as the graph induced by the nodes pertaining to the equivalence class E. Then CG is called connected if and only if it has a connected component.

For a competitiveness graph $CG = (R, C)$ and a rule $x \in R$, it is said that x is an articulation of CG if and only if the subgraph resulting from the elimination of rule x has more connected components than CG.

3.3 The Algorithm Based on Rules Competitiveness

Based on the rules competitiveness relation, one can improve the algorithm of elimination of active rules. To do this, an analysis must be made of the evolution rules of each membrane prior to P system evolution. The analysis will determine the order of active rules elimination and what rules set are used in the first action of each elimination step of a given rule. The following optimizations can be made of the algorithm of rule elimination:

First optimization. The idea of this optimization is based on the fact that in the elimination step of a rule, the first action of the algorithm must be applied to the rules in the same connected component of the competitiveness graph. This

can be done because the antecedents of rules in different connected components do not compete for common objects of the multiset.

The analysis prior to the execution of each P system calculates the competitiveness graph of each membrane. Then the connected components of the graph are calculated. The algorithm of active rule elimination will be applied independently to the rules of each of the connected components, with no need for any change in its codification.

In the worse case of the example in Figure 4, the sequential version of this algorithm will need to perform 3 iterations in the connected component consisting of the rules $\{r_1,\ r_2\}$ and 36 iterations in the connected component consisting of the rules $\{r_3,\ r_4,\ r_5,\ r_6,\ r_7,\ r_8,\ r_9,\ r_{10}\}$. Therefore, this example has gone from 55 iterations in the worst case of the algorithm of active rules elimination to 39 iterations (Figure 5), that is, it has been reduced by 71% compared to the active rules elimination algorithm.

Making a parallel version of the algorithm is quite simple. One needs only to apply the algorithm of active rules elimination in parallel to the rules of each connected component on the competitiveness graph. The parallel version would require only 36 iterations ($maximum(36, 3)$) in the worst case, as shown in Figure 6), therefore it has been reduced by 65% compared to the active rules elimination algorithm.

Second optimization. This optimization is applied in each connected component of the competitiveness graph. If the competitiveness graph of a membrane has articulations, the algorithm can be used to eliminate these rules first and

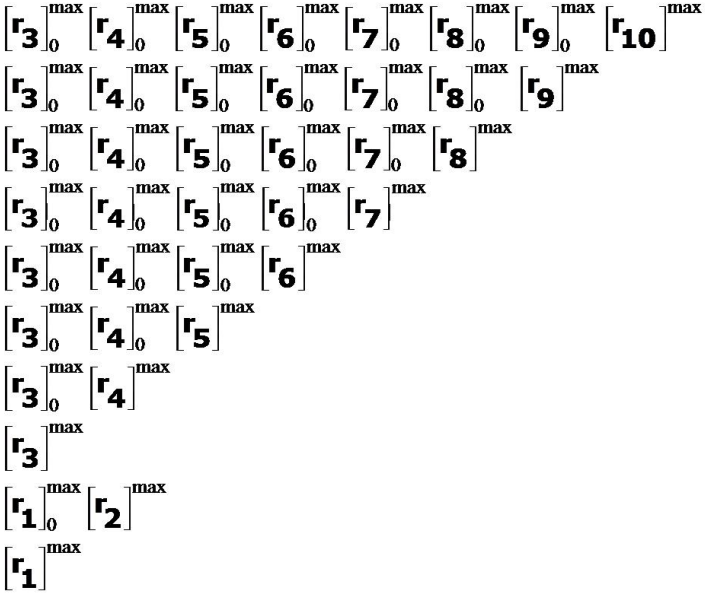

Fig. 5. Execution trace of 1^{st} sequential optimization

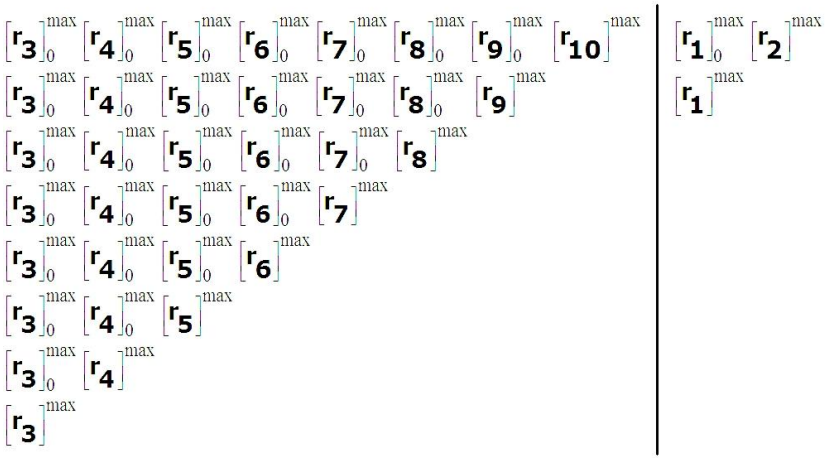

Fig. 6. Execution trace of 1^{st} parallel optimization

cause the appearance of new connected components. Thus, if rule r_6 is eliminated in our example (Figure 4) the connected component splits in two: the one composed of $\{r_3,\ r_4,\ r_5\}$ and the one composed of $\{r_7,\ r_8,\ r_9,\ r_{10}\}$.

When a connected component has no articulations, elimination of more than one rule can break it into more than one connected component. Continuing with the example we have proposed, if we first remove from connected component $\{r_7,\ r_8,\ r_9,\ r_{10}\}$ rules r_7 and r_{10} in two elimination steps, it then splits into two connected components consisting of the rules r_8 and r_9, respectively.

To perform this optimization, a slight change must be made in the sequential algorithm of active rules elimination. Now, each step of elimination of a rule must eliminate a specific rule. Moreover, there is a certain partial order in the elimination steps of a rule. Whereas order is irrelevant in previous versions of the active rules elimination algorithm, it is decisive in this version. The set of rules used and the rule being eliminated in each elimination step is calculated for each membrane in the analysis prior to the evolution of the P system; as a result, the calculation does not penalize the execution time of the algorithm.

Figure 7 shows the order in which evolution rules are eliminated and the set of rules used in each elimination step for the example in Figure 4. The number of iterations of this algorithm in the worst case is 25, so it has been reduced by 45% compared to the active rules elimination algorithm.

The parallel version of the algorithm involves applying the sequential version to each of the connected components that are either in the original competitiveness graph or that are generated as a result of the elimination of a rule.

The execution trace of the parallel algorithm used with the set of rules of the example in Figure 4 is shown in Figure 8. It may be noted that the number of iterations in the worst case is 16 ($maximum(8,2) + maximum(3,4,1) + maximum(1,1,3) + maximum(1,1)$) using 5 processes. Hence, the number of iterations is reduced by 29% compared to the active rules elimination algorithm.

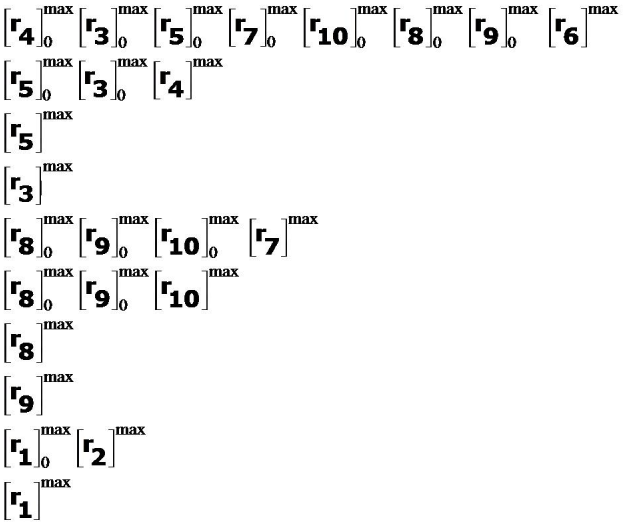

Fig. 7. Execution trace of 2^{nd} sequential optimization

Third optimization. This last optimization is based on an analysis of the execution trace of the 2^{nd} optimization. It can occasionally be observed that the elimination step of one rule r_j also eliminates one or more additional rules r_i. This can occur either because r_i is applied a number of times that coincides with the maximum applicability threshold, or the rules applied prior to r_i consume the objects needed to continue being active. This can be used in three ways to improve the execution time of the algorithm:

1. There is no need to execute the elimination step of the rule r_i eliminated in a previous step. Bearing in mind the execution trace in Figure 7, if the elimination step of rule r_6 also eliminates rule r_4, then it would no longer be necessary to execute the elimination step of r_4, thus allowing execution of the algorithm to save 3 iterations.
2. The rule r_i is not to be applied in the elimination steps of subsequent rules. Bearing in mind the execution trace in Figure 7, if the elimination step of the rule r_6 also eliminates rule r_8, it is therefore unnecessary in elimination steps of the rules r_7 and r_{10} to try to apply r_8, thus allowing execution of the algorithm to save 2 iterations.
3. Elimination of the rule r_i causes a change in the composition and order of the subsequent elimination steps. Keeping in mind the execution trace in Figure 7, if the elimination step of rule r_6 also eliminates rule r_8, then it is beneficial for the execution of the algorithm that r_9 be the next rule to be eliminated. This is the case because once r_6, r_8 and r_9 have been eliminated, r_7 and r_{10} can be eliminated in a single iteration in their elimination step since they do not share objects. Here, 3 iterations would be saved.

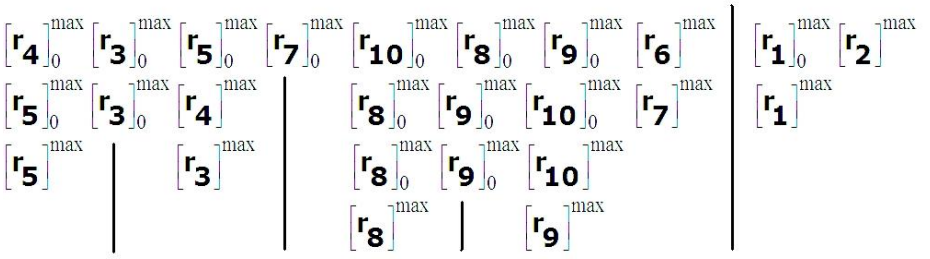

Fig. 8. Execution trace of 2^{nd} parallel optimization

To implement this optimization, a determination is necessary of what rules continue to be active whenever an elimination step is performed, and this information is used to calculate the next optimal elimination step to be taken. Logically, calculation of the next optimal elimination step would severely penalize the execution time of the algorithm. Hence, a different solution must be sought. This solution involves making an analysis prior to the execution of each P system, in which we can calculate all the possible active rule sets and assign them the next optimal step of rule elimination. All this information would be reflected in a director graph of the algorithm, the definition of which is as follows.

Let R be a set of active rules. The director graph of the algorithm of rule application is composed of a triple $DG = (Q, A, T)$ where:

1. Q is the node set of the graph, composed of a subset of parts of R, that is: $\forall q \in Q, , q \in \mathcal{P}(R)$
2. A is a correspondence whose initial set is Q and whose final set is a set of sequences of rules composed of rules from the origin element of Q. Thus, each set of active rules has one or more sequences of rules. Each sequence of rules indicates the order in which elimination step rules are applied. So, a state can have several elimination steps associated in the analysis prior the evolution of each P system.
   $A : Q \rightarrow E$ *where* E *is the set of possible sequences with elements in* Q
3. T is a set of transitions. Each transition is composed of a triad $\langle q_i, A(q_i), q_f \rangle$ where q_i, $q_f \in Q$ are the initial and final state, respectively, of the transition and $A(q_i)$ are the elimination step (s) of rules associated to state q_i, which, after being executed, means that active rules are those of state q_f.

The execution of the sequential algorithm of application of competitive rules will involve making a loop that ends when it reaches a state with no active rules. In each iteration, there are three steps:

1. The elimination steps associated to the state are executed.
2. Active rules are calculated.
3. The state represented by active rules is transited.

Execution of the parallel algorithm of application of competitive rules will be similar to the sequential one. The difference is that execution of several

elimination steps associated to a state is performed in a parallel way. At worst, the third optimization performs the same iterations as the second optimization.

4 Multisets and Rules Compression

Algorithmic complexity of the operations over multisets used in the evolution rules application algorithms is determined by the representation of multiset information of these rules. This representation also affects the communication between membranes. So the use of a suitable compression schema can have a positive influence over the reduction of evolution rule application and communication times.

4.1 Compression Requirements

First, unlike other environments, where it is admissible a non lossless information system (i.e., multimedia contents transmission), in our environment it is essential that our compression system has no information loss.

Almost all the compression methods require two phases: the first one for analysis followed by a second one for conversion. First, an initial analysis of the information is done to identify repeated strings. From this analysis, an equivalences table is created to assign short codes to those strings. In a second phase, information is transformed using equivalent codes for repeated strings. Besides, this table is required with the information for its future compression/decompression. On the other hand, we must realize that a higher compression without any information loss will take more processing time. *Bitrate* is always variable and it is used mainly in text compression [17]. Because all of this, in spite of the fact that there are compression systems that are able to reach entropy limit - highest limit for data compression (e.g., Huffman codes) - they are not the ideal candidates for our system because of the following reasons:

1. Table storage will increase the needs for memory resources and would decrease compression goal.
2. Time for the phase of evolution rules application is penalized with compression/decompression processes when accessing compressed information on the P system. This reduces parallelism level from distributed systems and increases evolution time.
3. And also, despite of the fact that communication phase time will be reduced because a lowest amount of information is transmitted, this will be counteracted by the time needed for decompression in the destination.

In this way compression schema for information from P system should accomplish the following requirements:

1. there should be no information loss;
2. it should use the lowest amount of space for storage and transmission;
3. it should not penalize time for rules application phase and communication among membranes while processing compressed information. Thus, this means that the system should:

(a) encode information for a direct manipulation in both phases without having to use coding/decoding processes,
(b) do the compression in a previous stage to the P system evolution,
(c) therefore, abandon entropy limit to be able to maintain parallelism level and evolution time reached in previous research works.

4.2 Compression Schema

The second goal of this work pretends to compress the information from multisets that are present in regions and rules antecedents and consequents from each rule of a P system. But it does not address the compression of another kind of information, such as priorities, membrane targets in rule consequents nor dissolving rule capability.

Representation for multisets information in related literature is Parikh's vector [4]. Data compression is directly associated with its representation. A compression schema is presented here in three consecutive steps beginning with Parikh's vector codification over the P system alphabet.

Parikh's vector over P system alphabet. Each region of a membrane can potentially host an unlimited number of objects, represented by the symbols from a given alphabet V. We use V^* to denote the set of all strings over the alphabet V (we consider only finite alphabets). For $a \in V$ and $x \in V^*$ we denote by $|x|_a$ the number of occurrences of a in x. Then, for $V = \{a_1, ..., a_n\}$, the Parikh vector associated with V is the mapping on V* denoted by $\psi_V(x) = (|x|_{a_i}, \cdots, |x|_{a_n})$ for each x $\in V^*$. The byte's order reflects the order of the objects within the alphabet and consequently, the position directly indicates which symbol's multiplicity is being stored.

Parikh's vector for each membrane's alphabet. First step in compression considers only the alphabet subset that may exist in each of the regions for the membrane system, whatever are the possible configurations for the P system evolution. This subset may be calculated by a static analysis, previous to P system evolution time. Rules to consider when determining each membrane's alphabet in a given P system are:

1. Every object present at the region for the P system initial configuration belongs to its membrane's alphabet.
2. Every object present at the consequent for a membrane's evolution rule with target "here" belongs to its membrane's alphabet.
3. Every object present at the consequent for a membrane's evolution rule with target "in" to another membrane, belongs to the target membrane's alphabet.
4. Every object present at the consequent for a membrane's evolution rule with target "out", belongs to its father alphabet.
5. Every object present at any membrane alphabet with an evolution rule with dissolution capability belongs to its father alphabet.

Parikh's vector without null values. Next compression step is an alteration over the *Run Length Encoding (RLE)* algorithm [11], used mainly to compress FAX transmissions. In this lossless codification, data sequences with same value (usually zeros) are stored as a unique value plus its count. RLE compression factor is, approximately:

$$\frac{E\,(X)}{E\,\{\log_2 x\}}$$

where X is a discrete random variable that represents the number of successive zeros between two ones and E(X) is its expected value (average). Compression value stands between 20% and 30%.

In our case, what we pretend is to eliminate all the null values in Parikh's vector, that is, to eliminate all the references to the alphabet elements in a membrane that do not appear in its multiset. This information may be considered as redundant because it may be obtained from the new coded information. In a formal way, let $V = \{a_1, a_2, \ldots, a_n\}$ be an ordered finite alphabet, for $x \in V^*$ the encoded Parikh vector associated with V is defined by $\Psi_V^E(x) = \{(|x|_{a_i}, i) \quad | \quad |x|_{a_i} \neq 0\}$.

At this point we should remark an important factor that is the variable or constant character for the multiset multiplicities. For the cases with multisets present at a membrane region, independently from the initial configuration, its multiplicities values are variable depending on the evolution that takes the membrane system in a non deterministic way. On the other hand, for the cases with multisets present at the evolution rules antecedents and consequents, its multiplicities values are constant and known previously to the P system evolution.

According to this situation, the compression second step encodes without null values just the information that belongs to constant multisets present at evolution rules. Thus, we get a more compressed (and lossless) representation. The reason that does this representation possible is the fact that the absence of these null values multiplicities does not affect none of the multisets operations (addition, subtraction, applicability, scalar product, ...).

Storage unit compression. Last compression step concerns storage unit size for each of the P system information values. Depending on the storage unit size (measured in bits), we will be able to codify a greater or smaller range of values. In membrane computing, that does not allow negative values, given t bits for the storage unit, the range for possible values will vary from 0 to $2^t - 1$.

In this section, we will have to take into account multisets present in the regions separately from the ones present in evolution rules. For the first case, storage unit size depends on the value range we want to reach during evolution without having an overflow. Instead of this, for the second case, we have to take into account, as it was shown in previous sections, that each membrane's ordered alphabet and their multiplicities are constant. Thus, an analysis previous to the P system evolution allows calculating the value ranges that are present in constant multisets for evolution rules and, so, the size that is needed to get their codification:

1. value range for multiplicities present at the antecedents and consequents for each membrane,
2. value range for Parikh's vector positions over the ordered alphabet for each membrane.

5 Analysis of Results

In this section we present the analysis of results obtained from the improvement of evolution rules algorithms and the usage of compression schema. First we analyze the impact that algorithms and compression have over the time required for evolution rules application. Second, we analyze the impact that compression has over the time needed for communication among membranes. Afterward, we analyze the global impact over distributed architectures parameters: evolution minimum time, optimum number of processors and membranes in each processor. Finally, we analyze the schema compression itself and its benefits over viable architectures for P systems implementations.

For the analysis of the following sections, we examine some P systems considered in [14] and [15]. Table 3 describes these P Systems.

Table 3. P System used for testing

P System	*Task*	*Reference*
A.	First example	[14]
B.	Decidability: n mod k = 0	[14]
C.	Generating: $n^2, n \geq 1 (1^{st}$ version)	[14]
D.	Generating: $n^2, n \geq 1$ (2^{nd} version)	[15]

5.1 Impact Analysis for Evolution Rules Application Time

The algorithms for evolution rules application that have been referred to in this paper, are based upon a limited set of primitive operations over multisets. These are computation of: applicability, maximum applicability, antecedent/consequent addition and subtraction over its region multiset and the scalar product of an antecedent/consequent.

Table 4 shows the number of operations over multisets performed at worst by the algorithms:

- Actives rules elimination (ARE) [20]
- Sequential version of competitive rules with 2^{nd} optimization (SCR)
- Delimited massively parallel (DMP) [8]
- Parallel version of competitive rules with 2^{nd} optimization (PCR)

applied to P systems mentioned in Table 3.

Table 4. Number of operations over multisets performed at worst

	Sequential			**Parallel**		
P System	ARE	SCR	SCR/ARE	DMP	PCR	PCR/DMP
A.	18	12	66,6%	18	9	50%
B.	9	9	100%	10	9	100%
C.	18	12	66,6%	18	9	50%
D.	18	12	66,6%	18	9	50%
Average	**15.75**	**11.25**	**75%**	**16**	**9**	**60%**

According to these empirical values, SCR algorithm decreases its execution time 75% against ARE. Consequently, evolution rules application time will be approximately 1.33 times faster. With the parallel algorithms we have that PCR algorithm decreases its execution time 60% against DMP. Consequently, evolution rules application time will be approximately 1.67 times faster.

The algorithmic complexity of the operations over multisets used in the evolution rules application algorithms is determined by the representation of multiset information of these rules. At worst, using representation through Parikh's vector over the P system alphabet, complexity will be equal to the alphabet cardinality. On the other hand, using representation through the proposed compression schema, complexity at worst will be equal to the multiset support that is present at the evolution rule antecedent/consequent. Table 5 presents, for each of the P systems in table 3, its alphabet support, the average support for multisets present in its evolution rules and a percentage based relation among both cardinalities. Last row presents these cardinalities average values and their relation.

Table 5. Alphabet cardinality and support average from P systems of Table 3

P System	$\mid V \mid$	$\mid$ support(w) $\mid$	*%*
A.	4	1.05	26.3%
B.	4	1.50	37.5%
C.	5	1.13	22.6%
D.	5	1.13	22.6%
Average	**4.5**	**1.20**	**27.25%**

According to these empirical values, each of the primitive operations previously mentioned will decrease its execution time approximately until a 27.25%. Consequently, evolution rules application time will be approximately 3.67 times faster.

Taking into account both factors (decrease number of operations and decrease time per operation) we can affirm that the evolution rules application time with SRC algorithm will be approximately 4.88 times faster than ARE algorithm and PRC algorithm will be approximately 6.09 times faster than DMP algorithm.

5.2 Impact Analysis for Communication Time Among Membranes

Communication among membranes addresses submission of multisets present at the applied application rules consequents and, in case of dissolution, the region multiset itself. Depending on information representation, the data packet size to transmit will be smaller or bigger. Table 6 shows, for each of the P systems shown in Table 3, information compression rate for its communication for different storage units sizes. Last row presents compression rates average.

Table 6. Compression degree for communication units from P systems of Table 3

	Storage unit size			
P System	*8 bits*	*16 bits*	*32 bits*	*64 bits*
A.	55.0%	45.0%	40.0%	37.5%
B.	60.0%	50.0%	45.0%	42.5%
C.	54.0%	44.0%	39.0%	36.5%
D.	53.3%	44.4%	40.0%	37.8%
Average	**55.6%**	**45.9%**	**41.0%**	**38.6%**

According to these empirical values, a reduction until a 55.6% of the information to transmit among membranes may be reached in the worst case. Considering that communication is a linear process that depends upon the amount of information to transmit, communication time among membranes will be approximately 1.8 times faster.

5.3 Global Impact Analysis

At this point, we present an impact analysis of the optimization of the evolution rule application and communication times over distributed architecture parameters. In particular, we examine, following the criteria shown in Table 2, the implication in optimum number of processors and membranes per processor and minimum evolution time.

On one hand, time reduction for evolution rules application increases the number of membranes per processor. It also decreases the number of processors and evolution time.

Using the compression schema with SCR algorithm the following results are obtained: the evolution rules application time will be approximately 4,88 times faster so we get an increment of a 120.9% for membranes per processor and a reduction until a 45.26% for number of processors and for evolution time.

Using the compression schema with PCR algorithm the following results are obtained: the evolution rules application time will be approximately 6.09 times faster so we get an increment of a 146.78% for membranes per processor and a reduction until a 40.52% for number of processors and for evolution time.

On the other hand, time reduction for communication among membranes increases the number of processors. It also decreases the number of membranes

per processor and evolution time. According to the previous empirical data, from a communication time 1.80 times faster, we get, for the worst case, a 34.2% increment for number of processors and a reduction until a 74.5% for the number of membranes per processor and for evolution time.

Taking into account both factors, reduction for application and communication time, counteract their effects over the number of processors and the number of membranes per processor.

Using the compression schema with SCR algorithm the following results are obtained: we get a reduction of a **60.7%** for the number of processors, an increment of a **64.65%** for the number of membranes per processor and a reduction until a **33,74%** for evolution time.

Using the compression schema with PCR algorithm the following results are obtained: we get a reduction of a **54.36%** for the number of processors, an increment of a **83.93%** for the number of membranes per processor and a reduction until a **30.2%** for evolution time.

5.4 Compression Schema Analysis

Table 7 shows compression rates reached for each P system from Table 4, considering different storage unit sizes. Last row presents average compression rates for each storage size.

Table 7. Compression degree for P System from Table 3

	Storage unit size			
P System	*8 bits*	*16 bits*	*32 bits*	*64 bits*
A.	59.8%	37.8%	26.8%	21.3%
B.	75.0%	47.7%	34.1%	27.3%
C.	51.1%	32.1%	22.8%	18.1%
D.	52.2%	33.3%	23.9%	19.2%
Average compression degree	**59.5%**	**37.7%**	**26.9%**	**21.5%**

Considering the worst case for this compression schema (8 bits for all the storage units), at least, we reach a compression rate of 75,0%, which implies an increase of a 33,3% for memory availability to store information. For average compression rate (59,5%), it is reached an increase of 68,0% of memory availability. So we attenuate the storage problem for information in distributed architectures implemented with low storage capacity microcontrollers based technologies. Using this compression schema, it will be possible to allocate more membranes in each microcontroller and so, it will be possible to reach minimum times at the same time that we are maximizing resources.

On the other hand, it has to be underlined that the compression process is done by an analysis previous to the P system evolution. Thus, evolution is not

penalized with compression/decompression processes while phases for evolution rules application or communication among membranes.

6 Conclusions

Several published time analyses have proved that there is a very strong relationship between communication and evolution rules application times during membranes evolution in P systems implementation. This relation determines the number of membranes that can be allocated per processor in order to obtain the minimum evolution time for the P system. This work shows how to optimize the evolution rule application and communication times using two complementary techniques: the improvement of evolution rules algorithms and the usage of compression schema.

On the one hand, this paper introduces the concept of a competitiveness relationship among active rules. Based on this concept, a new way of parallelism has been opened toward the massively parallel character needed in rules application in P systems. Moreover, the sequential version of this algorithm performs a lower number of operations in execution than in other sequential algorithms published to date. Both the sequential and the parallel versions of the algorithm carry out a limited number of operations, thus allowing for prior knowledge of the execution time. This characteristic makes both versions of the proposed algorithm appropriate for being used in viable distributed architectures for P systems implementations. This is very important because architectures require determining the distribution of the number of membranes to be located in each processor of the architecture in order to obtain minimal evolution step times with minimal resources.

On the other hand, this work has presented a schema for compressing multisets and evolution rules for P system. The schema gets the possible highest compression level for the information without penalizing compression and decompression time with cost-consuming operations.The whole compression process is performed by mean of a previous static analysis to the P system execution. These facts, thanks to the chosen representation of information, improve the system performance reducing evolution rule application and communication times, what is very important because it implies a direct reduction of the evolution time in system execution.

An additional advantage obtained by the new algorithm and compression scheme is applied to hardware solutions and architectures based on microprocessors nets. In these cases the amount of information that has to be stored and transmitted is very important. In the first case, the main problem is due to the low storage capacity of microcontrollers. So, reducing this amount of information needed to represent a membrane, means to be able to extend the variety of problems that can be solved with this technology. In the second case, reducing the amount of information to transmit means to minimize the bottleneck in processor communication and so, increase the parallelism level.

References

1. Bravo, G., Fernández, L., Arroyo, F., Tejedor, J.: Master-Slave Parallel Architecture for Implementing P Systems. In: MCBE 2007. The 8th WSEAS International Conference on Mathematics and Computers in Business and Economics, Vancouver (Canada) (June 2007)
2. Ciobanu, G., Păun, G., Pérez-Jiménez, M. (eds.): Applications of Membrane Computing. Natural Computing Series. Springer, Heidelberg (2006)
3. Ciobanu, G., Wenyuan, G.: A P System running on a Cluster of Computers. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing. LNCS, vol. 2933, pp. 123–150. Springer, Heidelberg (2004)
4. Dassow, J.: Parikh Mapping and Iteration. In: Calude, C.S., Pun, G., Rozenberg, G., Salomaa, A. (eds.) Multiset Processing. LNCS, vol. 2235, pp. 85–102. Springer, Heidelberg (2001)
5. Fernández, L., Martínez, V.J., Arroyo, F., Mingo, L.F.: A Hardware Circuit for Selecting Active Rules in Transition P Systems. In: Workshop on Theory and Applications of P Systems, Timisoara (Romania) (September 2005)
6. Fernández, L., Arroyo, F., Castellanos, J., Tejedor, J.A., García, I.: New Algorithms for Application of Evolution Rules based on Applicability Benckmarks. In: BIOCOMP 2006. International Conference on Bioinformatics and Computational Biology, Las Vegas (EEUU) (July 2006)
7. Fernández, L., Arroyo, F., Tejedor, J.A., Castellanos, J.: Massively Parallel Algorithm for Evolution Rules Application in Transition P Systems. In: WMC 2006, pp. 337–343 (July 2006)
8. Gil, F.J., Fernández, L., Arroyo, F., Tejedor, J.A.: Delimited Massively Parallel Algorithm based on Rules Elimination for Application of Active Rules in Transition P Systems. In: i.TECH-2007. Fifth International Conference Information Research and Applications, Varna (Bulgary) (June 2007)
9. Gutiérrez, A., Fernández, L., Arroyo, F., Martínez, V.: Design of a Hardware Architecture based on Microcontrollers for the Implementation of Membrane Systems. In: SYNASC 2006. 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, (September 26-29, 2006), Timisoara, Romania (2006)
10. Gutiérrez, A., Fernández, L., Arroyo, F., Alonso, S.: Hardware and Software Architecture for Implementing Membrane Systems: A case of study to Transition P Systems. In: DNA13 2007. 13th International Meeting on DNA Computing Memphis, EEUU (June 4-8, 2007)
11. Lelewer, D.A., Hirschberg, D.S.: Data Compression. ACM Computing, 8902-0069 (1987)
12. Martínez, V., Fernández, L., Arroyo, F., Gutiérrez, A.: A Hardware Circuit for the Application of Active Rules in a Transition P Systems Region. In: Fourth Inter. Conference Information Research and Applications, (June 20-25, 2006), Bulgaria, Varna (2006)
13. Parkinson, C.N.: Parkinson's Law, or the Pursuit of Progress. John Murray (1957)
14. Păun, G.: Computing with Membranes. Journal of Computer and System Sciences 61 (2000), Turku Center of Computer Science-TUCS Report 208 (1998)
15. Păun, G., Rozenberg, G.: A Guide to Membrane Computing. Theoretical Computer Science 287, 73–100 (2000)
16. Petreska, B., Teuscher, C.: A Reconfigurable Hardware Membrane System. In: Alhazov, A., Martín-Vide, C., Paun, G. (eds.) Preproceedings of the Workshop on Membrane Computing, Tarragona, July 17-22 2003, pp. 343–355 (2003)

17. Salomon, D.: Data Compression: The Complete Reference. Springer, Heidelberg (2004)
18. Syropoulos, A., Mamatas, E.G., Allilomes, P.C., Sotiriades, K.T.: A Distributed Simulation of P Systems. In: Preproceedings of the Workshop on Membrane Computing, Tarragona, pp. 455–460 (2003)
19. Tejedor, J.A., Fernández, L., Arroyo, F., Bravo, G.: An Architecture for Attacking the Bottleneck Communication in P System. In: AROB 2007. XII International Symposium on Artificial Life and Robotics, Oita, JAPAN (January 25-27, 2007)
20. Tejedor, J.A., Fernández, L., Arroyo, F., Gutiérrez, A.: Algorithm of Active Rule Elimination for Application of Evolution Rules. In: MCBE 2007. The 8th WSEAS International Conference on Mathematics and Computers in Business and Economics, Vancouver (Canada) (June 2007)