MEMBRANE COMPUTING: NON DETERMINISTIC TECHNIQUE TO CALCULATE EXTINGUISHED MULTISETS OF OBJECTS.

Alberto Arteta, Angel Castellanos, Ana Martinez

***Abstract**: Within the membrane computing research field, there are many papers about software simulations and a few about hardware implementations. In both cases, algorithms are implemented. These algorithms implement membrane systems in software and hardware that try to take advantages of massive parallelism. P-systems are parallel and non deterministic systems which simulate membranes behavior when processing information.*

This papers describes the evolution rules application process and it presents software techniques for calculating maximal multisets on every evolutionary step.

These techniques improve the best performance achieved by the p-systems when applying evolution rules.

Algorithms could stop being useful when the number of objects “n” in which they depends on, increases. By using this technique, that specific problem can be overcome. The output can be given under a constant complexity order. The complexity order might be constant under certain conditions, regardless the value “n”. In order to do this, the proper use of memory is essential. This work will provide the details for building a structure. This structure will allow us to improve performance in terms of time. Moreover this structure can be allocated in the random access memory and/or the virtual memory

***Keywords**: P-systems, Parallel systems, Natural Computing, evolution rules application, set of patterns, structure.*

***ACM Classification Keywords**: D.1.m Miscellaneous – Natural Computing*

Introduction

Membrane computing is a parallel and distributed computational model based on the membrane structure of living cells [Păun 2000]. This model has become, during these last years, a powerful framework for developing new ideas in theoretical computation. The main idea was settled in the base of connecting the Biology with Computer Science in order to develop new computational paradigms. P-systems are the structures that reproduce the information processing occurring in living cells. Nowadays, it can be found that several P Systems simulators are much elaborated” [Păun 2005] .At this point, there is a problem with the parallelism synthesized in [Păun 2005] by: “parallelism a dream of computer science, a common sense in biology“. This is the reason why, Păun avoids “to plainly saying that we have ‘implementations of P systems’, because of the inherent non-determinism and the massive parallelism of the basic model, features which cannot be implemented, at least in principle, on the usual electronic computer. […]

There are several uses of the p-systems. P-systems can be used to increase performance when dealing with known problems; for example, the knapsack problem [Pan, Martin 2005].

Also, Membrane computing is currently used to model other parallel and distributed systems as Grid [Qi, Li, Fu, Shi, You 2006].

An overview of membrane computing software can be found in [Ciobanu, Pérez-Jiménez, Ciobanu, Păun 2006], or tentative for hardware implementations [Fernández, Martínez, Arroyo, Mingo 2005] is enough to understand how difficult it is to implement membrane systems on digital devices. This does not mean that simulations of P systems on common computers are not useful; actually, such programs were used in biological applications, and can also have important didactic and research applications". Every algorithm that implements the evolution rules application, for making evolving the systems is sequential. This is due mainly to the fact that technologies are inherently sequential or, when concurrent technology is used, evolution rules are applied to the multiset under mutual exclusion. In this paper, by using a linear structure and allocating it in the physical and/or virtual memory of a given computer, response timing might be reduced considerably. In scenarios where traditional algorithms do not obtain good performance, the technique explained in this paper can guarantee an excellent performance as long as a few requirements are fulfilled. Respecting the inherent non-determinism within the p-systems, the technique respects it as it is a nondeterministic technique itself.

This paper is structured as follows:

- Introduction to membrane computing
  - Definition of concepts as: *P-systems*, *Extinguished multisets*, *multisets of objects, multiplicity of an object and evolution rules.*
  - We focus on the evolution rule application phase. A brief description of this phase is provided
- Creation of a *n-dimensional structure*. That *structure* is created throughout an application that establishes a link between the *initial multisets*, and the number of times that each *evolution rule* should be applied in order to obtain an *extinguished multiset.* The dimension of the structure will be equal to the number of objects we are working with. The number of entries this structure has will be determined by a prefixed value called *"benchmark".*
- Analysis of the *technique* described in this work covering:
- Finally, some conclusions will be established. These conclusions will be the result of doing a study in detail throughout the entire work presented here.

Membrane computing

Transition P Systems

A Transition P System of degree n $n > 1$ is a construct $\Pi = (V, \mu, \omega_1, .., \omega_n, (R_{1,}\rho_1), ..(R_{n,}\rho_n), i_0)$

Where:

V is an alphabet; its elements are called objects;

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*;

ω_i $1 \le i \le n$, are strings from V^* representing multisets over *V* associated with the regions 1,2,..,*n* of μ

R_i $1 \le i \le n$, 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 $1 \le i \le n$, 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'*δ where *v*' is a string over $(V \times \{here, out\}) \cup (V \times \{in_j \; 1 \le j \le n\})$, and δ is a special symbol not in. The length of u is called the radius of the rule $u \to v$

i_o is a number between 1 and *n* which specifies the output membrane of Π

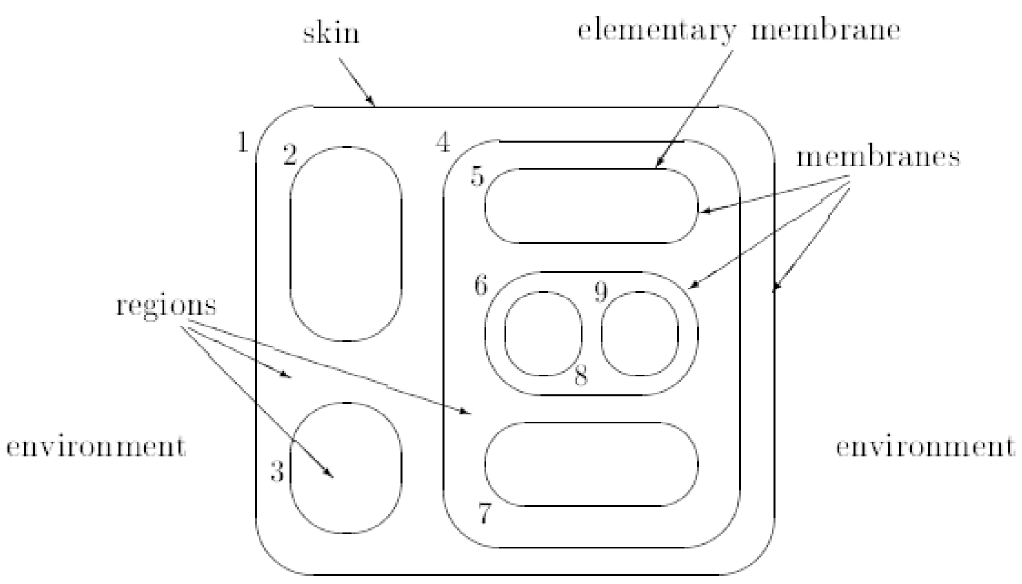

Fig 1 P-system structure

Definition Multiset of objects

Let *U* be a finite and not 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.

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

Note: *Initial Multiset* is the multiset existing within a given region in where no application of evolution rules has occurred yet.

Definition *Evolution rule* with objects in *U* and targets in *T*

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 a s the consequent of the evolution rule *'r'.*

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

Definition 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 over a multiset of objects such as:

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

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

Definition Multiplicity of an object in an evolution rule *r*

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 is defined over an evolution rules such as:

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

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

P-system evolution

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

the representation of the evolution rules are:

$r_1 : a_1^{u11} a_2^{u12} .. a_n^{u1n} \to C_1$

$r_2 : a_1^{u21} a_2^{u22} .. a_n^{u2n} \to C_2$

$\ldots\ldots\ldots\ldots\ldots\ldots \to \ldots\ldots\ldots\ldots\ldots\ldots$

$r_m : a_1^{um1} a_2^{um2} .. a_n^{umn} \to C_m$

P-systems evolve, which makes it change upon time; therefore it is a dynamic system. Every time that there is a change on the p-system we will say that the p-system is in a new transition. The step from one transition to another one will be referred to as an evolutionary step, and the set of all evolutionary steps will be named computation. Processes within the p-system will be acting in a massively parallel and non-deterministic manner. (Similar to the way the living cells process and combine information).

We will say that the computation has been successful if:

- The halt status is reached.
- No more evolution rules can be applied.
- Skin membrane still exists after the computation finishes.

Evolution rule application phase

This phase is the one that has been implemented following different techniques.

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

Rules application to a multiset of object 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 $U = \{a_i \mid 1 \le i \le n\}$ 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*.

The input in the region is the initial multiset *m.*

The output is a maximal multiset *m*'.

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

The transformation only needs rules antecedents as the consequents are part of the communication phase.

Observation

Let $k_i \in N$ be the number of times that the rule 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 will be:

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

Definition Maximal Multisets

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

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

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

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

Observation

In other words, *m* is a maximal or extinguished multiset if and only if it is not possible to apply any more evolution rules over it.

Definition

Given a region *R* and alphabet of objects *U*, and *R (U, T)* set of evolution rules over *U* and targets in *T*.

$r_1 : a_1^{u_{11}} a_2^{u_{12}} . a_n^{u_{1n}} \to C_1$

$r_2 : a_1^{u_{21}} a_2^{u_{22}} . a_n^{u_{2n}} \to C_2$

$\quad \ldots \qquad \to \ldots$

$r_m : a_1^{u_{m1}} a_2^{u_{m2}} . a_n^{u_{mn}} \to C_m$

Maximal multiset is that one that complies with:

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

Within a given membrane, Let $U = \{a_i \mid i = 1, ..n\}$ be a set of objects. Let T be a set of targets. Let $\omega = a_1 a_{2..} a_n$ be a multiset of objects and let u_i be the multiplicity of a_i. Let *R(U,T)* be a multiset of evolution rules with objects in *U* and targets in *T*. Let *m* be the number of evolution rules within *R(U,T)*. Let k_i be the number of times that the rule $r_i \in R(U,T)$ is applied. We define:

$$\begin{aligned} \varphi_1 : N^m \quad &\to \quad N^n \\ (k_1, k_2, .., k_m) \quad &\to \quad (u_1, u_2, .., u_n) \end{aligned}$$

"*m*" is the number of the evolution rules within *R(U,T)*, "*n*" is the number of symbols within the given membrane.

$$\varphi_1(k_1,k_2,..,k_m,) \equiv \begin{pmatrix} u_{11} & u_{21} & ...u_{m1} \\ u_{12} & u_{22} & ...u_{m2} \\ ... & ... & ... \\ u_{1n} & u_{2n} & ...u_{mn} \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \\ ... \\ k_m \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ ... \\ u_n \end{pmatrix}$$

Based on definition we are interested on $\{k=(k_1,k_2,..k_m) \in N^m / \varphi_1(k)=u=(u_1,u_2,..u_n)\}$ and $\bigcap_{l=1}^{m}\left[\bigcup_{i=1}^{n}\left(u_i - \sum_{j=1}^{m}(k_j \cdot u_{ji}) \le u_{li}\right)\right]$

$$\begin{aligned} \Phi_1 : \mathrm{N}^n \quad & \rightarrow \quad \mathrm{P}(\mathrm{N}^m) \\ u=(u_1,u_2,..,u_n) & \rightarrow k=(k_1,k_2,..,k_m) \end{aligned}$$

This function returns a random value $k=(k_1,k_2,..,k_m) \in N^m / \varphi_1(k)=u$. We know that there might be more than one “*k*”. We are just interested on one value ‘*k*’ that complies with $\varphi_1(k)=x$. This is possible as the set $K=\{(k_1,k_2,..,k_m) \in N^m / \varphi_1(k_1,k_2,..,k_m)=x\}$ is always included in a feasible region [1]. This feasible region is an area which contains the set K. There are several methods to select one element of K within a feasible region. These methods are studied on [Arteta, Fernandez, Gil 2008].

Because any of these methods always returns a value, we can state that Φ_1 is a computable function. For example, we can compute this function by implementing the algorithm for Application of Evolution Rules based on linear Diophantine equations. [1]

Patterns function Φ_2

Φ_2 is called pattern function because has as a set of set of patterns as input . This function transforms this set of set of patterns into a set of numbers we call $A=\{(u_1,u_2,..,u_n) \in \mathrm{N}^n\}$. The elements of this resulting set are all the combinations of all the possible values which are included in the Set of set of patterns SP

$$\Phi_2 : \qquad \mathrm{P}(\mathrm{N}^m) \qquad \rightarrow \qquad \mathrm{N}^n$$

$$\begin{pmatrix} [[0,u_{11}],...[0,u_{m1}]],...,[[0,u_{11}],..[0,u_{mn}]], \\ ... \\ [[0,u_{12}],..[0,u_{m1}]],...,[[0,u_{12}],..[0,u_{mn}]], \\ \end{pmatrix} \rightarrow \begin{bmatrix} u_1,u_2,..,u_n \\ u_1',u_2',..,u_n' \\ ... \\ u_1'^p,u_2'^p,...,u_n'^p \end{bmatrix}$$

Bulding the Structure

Creation of the structure L_Φ

Within a region, let $U=\{a_i \mid i=1,..n\}$ be a multiset of objects and let *R (U, T)* be a multiset of evolution rules. Given the set $\{k_i \in N\}$ the number of times that the evolution rule r_i is applied over the initial multiset}, we build the evolution rules function. Φ

Once the function has been constructed, the new structure must be built this way:

- The output of Φ_2 , which is a set of numbers we call *A*= $\{(u_1,u_2,..,u_n) \in \mathrm{N}^n\}$.
- The output of Φ is a unique random value $k=(k_1,k_2,..,k_m) \in N^m$
- For any element, $u=(u_1,u_2,..,u_n) \in A$ we store the value $k=(k_1,k_2,..,k_m) \in N^m \,/\, \varphi_1(k)=u$ .Also we store this value in all the positions that are included in [$(u_1,u_2,..,u_n)$] of the structure. Thus, $L_\Phi\ [(u_1,u_2,..,u_n)] = k_1k_2...k_m$
- If the position $(u_1,u_2,..,u_n)$ of the structure already has information, then we do not store anything.

The expression to build the structure is as follows:

$$L_\Phi\left[\Phi_2\left(SP_{R(U,T)}\right)\right]=\Phi\left(SP_{R(U,T)}\right)$$

Note: $SP_{R(U,T)}$ is the set of set of patterns associated to *our R(U,T)*

Each entry of the structure contains the values: $k_1,...,k_m$. These values will indicate the number of times that an evolution rule should be applied to an initial multiset in order to obtain an extinguished multiset. The number of objects determines the coordinates of a given position in the structure; thus, as the number of objects increases so it does the dimension of the structure; i.e. working with an alphabet *V* of 2 symbols *V*= (a_1,a_2) means that our structure is two-dimensional. The numbers of entries existing in the structure depends on the dimension, (number of symbols of our alphabet) and the benchmark of every object. This benchmark is a value associated to every object of our alphabet.

The benchmark fixes a limit for the multiplicity of the objects existing in our alphabet. In order to make the structure useful, we need to fix reasonably high benchmarks. Our structure does not store information about the values that are higher that the prefixed benchmarks.

Non determinism

Let us *B* be the set $\{k_1,..,k_m \mid$ after applying the evolution rule r_i a number of k_i times to a given multiset $M=\{(a_1,u_1),(a_2,u_2),..,(a_n,u_n)\}$, $\forall\ i \in N\ \ i \leq m$, we obtain an extinguished multiset}

In order to preserve the non-determinism, we must assure that the value returned by our technique is $k \in B$.*k* is chosen randomly between all the elements within *B.*

So far the structure we are building complies with that.

As per [1] we know that there might be more than one element in *B.*

Thus, we must build a dynamic structure which updates itself after the structure returns a solution $(k_1,..,k_m) \in N^m$.

The way to do this is:

For any entry of $L_\Phi\ [u_1,u_2,..,u_n]$:

Let us *B* be the set $\{k_1,..,k_m \mid$ after applying the evolution rule r_i a number of k_i times to a given multiset $M=\{(a_1,u_1),(a_2,u_2),..,(a_n,u_n)\}$, $\forall\ i \in N\ \ i \leq m$, we obtain an extinguished multiset}.

We store in that entry $L_\Phi\ [u_1,u_2,..,u_n]$ an element K= $(k_1,..,k_m) \in B$

- When an initial multiset $M=\{(a_1,u_1),(a_2,u_2),..,(a_n,u_n)\}$ is provided as input, we return K.
- We update the entry $L_\Phi\ [u_1,u_2,..,u_n]$ by allocating a value $k'=(k_1^{'}\ ,..,k_m^{'})\in B$

$(k_1,..,k_m)$ is returning in constant time. The value within the structure is updated once $(k_1,..,k_m)$ is returned. Following this procedure, our technique is ready to return a different output every time we have the same input. The output is chosen in a non deterministic manner.

Technique's Analysis

Building the structures guarantees, that any input value $(u_1,..,u_n)\in N^n$ has a direct relation with $(k_1,..,k_m)\in N^m$. This implies that the structure can be built from a given set of evolution rules. In this way, any algorithm that intends to calculate an extinguished multiset from an initial multiset of objects can generate the structure L_Φ during the compilation time.

During the compiling stage of an algorithm, a new structure is created. When the multiplicities of the initial multisets are provided as input values $u_1,..,u_n$ of a given algorithm, this algorithm will search the entry $[\,u_1,..,\,u_n\,]$ of the structure. Thus, the algorithm returns a combination of values $(k_1,..,k_m)\in N^m$ which are stored in that entry. Any entry or cell stores a set of combinations $(k_1,..,k_m)\in N^m$. Any of these combinations complies with:

$\varphi_1(k_1,k_2,..,k_m)=\{u_1,u_2,..,u_n\}$

One of the combinations stored in a cell is selected randomly. The dynamic way that the structure is created, assures that for each input values, different outputs might be generated. Thus, we are in conditions to state that the use of the structure preserves the non-determinism, which is a characteristic of the Paun's biological model.

In both structures, the number of cells that L_Φ needs, are: $\prod_{i=1}^{n} benchmark\quad(a_i)$.

a_i is an object of the alphabet of our p-system and n is the number of objects of our alphabet. The combination of the number of objects and the selected benchmarks of each object will determine the number of cells that the structure must have. Thus, the combination of these two factors will have a major influence in the amount of memory used by L_Φ .

Within the last years, the algorithms that calculate extinguished multisets have had a complexity Order $\Theta(m)$ where m is the number of evolution rules involved. When the number of evolution rules is high, the execution time can take too long. On the contrary, the best scenario for this technique is when the combination between number of symbols and their benchmarks is low, regardless the number of evolution rules and the maximum applicability benchmark [Fernández, Castellanos, Arroyo, tejedor, Garcia 2006] of any evolution rule.

For instance, we consider a hypothetical scenario where the alphabet of objects $S=\{a_1,a_2\}$, and the number of evolution rules is considerably high. Let us take a number extremely high as 10^{12} .Let us also consider that the maximum value that an evolution rule can be applied is 10^{24} . Under these conditions, an algorithm based on the maximal applicability benchmark [Fernández, Castellanos, Arroyo, tejedor, Garcia 2006] would be inefficient as its complexity is *O(m)*, where *m* is the number of evolution rules. On the contrary, an algorithm that uses this technique can find extinguished multisets with a constant complexity order.

The amount of memory needed to create the structure is explained as follows:

Using this technique we will have to reserve 124 bits for each structure's cell as per: $10^{12} \cdot 10^{24} = 10^{36} \leq 2^{4^{36}} = 2^{124}$. This means 15 bytes per cell .Let us set a benchmark of 30000 for each symbol a_i of the multiset. Considering that there are two objects within our current Multiset: $S = \{a_1, a_2\}$, then we will have $30000 \cdot 30000 = 9 \cdot 10^8$ number of cells of L_Φ . Thus, the amount of memory needed to create L_Φ is $9 \cdot 10^8\ cells \cdot 15 (bytes/cell) = 135 \cdot 10^8\ bytes = 13.5 Gb$

Thus, we can assure that when having 13.5 Gb any algorithm using this technique can find extinguished multisets with complexity order *O(1)* as long as Input values for a_1 *and* a_2 *multiplici ties* are <30000. This will occur regardless the number of evolution rules.

Once this is done, the structure can be built. This structure relates the times that each evolution rule should be applied to the multiplicity input values of the initial multiset. This occurs as long as these values are always either less than or equal to their benchmark.

An algorithm, as the one based on the maximum applicability benchmark, could create the structures during compilation stage. If the number of evolution rules is high, it would be worth considering creating the structure. This will find the corresponding values to the number of times that each rule should be applied in order to obtain an extinguished multiset. This will happen as long as the input values of the object’s multiplicities are lower than the previously fixed benchmarks.

The algorithm

The algorithm we present here is very simple. Once the structure has been built based on the indications given in this paper, it searches on the structure.

The multiplicities of a given initial multiset are provided as input for the algorithm.

The algorithm then searches in the corresponding entry. The only thing it does it is asking for the input numbers which correspond to the initial multiplicity of a given multiset. After that, it just look it up in the entry that has the coordinates equal to the input numbers. It just returns the stored values.

```
(1) $u_1, u_2, .., u_n \leftarrow Multiplici\ ty(m)$
(2) BEGIN
(3) $output(L_\Phi(u_1, u_2, .., u_n))$
(4) END
(5) $REPLACE(L_\Phi(u_1, u_2, .., u_n))$
```

$(u_1, u_{2,} \ldots u_n)$ is the input value corresponding to the multiplicities of the initial multiset m. The algorithm searches in the structure L_Φ the position $(u_1, u_{2,} \ldots u_n)$. After the output is given, another value replaces the entry $(u_1, u_{2,} \ldots u_n)$.

A traditional algorithm can take advantage of the technique as follows:

```
(1) $u_1, u_2, .., u_n \leftarrow Multiplicity(m)$
(2) IF $(u_1, u_2, .., u_n) < BENCHMARK\ (u_1, u_2, .., u_n)$ THEN
    (3) BEGIN
    (4) $output(L_\Phi(u_1, u_2, .., u_n))$
    (5) END
    (6) $REPLACE(L_\Phi(u_1, u_2, .., u_n))$

(7) ELSE
    (8) {TRADITIONA L ALGORITHM }
```

If the multiplicities of the initial multiset are lower than the benchmark then we use the structure; If not, then we use the traditional algorithm.

Conclusions

This work contributes with a new technique in finding extinguished multisets from initial multisets. The technique offers a complement to be used by the traditional algorithms which calculate extinguished multisets. These algorithms find the extinguished multisets after applying a certain number of evolution rules a certain number of times. The technique presents the idea of going "backwards" from the initial multisets to the number of times that each rule should be applied. This process automatically obtains the times that each rule should be applied and no extra calculation is necessary. From there, calculating the extinguished multisets is immediate.

New and old algorithms can take advantage of this technique. Differential factor is that this technique is not affected by the number of evolution rules, which is the major problem traditional algorithms face. Here we return a solution with constant complexity as long as a few requirements are fulfilled.

We've been able to prove that, using this technique, the number of rules has little effect when reserving space for the structures we use. Moreover the use of this technique guarantees that the resulting multisets are obtained in a non deterministic manner.

In order to generalize this technique, a deeper analysis would be necessary to estimate the memory needed when certain rules are provided. This study will focus on four factors when building the structures:

- Number of evolution rules
- Applicability Benchmark of an evolution rule
- Number of symbols
- Benchmark associated to the symbols

This rigorous study will be explained in successive papers.

It is important to stress the point that, when input values are higher than the benchmark previously set, algorithms as *maximum applicability benchmark* [Fernández, Castellanos, Arroyo, tejedor, Garcia 2006] or algorithm based on Diophantine solution [1] should be used. A combination of either one of these algorithms with this technique would be ideal, as the performance will increase depending on the case we are working on.

In a real scenario, the possible amount of memory will determine the optimum number of evolution rules, the applicability benchmark of each rule, the number of required symbols and the benchmarks associated to these symbols.

Within the researching area: “*membranes computing”,* this work proves that not only using parallelism is beneficial in terms of performance but also memory is really helpful under certain conditions. In this case, a proper utilization of memory let us build a structure which will reside in the physical and/or the virtual memory. This structure will have the information that relates multisets to evolution rules.

It will always be a smart option to combine the technique with traditional algorithms.

Bibliography

[Arteta, Fernandez, Gil 2008] A. Arteta, L.Fernandez, J.Gil. Algorithm for Application of Evolution Rules based on linear diophantine equations. Synasc 2008. Timisoara Romania September 2008.

[Ciobanu, Pérez-Jiménez, Ciobanu, Păun 2006] M. Pérez-Jiménez, G. Ciobanu, Gh. Păun. Applications of Membrane Computing, Springer Verlag. Natural Computing Series, Berlin, October, 2006.

[Fernández, Castellanos, Arroyo, tejedor, Garcia 2006] L. Fernández, J.Castellanos, F. Arroyo, J. tejedor, I.Garcia. New algorithm for application of evolution rules. Proceedings of the 2006 International Conferenceon Bioinformatics and Computational Biology, BIOCOMP’06, Las Vegas, Nevada, USA, 2006.

[Fernández, Martínez, Arroyo, Mingo 2005] L. Fernández, V.J. Martínez, F. Arroyo, L.F. Mingo. A Hardware Circuit for Selecting Active Rules in Transition P Systems. Proceedings of International Workshop on Theory and Applications of P Systems.Timisoara (Romania), September, 2005.

[Pan, Martin 2005] L. Pan, C. Martin-Vi de. Solving multidimensional 0-1 knapsack problem by P systems with input and active membranesl. Journal of Parallel and Distributed Computing Volume 65 , Issue 12 (December 2005)

[Păun 2000] Gh. Păun. Computing with Membranes. Journal ofComputer and System Sciences, 61(2000), and Turku

Center of Computer Science-TUCS Report nº 208, 1998.

[Păun 2005] Gh. Păun. Membrane computing. Basic ideas, results, applications. Pre-Proceedings of First International

Workshop on Theory and Application of P Systems,Timisoara (Romania), pp. 1-8, September , 2005.

[Qi, Li, Fu, Shi, You 2006] Zhengwei Qi, Minglu Li, Cheng Fu, Dongyu Shi, Jinyuan You. Membrane calculus: A formal . method for grid transactions. Concurrency and Computation: Practice and Experience Volume18,Issue14 , Pages1799-1809. Copyright © 2006 John Wiley & Sons, Ltd.

Authors' Information

Alberto Arteta *–Associate ProfessorTechnical University of Madrid.aarteta@eui.upm.es*

Angel Castellanos *–Departamento de Ciencias Basicas aplicadas a la Ingeniería Forestal. Escuela de Ingeniería Técnica Forestal. Technical University of Madrid, Avda. de Ramiro de Maeztu s/n 28040 Madrid, Spain. angel.castellanos@upm.es*

Ana MartinezCastellanos *–Departamento de Ciencias Basicas aplicadas a la Ingeniería Forestal. Escuela de Ingeniería Técnica Forestal. Technical University of Madrid, Avda. de Ramiro de Maeztu s/n 28040 Madrid, Spain.* ana.martinez@upm.es