P-SYSTEMS: STUDY OF RANDOMNESS WHEN APPLYING EVOLUTION RULES

Alberto Arteta, Luis Fernández, Fernando Arroyo

Abstract: Membrane computing is a recent area that belongs to natural computing. This field works on computational models based on nature's behavior to process the information. Recently, numerous models have been developed and implemented with this purpose. P-systems are the structures which have been defined, developed and implemented to simulate the behavior and the evolution of membrane systems which we find in nature. What we analyze in this paper is the power of the tools we currently have to simulate the randomness we find in nature. The main problem we face here, is trying to simulate non deterministic events by using deterministic tools. The goal we want to achieve is to propose an optimal method when simulating non deterministic processes. Talking about simulation of non deterministic method makes no sense when using deterministic tools; however we can get closer to the idea of non determinism by using more powerful randomness generators.

Keywords: P-systems, evolution rules application, non-determinism simulation, randomness in p-systems

Introduction

Natural computing is a new field within computer science which develops new computational models. These computational models can be divided into three major areas:

- Neural networks.
- Genetic Algorithms
- Biomolecular computation.

Membrane computing is included in biomolecular computation. Within the field of membrane computing a new logical computational device appears: The P-system. These P-systems are able to simulate the behavior of the membranes on living cells. This behavior refers to the way membranes process information. (Absorbing nutrients, chemical reactions, dissolving, etc)

Membrane computing formally represents, through the use of P-systems, the processes that take place inside of the living cells. In terms of software systems, it is the process within a complex and distributed software. In parallel computational models, p-systems might be as important as the Turing machine is in sequential computational models.[Arroyo, 2001]

In this paper, we study the current methods to implement the idea of randomness. Most of the times the function rnd is used for that purpose. By doing that we state that an important part of inner quality on nature is missed. We will prove that such function has low quality on terms of randomness. When a p-system has a few evolution rules, this will not create any problem. However the entire simulation will degrade when the number of evolution rules increases. By proposing a new way of generating randomness we will get close to the idea of 'pure randomness' we find in nature and also we would be able to show a higher quality simulation.

In order to do this, we will take the following steps:

- Introduction to P-systems theory;
- Analysis of rules application process;
- Analysis of the Random Function
- Study of the current methods to implement non-determinism

- Proposal of a new method.
- Conclusions and further work.

Introduction to P-systems Theory

In this section we will study into detail all of the theories related to the paradigm of the P-systems. A P-system is a computational model inspired by the way the living cells interact with each other through their membranes. The elements of the membranes are called objects. A region within a membrane can contain objects or other membranes. A p-system has an external membrane (also called skin membrane) and it also contains a hierarchical relation defined by the composition of the membranes. A multiset of objects is defined within a region (enclosed by a membrane). These multisets of objects show the number of objects existing within a region. Any object 'x' will be associated to a multiplicity which tells the number of times that 'x' is repeated in a region.

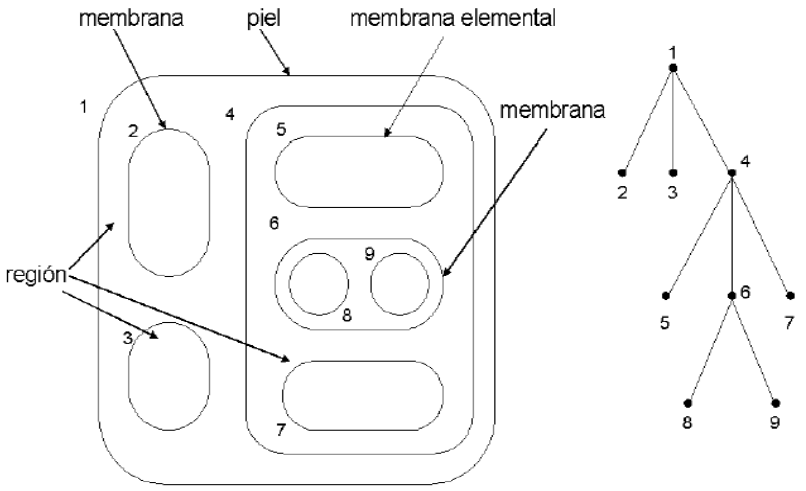

Fig. 1. The membrane's structure (left) represented in tree shape (right)

According to Păun 's definition, a transition P System of degree n, n > 1 is a construct: [Păun 1998]

$$\Pi = \left(V, \mu, \omega_1,.., \omega_n, (R_1, \rho_1),..(R_n, \rho_n), i_0\right)$$

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;
- $\omega_i \; 1 \le i \le n$, 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

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:

$$M : V \to \mathrm{N}$$

$$a_i \to u_1$$

Where a_i is an object and u_i its multiplicity.

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

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

Evolution rule with objects in *U* and targets in *T* is defined by $r = (m, c, \delta)$

where $m \in M(V), c \in M(VxT)\ and\ \delta \in \{to\ dissolve, not\ to\ dissolve\}$

From now on *'c'* will be referred to as the consequent of the evolution rule *'r'*

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

We represent a rule as:

$x \to y \quad or \quad x \to y\delta$ where x is a multiset of objects in M((*V*)xTar) where Tar ={here, in, out} and y is the consequent of the rule. When δ is equal to "dissolve", then the membrane will be dissolved. This means that objects from a region will be placed within the region which contains the dissolved region. Also, the set of evolution rules included on the dissolved region will disappear.

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:

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

Analysis of Rules Application Process

In this paper we focus on the application of evolution rules. Every region of a p_system contains a multiset of symbol-objects, which correspond to the chemicals swimming in a solution in a cell compartment; these chemicals are considered here as unstructured, that is why we describe them by symbols from a given alphabet.

The objects evolve by means of evolution rules, which are also localized, associated with the regions of the membrane structure. There are three main types of rules:[Păun 1998]

1. Multiset rewriting rules (one uses to call them, simply, evolution rules),
2. Communication rules,
3. Rules for handling membranes.

In this section we present the first type of rules. They correspond to the chemical reactions possible in the compartments of a cell, hence they are of the form $u \to v$, where u and v are multisets of objects. However, in order to make the compartments cooperate, we have to move objects across membranes, and to this aim we add

target indications to the objects produced by a rule as above (to the objects from multiset v). These indications are: *"here, in, out"*, with the meaning that an object having associated the indication *here* remains in the same region, one having associated the indication *in* goes immediately into a directly lower membrane, non-deterministically chosen, and *out* indicates that the object has to exit the membrane, thus becoming an element of the region surrounding it. An example of evolution rule is:

$$aab \rightarrow (a, here)(b, out)(c, here)(c, in)$$

(this is the first of the rules considered in Section 4, with target indications associated with the objects produced by rule application). After using this rule in a given region of a membrane structure, two copies of *a* and one *b* are consumed (removed from the multiset of that region), and one copy of a, one of b, and two of care produced; the resulting copy of a remains in the same region, and the same happens with one copy of c (indications here), while the new copy of b exits the membrane, going to the surrounding region (indication out), and one of the new copies of c enters one of the child membranes, non-deterministically chosen. If no such child membrane exists, that is, the membrane with which the rule is associated is elementary, then the indication in cannot be followed, and the rule cannot be applied. In turn, if the rule is applied in the skin region, then b will exit into the environment of the system (and it is "lost" there, as it can never come back). In general, the indication *here* is not specified (an object without an explicit target indication is supposed to remain in the same region where the rule is applied).

A rule as above, with at least two objects in its left hand side, is said to be cooperative; a particular case is that of catalytic rules, of the form $ca \rightarrow cv$, where c is an object (called catalyst) which assists the object a to evolve into the multiset v; rules of the form $a \rightarrow v$, where a is an object, are called non-cooperative.

The rules can also have the form $u \rightarrow v\delta$, where δ denotes the action of membrane dissolving:

if the rule is applied, then the corresponding membrane disappears and its contents , object and membranes alike, are left free in the surrounding membrane; the rules of the dissolved membrane disappear at the same time with the membrane. The skin membrane is never dissolved.

The communication of objects through membranes reminds the fact that the biological membranes contain various (protein) channels through which the molecules can pass (in a passive way, due to concentration difference, or in an active way, with a consumption of energy), in a rather selective manner. However, the fact that the communication of objects from a compartment to a neighboring compartment is controlled by the "reaction rules" is mathematically attractive, but not quite realistic from a biological point of view, that is why there were also considered variants where the two processes are separated: the evolution is controlled by rules as above, without target indications, and the communication is controlled by specific rules (by symport/antiport rules).

We have arrived in this way at the important feature of P systems, concerning the way of using the rules. The key phrase in this respect is: in the maximally parallel manner, non-deterministically choosing the rules and the objects.

More specifically, this means that we assign objects to rules, non-deterministically choosing the objects and the rules, until no further assignment is possible. More mathematically stated, we look to the set of rules, and try to find a multiset of rules, by assigning multiplicities to rules, with two properties: (i) the multiset of rules is applicable to the multiset of objects available in the respective region, that is, there are enough objects in order to apply the rules a number of times as indicated by their multiplicities, and (ii) the multiset is maximal, no further rule can be added to it (because of the lack of available objects).

Thus, an evolution step in a given region consists in finding a maximal applicable multiset of rules, removing from the region all objects specified in the left hand of the chosen rules (with the multiplicities as indicated by the rules and by the number of times each rule is used), producing the objects from the right hand sides of rules, and then

distributing these objects as indicated by the targets associated with them. If at least one of the rules introduces the dissolving action, then the membrane is dissolved, and its contents become part of the immediately upper membrane – provided that this membrane was not dissolved at t he same time, a case where we stop in the first upper membrane which was not dissolved (at least the skin remains intact). [Păun 1998]

Random Function

In common languages as C rand function is defined as a linear congruential generator. A linear congruential generator (LCG) represents one of the oldest and best-known pseudorandom number generator algorithms. The theory behind them is easy to understand, and they are easily implemented and fast.

The generator is defined by the recurrence relation:

$$X_{n+1} = (aX_n + c) \mod m$$

where *Xn* is the sequence of pseudorandom values, and:

$0 < m$ the "modulus"

$0 < a < m$ the "multiplier"

$0 < c < m$ the "increment" (the special case of *c* = 0 corresponds to Park Miller RNG)

$0 < X_0 < m$ the "seed" or "start value"

are integer constants that specify the generator.

While LCGs are capable of producing pseudorandom numbers, this is extremely sensitive to the choice of the coefficients *c*, *m*, and *a*.[Bravo, 2002]

The most efficient LCGs have an *m* equal to a power of 2, most often $m = 2^{32}$ or $m = 2^{64}$, because this allows the modulus operation to be computed by merely truncating all but the rightmost 32 or 64 bits. The following table lists the parameters of LCGs in common use, including built-in *rand()* functions in various compilers.

Source	m	a	c	output bits of seed in *rand()* / *Random(L)*
Numerical Recipes	2^{32}	1664525	1013904223	
Borland C/C++	2^{32}	22695477	1	bits 30..16 in *rand()*, 30..0 in *lrand()*
glibc (used by GCC)	2^{32}	1103515245	12345	bits 30..0
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++	2^{32}	1103515245	12345	bits 30..16
Borland Delphi, Virtual Pascal	2^{32}	134775813	1	bits 63..32 of *(seed * L)*
Microsoft Visual/Quick C/C++	2^{32}	214013	2531011	bits 30..16
Apple CarbonLib	$2^{31} - 1$	16807	0	see Park–Miller RNG

Fig. 2. Relation between random function and LCG parameters for each compiler.

Historically, poor choices had led to ineffective implementations of LCGs. A particularly illustrative example of this is RANDU which was widely used in the early 1970s and resulted in many results that are currently in doubt because of the use of this poor LCG. Moreover, If a linear congruential generator is seeded with a character and then iterated once, the result is a simple classical cipher called an affine cipher; this cipher is easily broken by standard frequency analysis.

When using a LCG, the numbers are distributed into Hyperplanes. If a set of random numbers are part of the same Hyperplane, then the randomness is poor. This is what happens when we use LCGs to generate random numbers. See Fig 3.

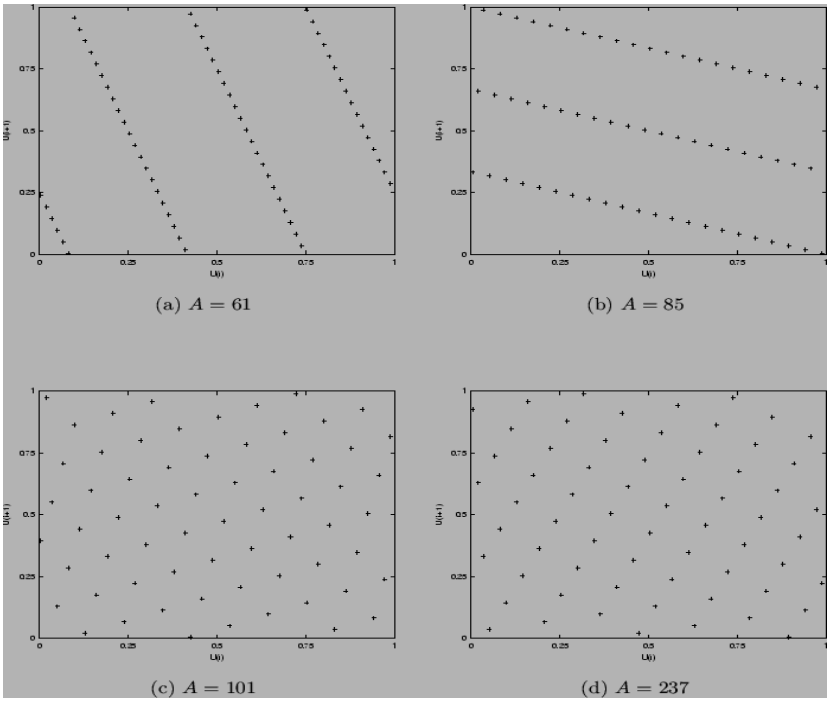

Fig. 3. Random distribution provided by LCG: $X_{n+1} = (aX_n + c) \mod m$

It seems that when using any of the compilers mentioned above, the determinism in our model practically does not exist. Moreover, when we fix a seed, we can totally reproduce the same sequence of numbers over and over again. In other words, the same seed generates the same output every time.

Poor randomness and predictability are signs of a deterministic process. This makes no reliable the idea of generating a non deterministic process as it is the one occurring within the living cells.

Rules Applicability: Implementation of Non Determinism

Applying evolution rules in a p_system is meant to be purely random. The way that reactions occur within the living cells is non deterministic. A common method to implement this behavior is to use the RND function. Nowadays, there are several methods of application of evolution rules which have been implemented. Algorithms as *Step by step Max applicability benchmark*, *Minimal applicability benchmark [Fernandez,2006]* . All of them study this point and try to improve the performance when applying rules. Here is an example of algorithm that applies the rules based on an applicability benchmark.

```
(1)  ω_R(U) <- Ø_M(R(U))
(2)  REPEAT
(3)   r_i <- random(A)
(4)   max <- maximalApplicable(r_i, ω)
(5)   IF max = 0 THEN
(6)     A <- A − {r_i}
(7)   ELSE BEGIN
(8)     k <- random(1,max)
(9)     ω_R(U) <- ω_R(U) + {(r_i,k)}
(10)    ω <- ω − k·input(r_i)
(11)  END
(12) UNTIL |A|=0
```

Fig 4. Maximal applicability benchmark algorithm.

As shown, there are two calls to the random function. The current implementation of the random function makes the entire algorithm not very accurate on simulating the inherent non determinism within the living cells. The main reason is because the use of LCG which produces creates poor randomness and generates predictable output streams, while in a real scenario this should not occur.

Rules Applicability: ICG, New Implementation Proposal

In order to simulate randomness better, we must use more accurate random number generators.

The random generator we propose is able to simulate randomness in a better way. As the *LCGs* are proved not to be good for this simulation, We focus on the non linear ones.

The non linear congruential generator we propose here, is an Inversive congruential generators (*ICGs*).

Inversive congruential generators are a type of nonlinear congruential pseudorandom number generator, which use the modular multiplicative inverse [2] (if it exists) to generate the next number in a sequence. The standard formula for an inversive congruential generator is

$$X_{n+1} = \overline{(aX_n + c)} \mod m$$

Sometimes the Parallel Hyperplanes phenomenon inherent in LCGs may cause adverse effects to certain simulation applications because the space between the hyperplanes will never be hit by any point of the generator, and the simulation result may be very sensitive to this kind of regularities. Inversive Congruential Generators (ICG) are designed to overcome this difficulty. It is a variant of LCG:

where $\bar{c} = 0$ if $c = 0$ and $\bar{c} = c^{-1} \bmod M$. To calculate $\bar{c}$, one can apply the reverse of Euclid's algorithm to find integer solutions for $\bar{c}c + KM = 1$.

Although the extra inversion step eliminates Parallel Hyperplanes (see Fig. 5), it also changes the intrinsic structures and correlation behaviors of LCGs. ICGs are promising candidates for parallelization, because unlike LCGs, ICGs do not have long-range autocorrelations problems.

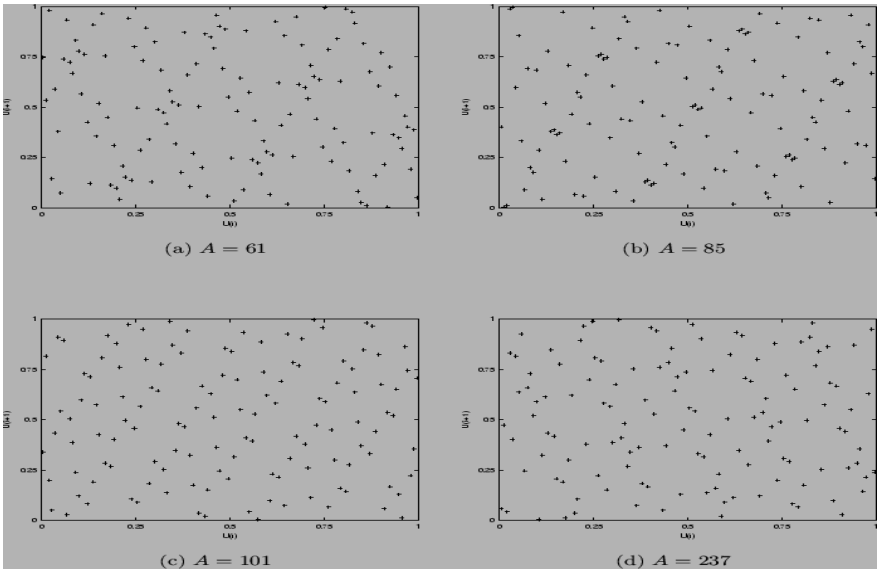

Fig. 5. Random distribution produced by ICG $X_{n+1} = \overline{(aX_n + c)} \mod m$

As shown the numbers are not distributed into Hyperplanes. This improves the simulation in terms of randomness which get us much closer to the idea of non determinism we are looking for.

Thus, the implementation of our random function to be used by our p-systems is:

Function $RANDOM\_ICG(n) = (A * RANDOM\_ICG(n-1) + C) \mod M$ where $0 \le n \le M$ and M the number of evolution rules. The parameters we propose are:

A=237

M= 2^{32}

C=1265

RANDOM(0) is the seed of the ICG and it can be set to any arbitrary number.

As shown in Figure 2 it is proved that this ICG does not generate parallel hyperplanes which get us closer to the idea of pure randomness in our model.

Conclusion and Further Work

In this paper, we have studied some topics of membrane computing. As a part of this study, we have explained some concepts of the p-systems. Concepts such as:

1. Components
2. Interactions between the components.
3. The evolution of a p-system.

Moreover, we have focused our work on a specific part of the p-systems: Evolution rules application. The way that rules are applied in a region must be purely random. In order simulate this behavior we see that random function has been used by most developers. Most Compilers have implemented the random function by using LCG. After analyzing LCG we have concluded that it is a poor tool in terms of randomness and non determinism. As stated, is practically impossible to simulate a non deterministic process through a deterministic machine. However we can get closer to the idea of non determinism by increasing the quality of the random number generators.

By implementing and using a new random function we have been able to provide a better simulation in terms of randomness. This function uses the ISG we proposed in the above section. The random numbers generated by ICG are not placed in Parallel hyperplanes which improves simulation in terms of randomness.

Although it is practically impossible to simulate a non deterministic process by using deterministic tools as computers, we can improve the quality of simulation by using new random generators. This can be noticeable when the number of evolution rules increases within a given region. Although we approached the idea of randomness in the evolution rules application process, we still need to work on avoiding predictability as we could guess a given random number by knowing the initial value or seed of the ICG. [Blackburn, 2004] In the future, we will try to improve even more the simulation explained on this paper in terms of randomness and non determinism.

Bibliography

[Păun 1998] "Computing with Membranes", Journal of Computer and System Sciences, 61(2000), and Turku Center of Computer Science-TUCS Report nº 208, 1998.

[Blackburn, 2004] "Predicting nonlinear pseudorandom number generators" Journal Mathematics of Computation. 74 (2005), 1471-1494.

[Arroyo, 2001] "Structures and Bio-language to Simulate Transition P Systems on Digital Computers," Multiset Processing International Workshop Membrane Computing, Curtea de Arges (Romania), August 2002, Springer-Verlag, Vol 2597, pp. 19-32, Berlin, 2003

[Bravo, 2002] " Una funcion random poco aleatoria". Spanish journal of physics , ISSN 0213-862X Vol.16 pp 60-62

[Fernandez,2006] "New Algorithms for Application of Evolution Rules based on Applicability Benchmarks". BIOCOMP06: International Conference on Bioinformatics and Computational Biology, Las Vegas, (June, 2006)

Authors' Information

Alberto Arteta Albert – Associate professor U.P.M Crtra Valencia km 7, Madrid-28031, Spain;
e-mail: aarteta@eui.upm.es

Research: Membrane computing, Education on Applied Mathematics and Informatics

Luis Fernández Muñoz – Associate professor U.P.M Crtra Valencia km 7, Madrid-28031, Spain;
e-mail:setillo@eui.upm.es

Fernando Arroyo Montoro– Associate professor U.P.M Crtra Valencia km 7, Madrid-28031, Spain;
e-mail: farroyo@eui.upm.es