DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution

M.A. Martínez-del-Amor[1], I. Pérez-Hurtado[1], M. García-Quismondo[1], L.F. Macías-Ramos[1], L. Valencia-Cabrera[1], A. Romero-Jiménez[1], C. Graciani-Díaz[1], A. Riscos-Núñez[1], M.A. Colomer[2], M.J. Pérez-Jiménez[1]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mdelamor@us.es, perezh@us.es, mgarciaquismondo@us.es, lfmaciasr@us.es, lvalencia@us.es, romero.alvaro@us.es, cgdiaz@us.es, ariscosn@us.es, marper@us.es

[2] Department of Mathematics
University of Lleida
Avda. Alcalde Rovira Roure, 191, 25198 Lleida, Spain
E-mail: colomer@matematica.udl.es

Summary. Population Dynamics P systems refer to a formal framework for ecological modelling. The semantics of the model associates probabilities to rules, but at the same time, the model is based on P systems, so the rules are applied in a maximally parallel way. Since the success of the first model using this framework [5], initially called multienvironment probabilistic P systems, several simulation algorithms have been defined in order to better reproduce the behaviour of the ecosystems with the models.

BBB and DNDP are previous attempts, which define blocks of rules having the same left-hand side, but do not define a deterministic behaviour when different rules are competing for the same resources. That is, different blocks of rules present in their left-hand side common objects, being applicable at the same time. In this paper, we introduce a new simulation algorithm, called DCBA, which performs a proportional distribution of resources.

Keywords: Membrane Computing, Population Dynamics, Simulation Algorithm, Probabilistic P systems, DCBA, P-Lingua, pLinguaCore

1 Introduction

Membrane Computing has a far–reaching background on the modelling of biochemical phenomena, within the framework of Computational Systems Biology

[2, 7, 17, 19], being complementary and an alternative to more classical approaches (i.e. ODEs, Petri Nets, etc). However, in 2011 a Membrane Computing modelling framework for ecosystem dynamics was introduced [3]. Based on this framework, several ecosystem models have already been presented. Some examples are the population dynamics of *Gypaetus barbatus* [4] and *Rupicapra p. pyrenaica* [8] in the Catalan Pyrenees, as well as the population density of *Dreissena polymorpha* in Ribarroja reservoir [3]. Some of the assets of this framework are the ability to analyse the simultaneous evolution of a high number of species, as well as the management of a large number of auxiliary objects. These objects could represent, for instance, grass, biomass or animal bones.

The results obtained from the application of the framework on different ecosystems prove its versatility and adaptability. Thus, a straightforward interpretation of the results of the simulations of its models can be easily obtained by checking the states and multisets associated to each one of the membranes.

Although this framework allows a direct interpretation of the simulations of its models, the simulation itself is a complicated problem to solve from a practical point of view. Therefore, algorithms capable of capturing the semantics described by the framework are necessary. These algorithms should be able to select rules in the models according to their associated probabilities while keeping the maximal semantics of P systems. In this scenario, the concept of *rule block* takes form. A rule block is a set of rules whose left hand side (that is, the necessary and sufficient condition for them to be applied) is exactly the same. That is, given a P system configuration, either all or none of the rules in the block can be applied. According to the semantics associated to the modelling framework, one or more blocks are selected on each step of computation. The probability for a block to be selected is calculated out of the probabilities of its rules. Once a rule block is selected, its rules are applied a number of times in a probabilistic manner according to their associated probabilities, also known as *local probabilities*. Henceforth, the condition of the sum of the probabilities associated to all rules in each block being equal to 1 is imposed.

The way in which the blocks and rules in the model are selected depends on the specific simulation algorithm employed. These algorithms should be able to deal with issues such as the possible overlapping of left hand sides from different blocks, which might result on the competition of blocks and rules for objects. So far, several algorithms have been developed in order to capture the semantics defined by the modelling framework. Some of these algorithms are the Binomial Block Based algorithm (*BBB*) and the Direct Non Deterministic algorithm with Probabilities (*DNDP*). A comparison on the performance of these algorithms can be found on [9].

The algorithms mentioned above share a common drawback. This drawback involves the distortion of the way in which blocks and rules are selected. That is, instead of blocks and rules being selected according to its probabilities in a uniform manner, this selection process is biased towards those with the highest probabilities. This paper introduces a new algorithm, known as Direct distribution based on Consistent Blocks Algorithm (*DCBA*). This algorithm is introduced to solve the aforementioned distortion, thus not biasing the selection process towards the most likely blocks and rules.

The rest of the paper is structured as follows: Section 2 introduces preliminary concepts, such as the formal modelling framework of PDP systems and the DNDP algorithm. Section 3 describes the DCBA algorithm, together with a test example to show the differences with DNDP, and some details on the implementation in the PlinguaCore software framework. Section 4 shows the behaviour of DCBA when simulating a real ecosystem model. The simulated model has been adapted and improved from the original version. The paper ends with some conclusions and ideas for future work in Section 5.

2 Preliminaries

2.1 The P system based framework

Definition 1. *A Population Dynamics P system of degree* (q, m) *with* $q \geq 1$, $m \geq 1$, *taking* T *time units,* $T \geq 1$, *is a tuple*

$(G, \Gamma, \Sigma, T, R_E, \mu, R, \{f_{r,j} : \ r \in R, 1 \leq j \leq m\}, \{\mathcal{M}_{ij} : \ 0 \leq i \leq q-1, \ 1 \leq j \leq m\})$

where:

- $G = (V, S)$ *is a directed graph. Let* $V = \{e_1, \ldots, e_m\}$ *whose elements are called environments;*
- Γ *is the working alphabet and* $\Sigma \subsetneqq \Gamma$ *is an alphabet representing the objects that can be present in the environments;*
- T *is a natural number that represents the simulation time of the system;*
- R_E *is a finite set of communication rules between environments of the form*

$$(x)_{e_j} \xrightarrow{p_{(x,j,j_1,\ldots,j_h)}} (y_1)_{e_{j_1}} \ldots (y_h)_{e_{j_h}}$$

  where $x, y_1, \ldots, y_h \in \Sigma$, $(e_j, e_{j_l}) \in S$ $(l = 1, \ldots, h)$ *and* $p_{(x,j,j_1,\ldots,j_h)}(t) \in [0, 1]$, *for each* $t = 1, \ldots, T$. *If* $p_{(x,j,j_1,\ldots,j_h)}(t) = 1$, *for each* t, *then we omit the probabilistic function. These rules verify the following:*

  - ⋆ *For each environment* e_j *and for each object* x, *the sum of functions associated with the rules from* R_E *whose left-hand side is* $(x)_{e_j}$ *coincides with the constant function equal to 1.*

- μ *is a membrane structure consisting of* q *membranes, with the membranes injectively labeled by* $0, \ldots, q-1$. *The skin membrane is labeled by 0. We also associate electrical charges from the set* $\{0, +, -\}$ *with membranes.*

- *R is a finite set of evolution rules of the form $r : u[\,v\,]_i^\alpha \to u'[\,v'\,]_i^{\alpha'}$ where u, v, u', v' are multisets over Γ, $i \in \{0, 1, \ldots, q-1\}$, and $\alpha, \alpha' \in \{0, +, -\}$.*
- *For each $r \in R$ and for each j, $1 \leq j \leq m$, $f_{r,j}$ is a computable function whose domain is $\{1, \ldots, T\}$ and its range is $[0, 1]$, verifying the following:*
  - ⋆ *For each $u, v \in \Gamma^*$, $i \in \{0, \ldots, q-1\}$ and $\alpha, \alpha' \in \{0, +, -\}$, if $r_1, \ldots, r_z$ are the rules from R whose left-hand side is $u[v]_i^\alpha$ and the right-hand side have polarization α', then $\sum_{j=1}^{z} f_{r_j}(t) = 1$, for each t, $1 \leq t \leq T$.*
  - ⋆ *If $(x)_{e_j}$ is the left-hand side of a rule $r \in R_E$, then none of the rules of R has a left-hand side of the form $u[v]_0^\alpha$, for any $u, v \in \Gamma^*$ and $\alpha \in \{0, +, -\}$, having $x \in u$.*
- *For each j $(1 \leq j \leq m)$, $\mathcal{M}_{0j}, \ldots, \mathcal{M}_{q-1,j}$ are strings over Γ, describing the multisets of objects initially placed in the q regions of μ, within the environment e_j.*

In other words, a system as described in the previous definition can be viewed as a set of m environments $e_1, \ldots, e_m$ linked between them by the arcs from the directed graph G. Each environment e_j contains a P system, $\Pi_j = (\Gamma, \mu, R, \mathcal{M}_{0j}, \ldots \mathcal{M}_{q-1,j})$, of degree q, such that $\mathcal{M}_{0j}, \ldots, \mathcal{M}_{q-1,j}$ describe the initial multisets for this environment, and every rule $r \in R$ has a computable function $f_{r,j}$ (specific for environment j) associated with it.

The tuple of multisets of objects present at any moment in the m environments and at each of the regions of each Π_j, together with the polarizations of the membranes in each P system, constitutes a *configuration* of the system at that moment. At the initial configuration of the system we assume that all environments are empty and all membranes have a neutral polarization.

We assume that a global clock exists, marking the time for the whole system, that is, all membranes and the application of all rules (both from R_E and R) are synchronized in all environments.

The P system can pass from one configuration to another by using the rules from $R = R_E \cup \bigcup_{j=1}^{m} R_{\Pi_j}$ as follows: at each transition step, the rules to be applied are selected according to the probabilities assigned to them, and all applicable rules are simultaneously applied in a maximal way.

When a communication rule between environments

$$(x)_{e_j} \xrightarrow{p_{(x,j,j_1,\ldots,j_h)}} (y_1)_{e_{j_1}} \ldots (y_h)_{e_{j_h}}$$

is applied, object x passes from e_j to $e_{j_1}, \ldots, e_{j_h}$ possibly modified into objects $y_1, \ldots, y_h$, respectively. At any moment t, $1 \leq t \leq T$, for each object x in environment e_j, if there exist communication rules whose left-hand side is $(x)_{e_j}$, then one of these rules will be applied. If more than one communication rule can be applied to an object, the system selects one randomly, according to their probability which is given by $p_{(x,j,j_1,\ldots,j_h)}(t)$.

For each j $(1 \leq j \leq m)$ there is just one further restriction, concerning the consistency of charges: in order to apply several rules of R_{Π_j} simultaneously to the same membrane, all the rules must have the same electrical charge on their right-hand side.

2.2 DNDP simulation algorithm

In this section, the Direct Non-deterministic Distribution with probabilities algorithm (DNDP) [14, 13] is briefly described (algorithm 1). The aim of this algorithm is to perform a non-deterministic object distribution, so rules having common objects in their left-hand sides (object competition) will have the same opportunities to consume objects.

The input consists on a PDP system of degree (q, m), and a number T of time units. The algorithm simulates T transition steps of the PDP system. Therefore, it only simulates one computation of the PDP system, by selecting and executing rules in a non-deterministic maximal consistent parallel way.

Algorithm 1 DNDP MAIN PROCEDURE

Require: A PDP system of degree (q, m) with $q \geq 1$, $m \geq 1$, taking T time units, $T \geq 1$.

1: $C_0 \leftarrow$ initial configuration of the system
2: **for** $t = 0$ to $T - 1$ **do**
3: $C'_t \leftarrow C_t$
4: *Initialization*
5: *First selection phase (consistency).*
6: *Second selection phase (maximality).*
7: *Execution of selected rules.*
8: $C_{t+1} \leftarrow C'_t$
9: **end for**

Similarly to the previous algorithms [14], the transitions of the P system are simulated in two phases, selection and execution, to synchronize the consumption and production of objects. However, selection is divided in two micro-phases: the first one calculates a multiset of mutually consistent applicable rules, and the second assure maximal application by eventually increasing the multiplicity of some rules in the previous multiset, obtaining a multiset of maximal mutually consistent applicable rules. The algorithm is described below, but for more details refer to [13].

First of all, in order to simplify the selection and execution phases, the *initialization* process constructs two ordered set of rules, A_j and B_j, gathering only rules from R_E and R_Π applicable in environment e_j, in the sense of having the same charge in the left-hand side than the membranes in the configuration.

In the *first selection* phase, a multiset of consistent applicable rules, denoted by R_j^1 for each environment e_j, is calculated. Moreover, a multiset of possible applicable rules, denoted by R_j^0, is also created. We will say that two rules are consistent if they are associated to the same membrane, and they update it to the same charge. It is used in order to store rules having 0 as the number of applications when using the random number generator function. Hence, this multiset allows to have elements with multiplicity 0.

First, a random order is applied to $A_j \cup B_j$, and stored in an ordered set D_j. Moreover, a copy of the configuration C_t, called C'_t, is created and it is updated each time that a rule is selected (removing the left-hand side). Then, a rule r is applicable if the following holds: it is consistent with the previously (according to the order in D_j) selected rules in R_j^1, and the number of possible applications M in C'_t is greater than 0. If a rule r is applicable, a binomial distributed random number of applications n is calculated according to the probability.

On the one hand, since C'_t has been updated by the previously selected rules, the number n cannot exceed M to guarantee a correct object distribution. On the other hand, if the generated number n is 0, the corresponding rule is added to the multiset R_j^0, giving another chance to be selected in the next phase (maximality). Note that only rules from R_j^1 are considered for the consistency condition, since rules from R_j^0 are not applied in the first selection phase.

In the *second selection* phase, the consistent applicable rules are checked again in order to achieve maximality. Only consistent rules are considered, and they are taken from $R_j = R_j^0 \cup R_j^1$. If one rule $r \in R_j$ has a number of applications M greater than 0 in C'_t, then M will be added to the multiplicity of the rule. In order to fairly distribute the objects among the rules, they are iterated in order with respect to the probabilities. Moreover, one rule from the multiset R_j^0 can be checked, so it is possible that another rule from R_j^1, inconsistent to this one, have been previously selected. In this case, the consistent condition has to be tested again.

An example of several executions of the DNDP algorithm is showed in section 3.3, together with a comparison with the new algorithm introduced in this paper.

3 Direct distribution based on Consistent Blocks Algorithm (DCBA)

3.1 Definitions for blocks and consistency

The selection mechanism starts from the assumption that rules in R can be classified into blocks of rules having the same left-hand side, following the definitions 2, 3 and 4 below.

Definition 2. *The left and right-hand sides of the rules are defined as follows:*

(a) Given a rule $r \in R_\Pi$ of the form $r : u[v]_h^\alpha \to u'[v']_h^{\alpha'}$:

- *The left-hand side of r is defined as $LHS(r) = (h, \alpha, u, v)$, where $h \in L$, $\alpha \in \{0, +, -\}$ and $u', v' \in \Gamma^*$. This corresponds to multiset u in the parent membrane of h, multiset v in membrane h, and membrane h with charge α.*
- *The right-hand side of r is defined as $RHS(r) = (h, \alpha', u', v')$, where $h \in L$, $\alpha' \in \{0, +, -\}$ and $u', v' \in \Gamma^*$. This corresponds to multiset u' in the parent membrane of h, multiset v' in membrane h, and membrane h with charge α'.*

(b) Given a rule $r \in R_E$ of the form $r : (x)_{e_j} \rightarrow (y_1)_{e_{j_1}} \dots (y_k)_{e_{j_k}}$:

- *The left-hand side of r is defined as $LHS(r) = (e_j, x)$, corresponding to the multiset with only one occurrence of object x in environment e_j.*
- *The right-hand side of r is defined as $RHS(r) = (e_{j_1}, y_1) \dots (e_{j_k}, y_k)$, corresponding to the k multisets with single objects $y_1 \dots y_k$, for each environment $e_{j_1} \dots e_{j_k}$ respectively.*

Definition 3. *Rules from R_Π can be classified in blocks associated to (h, α, u, v) as follows: $B_{h,\alpha,u,v} = \{r \in R_\Pi : LHS(r) = (h, \alpha, u, v)\}$.*

Definition 4. *Rules from R_E can be classified in blocks associated to (e_j, a) as follows: $B_{e_j,a} = \{r \in R_E : \exists a \in \Sigma, LHS(r) \equiv (a)_{e_j}\}$.*

Recall that, according to the semantics of the model, the sum of probabilities of all the rules belonging to the same block is always equal to 1 – in particular, rules with probability equal to 1 form individual blocks. Note that rules with overlapping (but different) left-hand sides are classified into different blocks.

Definition 5. *A block $B_{h,\alpha,u,v}$ is consistent if and only if $\exists \alpha', \forall r \in B_{h,\alpha,u,v}$, $charge(RHS(r)) = \alpha'$.*

Definition 6. *A consistent block $B_{h,\alpha,\alpha',u,v}$, with $h \in H$, $\alpha, \alpha' \in \{0, +, -\}$, $u, v \in \Gamma^*$, is of the form $B_{h,\alpha,\alpha',u,v} = \{r \in R : \exists u', v' \in \Gamma^* : r \equiv u[v]_h^\alpha \rightarrow u'[v']_h^{\alpha'}\}$.*

Remark 1. Note that **all** the rules $r \in B_{h,\alpha,\alpha',u,v}$ are consistent, in the sense that each membrane h with charge α goes to the **same** charge α' when any rule of $B_{h,\alpha,\alpha',u,v}$ is applied.

Definition 7. *Two blocks $B_{h_1,\alpha_1,\beta_1,u_1,v_1}$ and $B_{h_2,\alpha_2,\beta_2,u_2,v_2}$ are mutually consistent with themselves, if and only if $(h_1 = h_2 \wedge \alpha_1 = \alpha_2) \Rightarrow (\beta_1 = \beta_2)$.*

Definition 8. *A set of blocks $\mathcal{B} = \{B^1, B^2, \dots, B^s\}$ is self consistent (or mutually consistent) if and only if $\forall i, j (i \neq j \Rightarrow B^i$ and B^j are mutually consistent).*

Remark 2. In such a context, a set of blocks has an associated set of tuples (h, α, α'), that is, a relationship of $H \times C$ in C. Then, a set of blocks is mutually consistent if and only if the associated relationship $H \times C$ in C is functional.

3.2 DCBA pseudocode

This new simulation algorithm for PDP systems has the same general scheme than its predecessor, DNDP (algorithm 1). The main loop (algorithm 2) is divided into two stages: selection and execution of rules, similarly to the DNDP algorithm.

Algorithm 2 DCBA MAIN PROCEDURE

Require: A Population Dynamics P system of degree (q, m), $T \geq 1$ (time units), and $A \geq 1$ (*Accuracy*). The initial configuration is then called C_0.

1: *INITIALIZATION* ▷ (Algorithm 4).
2: **for** $t \leftarrow 0$ to $T-1$ **do**
3: $C'_t \leftarrow C_t$
4: Calculate probability functions $f_{r,j}(t)$ associated to the rules.
5: *SELECTION* of rules. ▷ (Algorithm 3)
6: *EXECUTION* of rules. ▷ (Algorithm 8)
7: $C_{t+1} \leftarrow C'_t$
8: **end for**

Note that the algorithm selects and executes rules, but not blocks of rules. Blocks are used by DCBA in order to select rules, and this is made in three micro-stages as seen in algorithm 3. Phase 1 calculates a proportional object *distribution* to the blocks. Phase 2 assures the *maximality* by checking the maximal applications of each block. And finally, phase 3 passes from block applications to rule applications by calculating random numbers following the multinomial distribution with the corresponding *probabilities*.

Algorithm 3 SELECTION

1: Selection *PHASE 1*: distribution ▷ (Algorithm 5)
2: Selection *PHASE 2*: maximality ▷ (Algorithm 6)
3: Selection *PHASE 3*: probabilities ▷ (Algorithm 7)

Before starting to select and execute rules in the system, some data initialization is required (see algorithm 4). For instance, the selection stage uses a table in order to distribute the objects among the blocks. This table T, also called *static table*, is used in each time step, so it is initialized only once, at the beginning of the algorithm. The *static table* has one column per each consistent block of rules, and one row per each pair of object and compartment (i.e. each membrane and the environment in the skeleton). An expanded static table T_j is also constructed for each environment, to consider also blocks from environment communication rules. Finally, two multisets, B_j and R^j_{sel}, are initialized for selected blocks and rules, respectively.

Remark 3. The columns of the static table contain the information of their left-hand side of the blocks. The rows of the static table contain the information of the competitions for objects: each block competing for a given object will have a value different to $-$ in the corresponding row.

Algorithm 4 INITIALIZATION

1: Construction of the static distribution table T:
- Columns: consistent blocks of rules from R_Π: $B_{h,\alpha,\alpha',u,v}$
- Rows: pairs $(obj, membr)$ and (obj', e), for all object $obj \in \Gamma$, $obj' \in \Sigma$ and membrane $membr \in \mu$, being e a way to generically identify the environment of the skeleton of the P systems in the multienvironment system.
- Values: place $1/k$ in the element (x, y) of the table T, if the corresponding object to the row x is in their left-hand side of the block given by column y, with multiplicity k. Otherwise, keep unmarked with $-$.

2: **for** $j = 1$ **to** m **do** ▷ (Construct the expanded table T_j)
3: $T_j \leftarrow T$. ▷ (Initialize the table with the original T)
4: Add to table T_j a column for each communication rule block from R_E associated to the environment e_j, and place the value 1 in the corresponding row for (obj', e), being obj' the object appearing in the left-hand side.
5: **end for**
6: Initialize the multisets $B_j \leftarrow \emptyset$ and $R^j_{sel} \leftarrow \emptyset$

The distribution of objects among the blocks with overlapping left-hand sides is performed in selection phase 1 (algorithm 5). The expanded static table T_j is used for this purpose in each environment. Three filters are defined in order to adapt the T_j to the current state of the system. That is, to select which rule blocks are going to receive objects. The first filter will delete columns of the table corresponding to non applicable rule blocks due to the charges in the left-hand side. The second filter will delete the columns of the rule blocks with no applications in a configuration, because of the objects in the left-hand side. The goal of the third filter is to save space in the table, deleting rows with no correspondence with the non-filtered columns. These three filters are applied at the beginning of phase 1, and the result is a *dynamic table* T^t_j (for the environment j and time step t).

Filter functions for selection Phase 1

function FILTER 1(table T, configuration C) ▷ (By columns and charges)
Delete columns from table T, according to the charge of the membrane in the left-hand side of the corresponding block and in the configuration C.
return T
end function

function FILTER 2(table T, configuration C) ▷ (By columns and multiplicity)
Delete columns from table T, such that for any row $(obj, membr)$ or (obj', e), the multiplicity of that object in C multiplied by $1/k$ (value in the table), returns a number $\kappa, 0 \leq \kappa < 1$. If all the values for that column are $-$, it is also filtered.
return T
end function

function FILTER 3(table T, configuration C) ▷ (By rows and multiplicity)
Delete rows from T of pairs $(obj, membr), (obj', e)$ according to the multisets of C, those having multiplicity 0.
return T
end function

Remark that the *static table* T contains all consistent blocks in the columns. The set of all consistent blocks is unlikely to be mutually consistent. However, two non-mutually consistent blocks, $B_{h,\alpha,\alpha'_1,u,v}$ and $B_{h,\alpha,\alpha'_2,w,y}$ (assigning a different charge to the same membrane), can be filtered as follows:

- If $u \neq w$ or $v \neq y$, and if one of these multisets is not present in C_t, then one of the blocks is not applicable, and therefore will be filtered by filter 2. This situation is commonly handled by the model designers, in order to take control of the model evolution.

It is very important to have a set of mutually consistent blocks before distributing objects to the blocks. For this reason, there are two complementary methods to detect it. First, and after applying filters 1 and 2, a loop to check the mutually consistency is performed. If this method ends with an error, meaning that an inconsistency was encountered, the simulation process can finish, warning the designer with the reason. Nevertheless, it can be interesting to find a way to continue the execution by non-deterministically constructing a subset of mutually consistent blocks. Since this method can be exponentially expensive in time, it is optional for the user to whether activate it or not.

Once the columns of the *dynamic table* represent a set of mutually consistent blocks, the distribution process starts. This is carried out by browsing the rows of the table, in such a way that the values of the rows, different to $-$, will be the multiplication of:

- The normalized value respecting the row, that is, the value divided by the total sum of the row. This calculates a way to *proportionally* distribute the corresponding object along the blocks. Since it depends on the value k (multiplicity in the left-hand side), the distribution actually penalize the blocks requiring more copies of the same object, what is inspired in the amount of energy required to join individuals from the same species.
- The value in the original dynamic table (i.e. $\frac{1}{k}$). This indicates the number of possible applications of the block with the corresponding object.
- The corresponding multiplicity of the object in the current configuration C'_t. This performs the distribution of the number of copies of the object along the blocks.

After the object distribution process, the number of applications for each block is calculated by selecting the minimum value in each column. This number is then used for consuming the left-hand side from the configuration. However, this application could be not maximal. The distribution process can eventually deliver objects to blocks that are restricted by other objects. In view of that this situation may occur frequently, the distribution and the configuration update process can be A times, being A an input parameter referring to *accuracy*. The more the process is repeated, the more is the distribution accurate, but the performance of the simulation can be lower. We have experimentally see that $A = 2$ gives the best accuracy/performance ratio.

Algorithm 5 SELECTION PHASE 1: DISTRIBUTION

1: **for** $j = 1$ **to** m **do** ▷ (For each environment e_j)
2: Apply filters to table T_j using C_t, obtaining T_j^t. The filters are applied as follows:
a. $T_j^t \leftarrow T_j$
b. $T_j^t \leftarrow$ *Filter 1* (T_j^t, C_t).
c. $T_j^t \leftarrow$ *Filter 2* (T_j^t, C_t).
d. Check *mutual consistency* for the blocks remaining in T_j^t:
- Create a vector MC_j^t, of order q (number of membranes in Π), with $MC_j^t(i) = -1, 1 \leq i \leq q$.
- **for each** column $B_{h,\alpha,\alpha',u,v}$ in T_j^t, **do**
  - **if** $MC_j^t(h) = -1$ **then** $MC_j^t(h) \leftarrow \alpha'$.
  - **else if** $MC_j^t(h) = \alpha'$ **then** do nothing.
  - **else** store all the information about the inconsistency.
- **if** there was at least one inconsistency **then** report the information about the error, and optionally stop the execution (in case of not activating steps *2* and *3*.)

e. $T_j^t \leftarrow$ *Filter 3* (T_j^t, C_t).
3: *(ACTIVATE OR NOT)* Generate, from T_j^t, sub-tables formed by sets of *mutually consistent* blocks, in a maximal way in T_j^t (by the inclusion relationship). This will produce a set of sub-tables $T_{j,i}^t, i = 1, \ldots, s$.
4: *(ACTIVATE OR NOT)* Randomly select one table from $T_{j,i}^t, i = 1, \ldots, s$: $T_{j,z}^t$
5: $a \leftarrow 1$
6: **repeat**
7: Add up the values of each row in $T_{j,z}^t$. Filter the rows whose sum is 0.
8: Each element of the table is divided by the sum of the corresponding row.
9: For each pair $(obj, membr)$ and $(obj', e) \in C_t^a$, if the object in C_t^a has multiplicity $mult > 0$, all the elements of the corresponding row in $T_{j,z}^t$ are multiplied by $mult$, by the corresponding value in $T_{j,z}^t$ (calculated in the previous step), and by the original value in T_j. That is, if in the position (x, y) of the table T_j there is a value different than $-$, and the corresponding object in the row x has multiplicity $mult_{x,a,t}$ in C_t^a, then:

$$T_{j,z}^t(x,y) = \lfloor mult_{x,a,t} \cdot T_{j,z}^t(x,y) \cdot \frac{T_{j,z}^t(x,y)}{RowSum_{x,t}} \rfloor = \lfloor mult_{x,a,t} \cdot \frac{(T_{j,z}^t(x,y))^2}{RowSum_{x,t}} \rfloor$$

10: For each block b (i.e. column) appearing (i.e. not filtered) in $T_{j,z}^t$, calculate the *minimum* number of the previously calculated values, $N_b^a \geq 0$. This is the number of times the block is going to be applied. This value is accumulated to the total calculated through the iteration of the loop over a: $B^j \leftarrow B^j \cup \{< b, N_b^a >\}$
11: $C_t^{a+1} \leftarrow C_t^a - LHS(b) \cdot N_b^a$ ▷ (Delete the left-hand side of the block.)
12: $T_{j,z}^t \leftarrow$ *Filter 3*(*Filter 2* $(T_{j,z}^t, C_t^{a+1})$,C_t^{a+1}), that is, apply filters 2 and 3.
13: $a \leftarrow a + 1$
14: **until** $(a > A) \vee$ *(all the selected minimums in step 10 are* 0*)*
15: **end for**

In order to efficiently repeat the loop for A, and also before going to the next phase (maximality), it is interesting to apply again filter 2. In this way, blocks

updating the configuration and without more applications, are deleted from the table.

After phase 1, some objects may be left unevolved. It can come from the issue of having a low A value, or because the rounded value calculated in the distribution process. Due to the maximal property of P systems, after each computation step no object can be left unevolved. In order to sort out this problem, a maximality phase is applied. This phase consists of selecting those blocks whose rules can still be applied. Then, a random order on these blocks is obtained. Finally, these blocks are applied by following that order. In this phase, each rule block is applied on a maximal manner. That is, blocks consume all objects which can be consumed. In order to minimize the distortion towards the most probable blocks, this phase is left after phase 1, as a residual number of objects is expected to be consumed in this phase.

Algorithm 6 SELECTION PHASE 2: MAXIMALITY

1: **for** $j = 1$ **to** m **do** ▷ (For each environment e_j)
2: Set a random order to the blocks remaining in the last updated table $T^t_{j,z}$ in Phase 1, step *12*.
3: **for** each block b, following the previous random order **do**
4: Calculate the number of applications, N_b, of b in C^A_t (last updated configuration in Phase 1, step *11*).
5: Add N_b to the total number of applications calculated for b in each loop of phase 1, step *10*: $B^j \leftarrow B^j \cup \{< b, N_b >\}$
6: $C^A_t \leftarrow C^A_t - LHS(b) \cdot N_b$ ▷ (delete the objects in the left-hand side of block b, N_b times.)
7: $C'_t \leftarrow C^A_t$
8: **end for**
9: **end for**

After the application of the phases 1 and 2, a maximal multiset of selected (mutually consistent) blocks is computed. The output of the selection stage is, in fact, a maximal multiset of selected rules. Hence, phase 3 (algorithm 7) passes from blocks to rules, by applying the corresponding probabilities (at the local level of blocks). The rules belonging to a block are selected according to a multinomial distribution $M(N, p_1, \ldots, p_k)$, where N is the number of applications of the block, and $p_1, \ldots, p_k$ are the probabilities associated with the rules within the block $r_1, \ldots, r_k$, respectively.

Algorithm 7 SELECTION PHASE 3: PROBABILITY

1: **for** $j = 1$ **to** m **do** ▷ (For each environment e_j)
2: **for all** block $< b, N_b > \in B^j$ **do**
3: Calculate a random multinomial $M(N_b, p_{r_1}, \ldots, p_{r_{l_b}})$ with respect to the probabilities of the l_b rules $r_1, \ldots, r_{l_b}$ within the block b.
4: **for** $i = 1$ **to** l_b **do** Add the randomly calculated value n_{r_i}, using the multinomial distribution for rule r_i, to the multiset of selected rules $R^j_{sel,t} \leftarrow R^j_{sel,t} \cup \{< r_i, n_{r_i} >\}$.
5: **end for**
6: **end for**
7: Delete the multiset of selected blocks $B^j \leftarrow \emptyset$. This is useful for the next step over time T.
8: **end for**

Once the rules to be applied on the current simulation step are selected, the execution stage (algorithm 8) is applied. This stage consists on executing the previously selected multiset of rules. As the objects present on the left hand side of these rules have already been consumed, the only operation left is the application of the right-hand side of the selected rules. Therefore, for each selected rule, the objects present on the right-hand side to the corresponding membranes are added and the indicated membrane charge is set.

Algorithm 8 EXECUTION

1: **for** $j = 1$ **to** m **do** ▷ (For each environment e_j)
2: **for all** Rule $< r, n > \in R^j_{sel}$ **do** ▷ (Apply the right-hand side of the selected rules)
3: $C'_t \leftarrow C'_t + n \cdot RHS(r)$
4: Update the electrical charges of C'_t from $RHS(r)$.
5: **end for**
6: Delete the multiset of selected rules $R^j_{sel} \leftarrow \emptyset$. This is useful for the next step over time T.
7: **end for**

3.3 Running a test example

Let us consider a test example, without any biological meaning, in order to show the different behaviour of the algorithms. This test PDP system is of degree $(2, 1)$, and of the following form:

$$\Pi_{test} = (G, \Gamma, \mu, R, T, \{f_r : r \in R\}, \mathcal{M}_e, \mathcal{M}_1, \mathcal{M}_2)$$

where:

- G is an empty graph because $R_E = \emptyset$.

- $\Gamma = \{a, b, c, d, e, f, g, h\}$
- $\mu = [\ [\ ]_2\ ]_1$ is the membrane structure, and the corresponding initial multisets are:
  - $\mathcal{M}_e = \{\ b\ \}$ (in the environment)
  - $\mathcal{M}_1 = \{\ a^{60}\ \}$ (in membrane 1)
  - $\mathcal{M}_2 = \{\ a^{90}\ b^{72}\ c^{66}\ d^{30}\ \}$ (in membrane 2)
- $T = 1$, only one time step.
- The rules R to apply are:

$$r_{1.1} \equiv [\ a^4\ b^4\ c^2\ ]_2 \xrightarrow{0.7} e^2\ [\ ]_2$$

$$r_{1.2} \equiv [\ a^4\ b^4\ c^2\ ]_2 \xrightarrow{0.2} [\ e^2\ ]_2$$

$$r_{1.3} \equiv [\ a^4\ b^4\ c^2\ ]_2 \xrightarrow{0.1} [\ e\ f\ ]_2$$

$$r_2 \equiv [\ a^4\ d\ ]_2 \xrightarrow{1} f^2[\ ]_2$$

$$r_3 \equiv [\ b^5\ d^2\ ]_2 \xrightarrow{1} g^2[\ ]_2$$

$$r_4 \equiv b\ [\ a^7\ ]_1^- \xrightarrow{1} [\ h^{100}\ ]_1^-$$

$$r_5 \equiv a^3\ [\ ]_2 \xrightarrow{1} [\ e^3\ ]_2$$

$$r_6 \equiv a\ b\ [\ ]_2 \xrightarrow{1} [\ g^3\ ]_2^-$$

We can construct a set of six consistent rule blocks $B_{\Pi_{test}}$ (of the form $b_{h,\alpha,\alpha',u,v}$) from the set R of Π_{test} as follows:

- $b_1 \equiv b_{2,0,0,\emptyset,\{a^4,b^4,c^2\}} = \{r_{1.1}, r_{1.2}, r_{1.3}\}$
- $b_2 \equiv b_{2,0,0,\emptyset,\{a^4,d\}} = \{r_2\}$
- $b_3 \equiv b_{2,0,0,\emptyset,\{b^5,d^2\}} = \{r_3\}$
- $b_4 \equiv b_{1,-,-,\{b\},\{a^7\}} = \{r_4\}$
- $b_5 \equiv b_{2,0,0,\{a^3\},\emptyset} = \{r_5\}$
- $b_6 \equiv b_{2,0,-,\{a,b\},\emptyset} = \{r_6\}$

It is noteworthy that the set $B_{\Pi_{test}}$ is not mutually consistent. However, only the blocks b_1, b_2, b_3 and b_5 are applicable in the initial configuration, and they, in fact, conform a mutually consistent set of blocks. Block b_4 is not applicable since the charge of membrane 1 is neutral, and block b_6 cannot be applied because there are no b's in membrane 1.

Table 1 shows five different runs for one time step of Π_{test} using the DNDP algorithm. The values refers to the number of applications for each rule, which is

Rules	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Simulation 5
$r_{1.1}$	11	0	0	0	0
$r_{1.2}$	4	4	3	0	0
$r_{1.3}$	1	0	0	0	0
r_2	6	18	6	22	2
r_3	1	6	12	4	14
r_4	-	-	-	-	-
r_5	20	20	20	20	20
r_6	-	-	-	-	-

Table 1: Simulating Π_{test} using the DNDP algorithm

actually the output of the selection stage (and the input of the execution stage). Note that for simulation 1, the applications for $r_{1.1}$, $r_{1.2}$ and $r_{1.3}$ follows the multinomial distribution. The applications of these rules are reduced because they are competing with rules r_2 and r_3, . However, this competition leads to situations where the applications of the block b_1 does not follow a multinomial distribution. It comes from the fact of using a random order over the rules, but not over the blocks. Rules having a probability equals to 1 are more restrictive on the competitions because they are applied in a maximal way in their turn. This is the reason because on simulations 4 and 5, none of the rules $r_{1.i}, 1 \leq i \leq 3$ are applied.

This behaviour could create a distortion of the reality described in the simulated model. But it is usually appeased running several simulations and making a statistical study. Finally, rules not competing for objects are applied as is, in a maximal way. For example, rule r_5 is always applied 20 times because its probability is equal to 1.

In the following, the test example is executed using the DCBA. The main results of the different phases of the process is also detailed.

In the initialization phase, the static table is created, containing all the consistent blocks. The static table of the Π_{test} P system is showed in table 2. As shown, the values inside the cells of the table represents the inverse $(1/k)$ of the multiplicity of the object (in the membrane, as specified in the row) inside the block indicated in the header of the column.

Once the static table has been initialized, the simulation main loop runs for the stated steps. Then, for each step of computation, the selection and execution of rules runs, as illustrated in the following paragraphs.

The selection starts with the distribution phase. The needed filters are performed, causing some objects and blocks to be discarded, as they need not present charges and/or objects. Then the corresponding calculus take place, getting the minimum number of applications of each way. The result of the selection phase 1 of the step 1 is showed in table 3. The sum of the previously obtained values is showed in the last column. Then, the possible number of applications of a block is calculated for each object, considering its multiplicity in the current configuration

Objects	**Consistent Blocks**					
	$b_{2,0,0,\emptyset,\{a^4,b^4,c^2\}}$	$b_{2,0,0,\emptyset,\{a^4,d\}}$	$b_{2,0,0,\emptyset,\{b^5,d^2\}}$	$b_{1,-,-,\{b\},\{a^7\}}$	$b_{2,0,0,\{a^3\},\emptyset}$	$b_{2,0,-,\{a,b\},\emptyset}$
$< a$,2>	1/4	1/4	-	-	-	-
$< b$,2>	1/4	-	1/5	-	-	-
$< c$,2>	1/2	-	-	-	-	-
$< d$,2>	-	1/1	1/2	-	-	-
$< a$,1>	-	-	-	1/7	1/3	1/1
$< b$,1>	-	-	-		-	1/1
$< b$,e>	-	-	-	1/1	-	-

Table 2: Static table

and the block, and the relation with the sum of the row. This relation somehow captures the proportion of objects to be initially assigned to each block. Then, the minimum number of each block (given by the column) is calculated.

Objects	**Consistent Blocks**				**Sum**
	$b_{2,0,0,\emptyset,\{a^4,b^4,c^2\}}$	$b_{2,0,0,\emptyset,\{a^4,d\}}$	$b_{2,0,0,\emptyset,\{b^5,d^2\}}$	$b_{2,0,0,\{a^3\},\emptyset}$	
$< a$,2> * 90	0.25 | 11	0.25 | 11	-	-	0.5
$< b$,2> * 72	0.25 | 10	-	0.2 | 6	-	0.45
$< c$,2> * 66	0.5 | 33	-	-	-	0.5
$< d$,2> * 30	-	1.0 | 20	0.5 | 5	-	1.5
$< a$,1> * 60	-	-	-	0.33 | 20	0.33
Applications	10	11	5	20	

Table 3: Selection Phase 1 - Distribution

The next phase, maximality, starts from the remaining objects, selecting new applications of the blocks in a maximal way. The result of this phase is showed in table 4. This table presents the remaining objects (the ones not assigned in phase 1) and the possible blocks to be selected. The blocks are chosen in a random way, as shown in algorithm 6, and the possible applications of the block are calculated. This process guarantees a maximal set of blocks to be selected, with a maximal number of applications of each block. The last row, applications, shows that the block $b_{2,0,0,\emptyset,\{a^4,b^4,c^2\}}$ is applying 1 time, additional to the number of applications calculated in the distribution phase.

Then the phase 3, probability, take place. For each block selected in the previous phases, its number of applications is divided among the rules being part of the

Objects	**Consistent Blocks**		
	$b_{2,0,0,\emptyset,\{a^4,b^4,c^2\}}$	$b_{2,0,0,\emptyset,\{a^4,d\}}$	$b_{2,0,0,\emptyset,\{b^5,d^2\}}$
$< a,2 >$ * 6	-	-	-
$< b,2 >$ * 7	-	-	-
$< c,2 >$ * 46	-	-	-
$< d,2 >$ * 9	-	-	-
Applications	1	-	-

Table 4: Selection Phase 2 - Maximality

block, according to their probabilities. As a result, the number of applications of each rule is obtained, as showed in table 5.

Rules	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Simulation 5
$r_{1.1}$	7	10	7	6	7
$r_{1.2}$	3	0	4	1	2
$r_{1.3}$	1	1	5	3	1
r_2	11	11	11	12	12
r_3	5	5	5	6	6
r_4	-	-	-	-	-
r_5	20	20	20	20	20
r_6	-	-	-	-	-

Table 5: Simulating Π_{test} using the DCBA algorithm

It is noteworthy that the selection of rules belonging to block 1 $\{r_{1.i}, 1 \leq i \leq 3\}$, in table 5, always follows a multinomial distribution respecting the 3 probabilities. This solves the drawback we showed on table 1. Moreover, it can be seen that the maximality sometimes can give one more application to blocks 2 and 3, in spite of keeping the original 10 applications for block 1 from phase 1. In any case, the number of applications is proportionally distributed, avoiding the distortion of using a random order over the blocks (or rules), as made in the DNDP algorithm.

3.4 Implementation in pLinguaCore

In [11], a Java library called *pLinguaCore* was presented under GPL license. It includes parsers to handle input files and built–in simulators to handle different P System based models. It is not a closed product because developers with knowledge of Java can add new components to the library. Within the scope of this paper, pLinguaCore has been upgraded to provide an implementation of the DBCA,

thus extending its existing probabilistic model simulation algorithms support. Along with the inclusion of other extensions, regarding to models such as Spiking Neural P Systems and Numerical P Systems, current version of the library, named *pLinguaCore 3.0*, and featuring an implementation of the introduced DBCA can be downloaded from [21]. In what follows, details of the implementation of the DBCA in pLinguaCore are shown. Data structures, methods, code optimization and bug fixes are reviewed. Going Top-down, Java classes involved in the implementation:

- *DynamicMatrix.* It provides an implementation for the main operations of the DBCA.

  DynamicMatrix is built as a dynamic map indexed by MatrixKey class objects. MatrixKey objects are implemented as a pair of (MatrixRow, MatrixColumn) class objects. Associated to each MatrixKey object within the map, multiplicity k of the object specified by the *MatrixRow row* in the left hand side of the rule specified by the *MatrixColumn column* is stored. Note that k is stored instead of $1/k$ for accuracy reasons.

  As different filters are applied over the DynamicMatrix object, a couple of lists of MatrixRow and MatrixColumn objects respectively are associated to the matrix to keep track of its *valid cells*. Removal of elements from these lists is performed when filters are applied, while the DynamicMatrix object itself is reset in every step of the main loop of selection phase. Thus, DynamicMatrix object can be viewed as a *hash table* of multiplicities that allows a significant reduction of the required amount of memory for execution of the DBCA.

  Also, attributes that stores the sum of the multiplicities of the objects in the matrix by row as well as the minimum of the columns are included in the DynamicMatrix class. Inconsistent blocks are controlled by means of a list of pairs of MatrixColumn objects.

  DynamicMatrix class directly extends from StaticMatrix class. Methods in DynamicMatrix implements the DBCA different phases themselves, remarkably:

  - *initData()* initializes valid rows and columns lists in the DynamicMatrix object, clearing up them; also application of rules data structure is initialized.
  - *filterColumns1()* computes valid columns and associates them to the DynamicMatrix object; applies Filter 1 to these columns;
  - *filterColumns2()* applies Filter 2 over valid columns associated to the DynamicMatrix object, removing the required ones.
  - *checkMutualConsistency()* checks mutual consistency over blocks of the DynamicMatrix object; if any inconsistency is found, an exception is thrown and execution of the simulator is halted; a message listing the mutual inconsistent blocks found is shown to the user.

  - *initFilterRows()* computes valid rows and associates them to the DynamicMatrix object; applies Filter 3 to these rows.
  - *filterRows()* applies Filter 3 to valid rows, removing the required ones; this method is called inside the main loop of the selection phase, while the previous one is called outside, at the beginning of this phase.
  - *normalizeRowsAndCalculateMinimums()* implements the main loop of selection phase.
  - *maximality()* implements maximality phase.
  - *executeRules()* implements execution phase; remarkably, multinomial distribution is computed by computing binomial distributions, implemented through the specialized CERN Java library (`cern.jet.random.Binomial`).

- *StaticMatrix.* Provides an implementation for the static matrix used by the DBCA. Similarly to DynamicMatrix class, cells within the matrix are stored as a map indexed by MatrixKey class objects, each one of them associated to a multiplicity. A couple of immutable lists of MatrixRow and MatrixColumn class objects determines the structure of the matrix. Contents of the cells are fixed once initialized.

- *MatrixRow.* Provides an implementation for rows featured in DynamicMatrix, StaticMatrix and MatrixKey objects. Implemented by a pair of String objects representing *object and membrane label* respectively, it also provides a method for computing the validity of the row, i.e. to determine if the row has to be kept within the DynamicMatrix object with respect to a given environment.

- *MatrixColumn.* Provides an implementation for columns featured in DynamicMatrix, StaticMatrix and MatrixKey objects. An abstract class, its extended and implemented by a couple of classes representing the two kinds of rule blocks:

  - SkeletonRulesBlock, which implements blocks of skeleton rules.

  - EnvironmentRulesBlock, which implements blocks of environment rules.

  Both classes have the same structure: a *single* object to store the common left hand rule side of the rule, plus a *collection* to store the several right hand rule side objects that conforms the block. Also, each one provides an specific method for computing the validity of the corresponding column within the dynamic matrix.

To conclude, let us note that while conducting the DBCA implementation, several bugs have been fixed in pLinguaCore, notably some of them regarding to the way in which rules are parsed and stored, thus applying beyond the scope of

the DBCA an affecting to implementation of probabilistic models simulators as a whole:

- Multisets of objects are now taken into account while checking rule blocks. In previous versions of pLinguaCore, when checking of the consistency of probabilities of a rule block was conducted (i.e. checking that sum of probabilities of the rules must equal to one), multiplicities of objects in the left hand side of the rules were ignored.

- Issues with "intentional duplicate rules" solved assigning an unique identifier for every rule within the scope of probabilistic models. Issues found were:

  – Instantiation of parameters in syntactically different rule schemes for some models produced duplicated rules and caused the parser to throw an error and halt. As this duplicity proved intentional, the parser was modified subsequently to take it into account.

  – Probability was not taken into account when differencing rules. This made the parser to discard a rule syntactically identical, except for its probability, to a previous parsed one.

4 Validation

4.1 Improved model for the scavenger bird ecosystem

In this section, it is presented a novel model for an ecosystem related to the Bearded Vulture in the Pyrenees (NE Spain), by using PDP systems. This model is an improved model of which is provided in [5]. The Bearded Vulture (*Gypaetus barbatus*) is an endangered species in Europe that feeds almost exclusively on bone remains of wild and domestic ungulates. In this model, the evolution of six species is studied: The Bearded Vulture and five subfamilies of domestic and will ungulates upon which the vulture feeds.

The model consists of a PDP system of degree $(2, 1)$,

$$\Pi = (G, \Gamma, \mu, R, T, \{f_r : \ r \in R\}, \mathcal{M}_1, \mathcal{M}_2)$$

where:

- G is an empty graph because $R_E = \emptyset$.
- In the alphabet Γ, we represent the six species of the ecosystem (index i is associated with the species and index j is associated with their age, and the symbols X, Y and Z represent the same animal but in different states); it

also contains the auxiliary symbol B, which represents 0.5 kg of bones, and C, which allows a change in the polarization of the membrane labeled by 2 at a specific stage.

$$\Gamma = \{X_{i,j}, Y_{i,j}, Z_{i,j} : 1 \leq i \leq 7, 0 \leq j \leq k_{i,4}\} \cup \{B, C\}$$

The species are the following:
- Bearded Vulture ($i = 1$)
- Pyrenean Chamois ($i = 2$)
- Red Deer Female ($i = 3$)
- Red Deer Male ($i = 4$)
- Fallow Deer ($i = 5$)
- Roe Deer ($i = 6$)
- Sheep ($i = 7$)

- $\mu = [\ [\ ]_2\ ]_1$ is the membrane structure, and the corresponding initial multisets are:
  - $\mathcal{M}_1 = \{\ X_{i,j}^{q_{i,j}} : 1 \leq i \leq 7, 0 \leq j \leq k_{i,4}\}$
  - $\mathcal{M}_2 = \{\ C, B^{\alpha}\}$

    where $\alpha = \lceil \sum_{j=1}^{21} q_{1,j} \cdot 1.10 \cdot 682 \rceil$

    Value α represents an external contribution of food which is added during the first year of study so that the Bearded Vulture survives. In the formula, $q_{1,j}$ represents the number of j years of age of Bearded Vultures, the finality of constant factor 1.10 is to guarantee enough food for 10% population growth. At present, the population growth is estimated an average 4%, but this value can reach higher values. Thus, to avoid problems related with the underestimation of this value the first year we estimated the population growth (overestimated) at 10%. The constant value 682 represents the amount of food needed per year for a Bearded Vulture pair to survive.

- Each year in the real ecosystem is simulated by 3 computational steps, so $T = 3 \cdot Years$, where $Years$ is the number of years to simulate.

- The rules R to apply are:
  - Reproduction rules for ungulates

    Adult males

    $r_{0,i,j} \equiv [X_{i,j}]_1 \xrightarrow{1-k_{i,13}} [Y_{i,j}]_1 : k_{i,2} \leq j \leq k_{i,4}, 2 \leq i \leq 7$

    Adult females that reproduce

    $r_{1,i,j} \equiv [X_{i,j}]_1 \xrightarrow{k_{i,5}k_{i,13}} [Y_{i,j}, Y_{i,0}]_1 : k_{i,2} \leq j < k_{i,3}, 2 \leq i \leq 7, i \neq 3$

    Red Deer females produce 50% of female and 50% of male springs

    $r_{2,j} \equiv [X_{3,j}]_1 \xrightarrow{k_{3,5}k_{3,13}{}^{0.5}} [Y_{3,j}Y_{3,0}]_1 : k_{3,2} \leq j < k_{3,3}$

    $r_{3,j} \equiv [X_{3,j}]_1 \xrightarrow{k_{3,5}k_{3,13}{}^{0.5}} [Y_{3,j}Y_{4,0}]_1 : k_{3,2} \leq j < k_{3,3}$

Fertile adult females that do not reproduce

$r_{4,i,j} \equiv [X_{i,j}]_1 \xrightarrow{(1-k_{i,5})k_{i,13}} [Y_{i,j}]_1 : k_{i,2} \leq j < k_{i,3}, 2 \leq i \leq 7$

Not fertile adult females

$r_{5,i,j} \equiv [X_{i,j}]_1 \xrightarrow{k_{i,13}} [Y_{i,j}]_1 : k_{i,3} \leq j \leq k_{i,4}, 2 \leq i \leq 7$

Young ungulates that do not reproduce

$r_{6,i,j} \equiv [X_{i,j}]_1 \xrightarrow{1} [Y_{i,j}]_1 : 0 \leq j < k_{i,2}, 2 \leq i \leq 7$

– Growth rules for the Bearded Vulture

$r_{7,j} \equiv [X_{1,j}]_1 \xrightarrow{k_{1,6}+k_{1,10}} [Y_{1,k_{1,2}-1}Y_{1,j}]_1 : k_{1,2} \leq j < k_{1,4}$

$r_{8,j} \equiv [X_{1,j}]_1 \xrightarrow{1-k_{1,6}-k_{1,10}} [Y_{1,j}]_1 : k_{1,2} \leq j < k_{1,4}$

$r_9 \equiv [X_{1,k_{1,4}}]_1 \xrightarrow{k_{1,6}} [Y_{1,k_{1,2}-1}Y_{1,k_{1,4}}]_1$

$r_{10} \equiv [X_{1,k_{1,4}}]_1 \xrightarrow{1-k_{1,6}} [Y_{1,k_{1,4}}]_1$

– Mortality rules for ungulates

Young ungulates which survive

$r_{11,i,j} \equiv Y_{i,j}[\,]_2 \xrightarrow{1-k_{i,7}-k_{i,8}} [Z_{i,j}]_2 : 0 \leq j < k_{i,1}, 2 \leq i \leq 7$

Young ungulates which die

$r_{12,i,j} \equiv Y_{i,j}[\,]_2 \xrightarrow{k_{i,8}} [B^{k_{i,11}}]_2 : 0 \leq j < k_{i,1}, 2 \leq i \leq 7$

Young ungulates which are retired from the ecosystem

$r_{13,i,j} \equiv Y_{i,j}[\,]_2 \xrightarrow{k_{i,7}} [\,]_2 : 0 \leq j < k_{i,1}, 2 \leq i \leq 7$

Adult ungulates that do not reach the average life expectancy

Those which survive

$r_{14,i,j} \equiv Y_{i,j}[\,]_2 \xrightarrow{1-k_{i,10}} [Z_{i,j}]_2 : k_{i,1} \leq j < k_{i,4}, 2 \leq i \leq 7$

Those which die

$r_{15,i,j} \equiv Y_{i,j}[\,]_2 \xrightarrow{k_{i,10}} [B^{k_{i,12}}]_2 : k_{i,1} \leq j < k_{i,4}, 2 \leq i \leq 7$

Ungulates that reach the average life expectancy

Those which die in the ecosystem

$r_{16,i} \equiv Y_{i,k_{i,4}}[\,]_2 \xrightarrow{k_{i,9}+(1-k_{i,9})k_{i,10}} [B^{k_{i,12}}]_2 : 2 \leq i \leq 7$

Those which die and are retired from the ecosystem

$r_{17,i} \equiv Y_{i,k_{i,4}}[\,]_2 \xrightarrow{(1-k_{i,9})(1-k_{i,10})} [\,]_2 : 2 \leq i \leq 7$

– Mortality rules for the Bearded Vulture

$r_{18,j} \equiv Y_{1,j}[\,]_2 \xrightarrow{1-k_{1,10}} [Z_{1,j}]_2 : k_{1,2} \leq j < k_{1,4}$

$r_{19,j} \equiv Y_{1,j}[\,]_2 \xrightarrow{k_{1,10}} [\,]_2 : k_{1,2} \leq j < k_{1,4}$

$r_{20} \equiv Y_{1,k_{1,4}}[\,]_2 \xrightarrow{1} [Z_{1,k_{1,2}-1}]_2$

$r_{21} \equiv Y_{1,k_{1,2}-1}[\,]_2 \xrightarrow{1} [Z_{1,k_{1,2}-1}]_2$

– Feeding rules

$r_{22,i,j} \equiv [Z_{i,j}B^{k_{i,14}}]_2 \xrightarrow{1} X_{i,j+1}[\,]_2^+ : 0 \leq j \leq k_{i,4}, 1 \leq i \leq 7$

– Balance rules

Elimination of remaining bones

$r_{23} \equiv [B]_2^+ \xrightarrow{1} [\,]_2$

Adult animals that die because they have not enough food

$r_{24,i,j} \equiv [Z_{i,j}]_2^+ \xrightarrow{1} [B^{k_{i,12}}]_2 : k_{i,1} \leq j \leq k_{i,4}, 1 \leq i \leq 7$

Young animals that die because the have not enough food

$r_{25,i,j} \equiv [Z_{i,j}]_2^+ \xrightarrow{1} [B^{k_{i,11}}]_2 : 0 \leq j < k_{i,1}, 1 \leq i \leq 7$

Change the polarization

$r_{26} \equiv [C]_2^+ \xrightarrow{1} [C]_2$

- The constants associated with the rules have the following meaning:
  - $k_{i,1}$: Age at which adult size is reached. This is the age at which the animal consumes food as an adult does, and at which, if the animal dies, the amount of biomass it leaves behind is similar to the total left by an adult. Moreover, at this age it will have surpassed the critical early phase during which the mortality rate is high.
  - $k_{i,2}$: Age at which it begins to be fertile.
  - $k_{i,3}$: Age at which it stops being fertile.
  - $k_{i,4}$: Average life expectancy in the ecosystem.
  - $k_{i,5}$: Fertility ratio (number of descendants by fertile females).
  - $k_{i,6}$: Population growth (this quantity is expressed in terms of 1).
  - $k_{i,7}$: Animals retired from the ecosystem in the first years, age $< k_{i,1}$ (this quantity is expressed in terms of 1).
  - $k_{i,8}$: Natural mortality ratio in first years, age $< k_{i,1}$ (this quantity is expressed in terms of 1).
  - $k_{i,9}$: 0 if the live animals are retired at age $k_{i,4}$, in other cases, the value is 1.
  - $k_{i,10}$: Mortality ratio in adult animals, age $\geq k_{i,1}$ (this quantity is expressed in terms of 1).
  - $k_{i,11}$: Amount of bones from young animals, age $< k_{i,1}$.
  - $k_{i,12}$: Amount of bones from adult animals, age $\geq k_{i,1}$.
  - $k_{i,13}$: Proportion of females in the population (this quantity is expressed in terms of 1).

  - $k_{i,14}$: Amount of food necessary per year and breeding pair (1 unit is equal to 0.5 kg of bones).

- In [5], they can be found actual values for the constants associated with the rules as well as actual values for the initial populations $q_{i,j}$ for each species i with age j. There are two sets of initial populations values, one beginning on year 1994 and another one beginning on year 2008.

4.2 Simulation results

In [5], a simulator for the model was presented. The authors show a comparison of the results provided by the simulator and actual data obtained from the ecosystem. That simulator was written in C++ and the rules were implemented directly on the source code. So, that is a simulator implemented *ad hoc* for the model. The simulator does not implement any described simulation algorithm for P systems and does not implement any generic method to define P systems. We have found that *ad hoc* simulators like the one presented in [5] have a strong coupling design and it is a problem for debugging. So, if the simulator does not reproduce the expected behaviour of the model, what is causing the problem?. In that situation, we could think that:

1. The model is wrong.
2. The rules are not correctly written in the source code.
3. The semantics of the model is not correctly implemented in the source code.

It is very difficult to find the cause of the problem with a strong coupling software design. Moreover, if we think that the cause of the problem is, for instance, 2, but it is really 1 or 3, then we can introduce new errors trying to correct it.

From a software engineering point of view it is very important to decouple software components, that is the point of view of P-Lingua and pLinguaCore [21]:

- The model is designed on a paper.
- The rules are written on a P-Lingua file. So, the parser checks the syntactical/semantics errors.
- The semantics of the model is implemented on the pLinguaCore library following a good described simulation algorithm.

PLinguaCore is a simulation library that accepts the input written in P-Lingua and provides simulations of the defined P systems. For each type of P system, there are one or more simulation algorithms implemented in pLinguaCore. It is a software framework, so it can be expanded with new simulation algorithms.

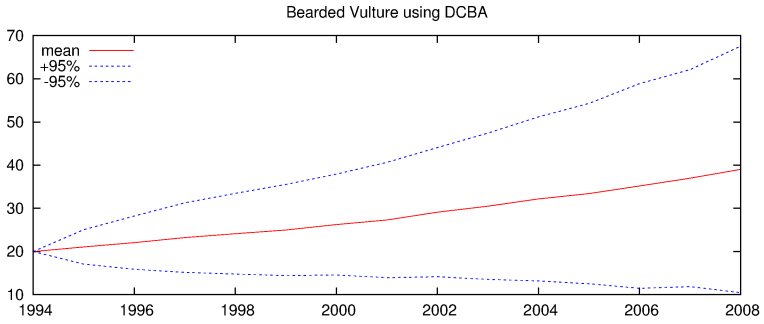
Thus, we have expanded the pLinguaCore library to include the DCBA simulation algorithm for PDP systems, the current version of pLinguaCore is 3.0 and it can be downloaded from [21].

In this section, we use the model of the Bearded Vulture described above to compare the simulation results produced by the pLinguaCore library using two different simulation algorithms: DNDP [14] and DCBA. We also compare the

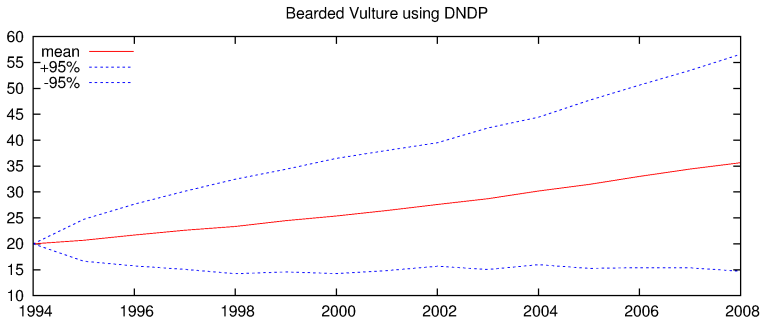
results of the implemented simulation algorithms with the results provided by the C++ *ad hoc* simulator and with the actual ecosystem data obtained from [5]. In [22] it can be found the P-Lingua file which defines the model and instructions to reproduce the comparisons.

We have set the initial population values with the actual ecosystem values for year 1994. For each simulation algorithm we have made 1000 simulations of 14 years, that is, 42 computational steps. The simulation workflow have been implemented on a Java program that runs over the pLinguaCore library (this Java program can be downloaded from [22]). For each simulated year (3 computational steps), the Java program counts the number of animals for each species i, that is: $X_i = \sum_{j=0}^{k_{i,4}} X_{i,j}$. After 1000 simulations, the Java program calculates average values for each year and species and writes the output to a text file. Finally, we have used the GnuPlot software [20] to produce population graphics.

In figures 1, 2, 3, 4 ,5, 6 and 7 the population graphics for each species and simulation algorithm are represented.



(a) Using DCBA



(b) Using DNDP

Fig. 1: Evolution of the Bearded Vulture birds

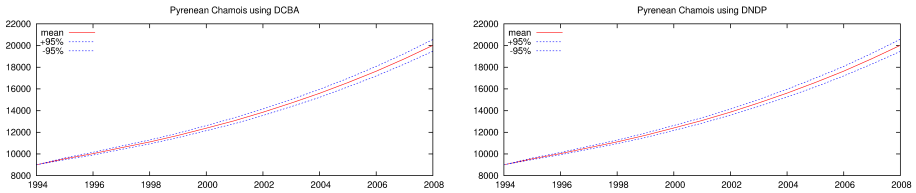


Fig. 2: Evolution of the Pyrenean Chamois

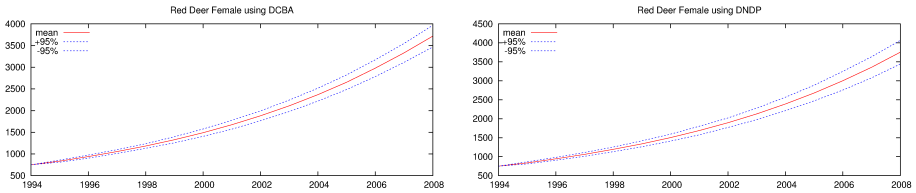


Fig. 3: Evolution of the female Red Deer

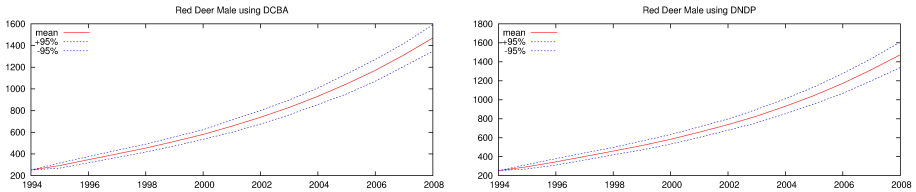


Fig. 4: Evolution of the male Red Deer

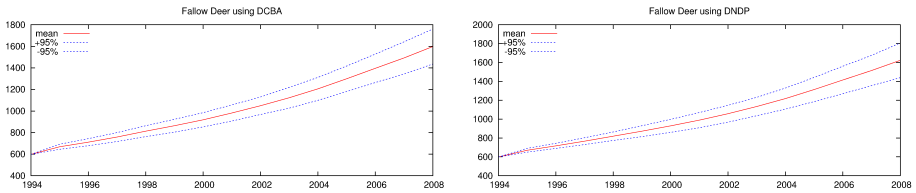


Fig. 5: Evolution of the Fallow Deer

The main difference between DNDP and DCBA algorithms is the way the algorithms distribute the objects between different rule blocks that compete for

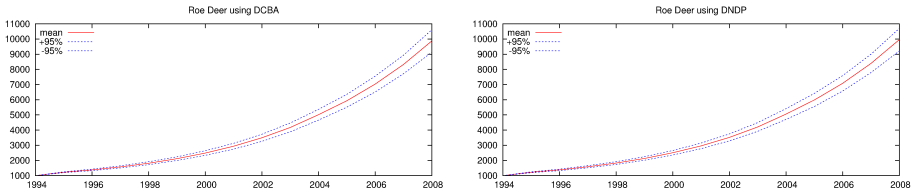


Fig. 6: Evolution of the Roe Deer

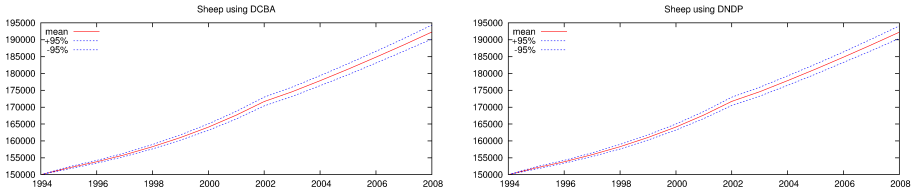


Fig. 7: Evolution of the Sheep

the same objects. In the model, the behavior of the ungulates are modeled by using rule blocks that do not compete for objects. So, the simulator provides similar results for both DCBA and DNDP algorithms. In the case of the Bearded Vulture, there are a set of rules $r_{22,i,j}$ that compete for B objects because $k_{1,14}$ is not 0 (the Bearded Vulture needs to feed on bones to survive). The $k_{i,14}$ constants are 0 for ungulates, $2 \leq i \leq 7$, because they do not need to feed on bones to survive. The initial amount of bones and the amount of bones generated during the simulation is enough to support the Bearded Vulture population regardless the way the simulation algorithm distributes the bones between vultures of different ages (rules $r_{22,1,j}$). By the way, there are a small initial population of bearded vultures (20 pairs), because of that, we can see differences between the results with DCBA, DNDP, C++ simulator and actual ecosystem data for the Bearded Vulture (39 bearded vultures with DCBA for year 2008, 36 with DNDP, 38 with the C++ simulator and 37 on the actual ecosystem).

In figure 8 it is showed the comparison between the actual data for year 2008 and the simulation results by using the C++ *ad hoc* simulator, the DNDP algorithm and the DCBA algorithm implemented in pLinguaCore. In the case of the Pyrenean Chamois, there is a difference between the actual population data on the ecosystem (12000 animals) and the results provided by the other simulators (above 20000 animals), this is because the population of Pyrenean Chamois was restarted on year 2004 [5]. Taking this into account, we can see that all the simulators behave in a similar way for the above model and they can reproduce the actual data after 14 simulated years. So, the DCBA algorithm is able to reproduce the semantics of PDP systems and it can be used to simulate the behavior of actual ecosystems by means of PDP systems.

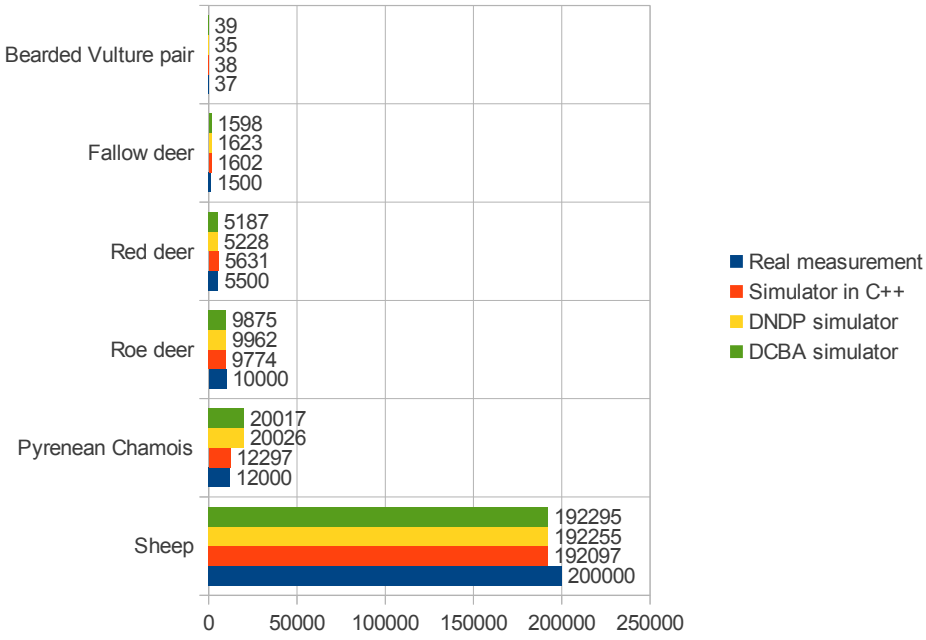


Fig. 8: Data of the year 2008 from: real measurements of the ecosystem, original simulator in C++, simulator using DNDP and simulator using DCBA.

5 Conclusions and Future Work

In this paper we have introduced a novel algorithm for Population Dynamics P systems (PDP systems), called Direct distribution based on Consistent Blocks Algorithm (DCBA). This new algorithm performs an object distribution along the rules that eventually compete for objects. The main procedure is divided into two stages: selection and execution. Selection stage is also divided into three micro-phases: phase 1 (distribution), where by using a table and the construction of rule blocks, the distribution process takes place; phase 2 (maximality), where a random order is applied to the remaining rule blocks in order to assure the maximality condition; and phase 3 (probability), where the number of application of rule blocks is translated to application of rules by using random numbers respecting the probabilities.

By using a test example, it is shown how this new algorithm solves some drawbacks in its predecessor, the DNDP algorithm. Moreover, both algorithms are validated towards a real ecosystem model (the bearded vulture birds), showing that they reproduce the same results than the original simulator written in C++.

Finally, some details and updates of its implementation in the pLinguaCore framework are provided.

The accelerators in High Performance Computing offers new approaches to accelerate the simulation of P systems and Population Dynamics [6]. An initial parallelization work of the DCBA by using multi-core processors is described in [12]. The analysis of the two parallel levels (simulations and environments), and the speedup achieved by using the different cores, make interesting the search for more parallel architectures. For the near future work, we will use the massively parallel architectures inside the graphics cards (GPUs) using CUDA. We will adapt and scale the DCBA algorithm using the CUDA programming model, and develop a parallel simulator for GPU based systems.

Acknowledgments

The authors acknowledge the support of "Proyecto de Excelencia con Investigador de Reconocida Valía" of the "Junta de Andalucía" under grant P08-TIC04200, and the support of the project TIN2009-13192 of the "Ministerio de Educación y Ciencia" of Spain, both co-financed by FEDER funds.

References

1. D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri. Modelling metapopulations with stochastic membrane systems, Biosystems, 91 (2008), 499–514.
2. L. Bianco, V. Manca, L. Marchetti, M. Petterlini. Psim: a simulator for biomolecular dynamics based on P systems, IEEE Congress on Evolutionary Computation (2007), 883–887
3. M. Cardona, M.A. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy. A computational modeling for real ecosystems based on P systems, Natural Computing, 10, 1 (2011), 39-53.
4. M. Cardona, M.A. Colomer, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy. A P system based model of an ecosystem of some scavenger birds, Membrane Computing, 10th International Workshop, LNCS 5957 (2010), 182–195.
5. M. Cardona, M.A. Colomer, M.J. Pérez-Jiménez, D. Sanuy, A. Margalida. Modeling ecosystem using P systems: The bearded vulture, a case study. Membrane Computing, 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers. Lecture Notes in Computer Science, 5391 (2009), 137-156.
6. J.M. Cecilia, J.M. García, G.D. Guerrero, M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez. Simulation of P systems with active membranes on CUDA, Briefings in Bioinformatics, 11, 3 (2010), 313–322
7. S. Cheruku, A. Păun, F.J. Romero-Campero, M.J. Pérez-Jiménez, O.H. Ibarra. Simulating FAS-induced apoptosis by using P systems, Progress in Natural Science, 17, 4 (2007), 424–431.

8. M.A. Colomer, S. Lavín, I. Marco, A. Margalida, I. Pérez-Hurtado, M.J. Pérez-Jiménez, D. Sanuy, E. Serrano, L. Valencia-Cabrera. Modeling population growth of Pyrenean Chamois (Rupicapra p. pyrenaica) by using P systems, Membrane Computing, 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010, Revised Selected Papers. LNCS, 6501 (2011), 144–159.
9. M.A. Colomer, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos. Comparing simulation algorithms for multienvironment probabilistic P system over a standard virtual ecosystem. Natural Computing, online version (http://dx.doi.org/10.1007/s11047-011-9289-2).
10. M.A. Colomer, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. Simulating Tritrophic Interactions by Means of P Systems. *Proceedings of the 5^{th} IEEE International Conference on Bio-Inspired Computing: Theories and Applications*, vol. 2 (2010), 1621–1628.
11. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, Agustín Riscos-Núñez. An overview of P-Lingua 2.0, Membrane Computing, 10th International Workshop, LNCS 5957 (2010), 264–288.
12. M.A. Martínez-del-Amor, I. Karlin, R.E. Jensen, M.J. Pérez-Jiménez, A.C. Elster. Parallel Simulation of Probabilistic P Systems on Multicore Platforms. In this volume.
13. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, F. Sancho-Caparrini. A simulation algorithm for multienvironment probabilistic P systems: A formal verification. International Journal of Foundations of Computer Science, 22, 1 (2011), 107–118.
14. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez, M.A. Colomer. A new simulation algorithm for multienvironment probabilistic P systems,*Proceedings of the 5^{th} IEEE International Conference on Bio-Inspired Computing: Theories and Applications*, vol. 1 (2010), 59–68,
15. V. Nguyen, D. Kearney, G. Gioiosa. An algorithm for non-deterministic object distribution in P systems and its implementation in hardware, Membrane Computing, 9th International Workshop, LNCS 5391 (2009), 325–354.
16. G. Păun. Computing with membranes. Journal of Computer and System Sciences, 61, 1 (2000), pp. 108–143, and Turku Center for Computer Science-TUCS Report No 208.
17. Gh. Păun, F.J. Romero-Campero. Membrane Computing as a Modeling Framework. Cellular Systems Case Studies, Formal Methods for Computational Systems Biology, LNCS, 5016 (2008), 168–214.
18. M.J. Pérez-Jiménez, F.J. Romero-Campero. P systems, a new computational modelling tool for systems biology, Transactions on Computational Systems Biology VI, LNCS 4220 (2006), 176–97.
19. G. Terrazas, N. Krasnogor, M. Gheorghe, F. Bernardini, S. Diggle, M. Cámara. Environment aware P-System model of quorum sensing, New Computational Paradigms, LNCS 3526 (2005), 473–485.
20. The GNUplot web page. `http://www.gnuplot.info`
21. The P-Lingua web page. `http://www.p-lingua.org`
22. The Bearded Vulture ecosystem model in P-Lingua. `http://www.p-lingua.org/wiki/index.php/bvBWMC12`