DCBA: Simulating Population Dynamics P Systems with Proportional Object Distribution

Miguel A. Martínez-del-Amor[1], Ignacio Pérez-Hurtado[1], Manuel García-Quismondo[1], Luis F. Macías-Ramos[1], Luis Valencia-Cabrera[1], Álvaro Romero-Jiménez[1], Carmen Graciani[1], Agustín Riscos-Núñez[1], Mari A. Colomer[2], and Mario 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
{mdelamor,perezh,mgarciaquismondo,lfmaciasr,lvalencia, romero.alvaro,cgdiaz,ariscosn,marper}@us.es
[2] Department of Mathematics
University of Lleida
Avda. Alcalde Rovira Roure, 191, 25198 Lleida, Spain
colomer@matematica.udl.es

Abstract. *Population Dynamics P systems* provide a formal framework for ecological modelling having a probabilistic (while keeping the maximal parallelism). Several simulation algorithms have been developed always trying to reach higher reliability in the way they reproduce the behaviour of the ecosystems being modelled.

It is natural for those algorithms to classify the rules into blocks, comprising rules that share identical left-hand side. Previous algorithms, such as the Binomial Block Based (BBB) or the Direct Non Deterministic distribution with Probabilities (DNDP), do not define a deterministic behaviour for blocks of rules competing for the same resources. In this paper we introduce the Direct distribution based on Consistent Blocks Algorithm (DCBA), a simulation algorithm which addresses that inherent non-determinism of the model by distributing proportionally the resources.

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

1 Introduction

Since the devising of the field of *Membrane Computing* [13,15], it has established as a feasible background for the modelling of biochemical phenomena. Within *Computational Systems Biology*, for example, it is complementary and an alternative [1,5,14,16] to more classical approaches (ODEs, Petri Nets, etc). Taking into account the particularities of ecosystem dynamics, P systems also suit as the base for their computational modelling. In this regard, the success attained with the models of several phenomena (population dynamics of *Gypaetus barbatus* [3] and *Rupicapra p. pyrenaica* [6] in the Catalan Pyrenees; population density of *Dreissena polymorpha* in Ribarroja reservoir [2]) has led to the development of a P systems based computing framework for the modelling of *Population Dynamics* [2].

One of the assets of this framework is the ability to conduct the simultaneous evolution of a high number of species, as well as the management of a large number of auxiliary objects (that could represent, for instance, grass, biomass or animal bones). Moreover, the compartmentalized structure, both as a directed graph (environments) and as a rooted tree (membranes), allows to differentiate multiple geographical areas. The framework also facilitates the elaboration of models for which a straightforward interpretation of the simulations can be easily obtained.

The development of efficient algorithms capable of capturing the semantics described by the framework is a challenging task. These algorithms should select rules in the models according to their associated probabilities, while keeping the maximal parallelism semantics of P systems. In this scenario, the concept of *rule blocks* arises. A rule block is a set of rules sharing the same left-hand side (more precisely, the necessary and sufficient conditions for them to be applicable are exactly the same). That is, given a particular P system configuration, either all or none of the rules in the block can be applied. On each step of computation one or more blocks are selected, according to the semantics associated with the modelling framework. For every selected block, its rules are applied a number of times in a probabilistic manner according to their associated probabilities, also known as *local probabilities*.

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 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 [7].

The algorithms mentioned above share a common drawback, regarding a distorted selection of blocks and rules. Indeed, instead of blocks and rules being selected according to their probabilities in a uniform manner, the 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*, that overcomes 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 the formal modelling framework. Section 3 describes the DCBA algorithm. The behaviour of DCBA when simulating a real ecosystem model is shown in Section 4. The simulated model has been adapted and improved from the original version. The paper ends with some conclusions and ideas for future work.

2 The P Systems Based Framework

Let us present the formal definition of Population Dynamics P systems, which have been specifically tailored for modelling the evolution of ecosystems. The intuition behind this framework is that the ecosystem being modelled is splitted into small geographical areas (environments) that are connected, and then the dynamics of each environment is simulated by a dedicated P system using cooperative and probabilistic rules.

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

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

where:

- *$G = (V, S)$ is a directed graph. Let $V = \{e_1, \dots, e_m\}$;*
- *Γ is the working alphabet and $\Sigma \subsetneq \Gamma$;*
- *T is a natural number greater or equal to 1;*
- *R_E is a finite set of communication rules of the form*

$$(x)_{e_j} \xrightarrow{p} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$$

  where $x, y_1, \dots, y_h \in \Sigma$, $(e_j, e_{j_l}) \in S$ $(1 \leq l \leq h)$ and p is a computable function from $\{1, \dots, T\}$ to $[0, 1]$. If for any rule p is the constant function 1, then we can omit it. These functions verify the following:
  - *For each $e_j \in V$ and $x \in \Sigma$, the sum of functions associated with the rules whose left-hand side is $(x)_{e_j}$, is the constant function 1.*
- *μ is a membrane structure consisting of q membranes injectively labelled by $1, \dots, q$. The skin membrane is labelled by 1. We also associate electrical charges from the set $EC = \{0, +, -\}$ with membranes.*
- *R is a finite set of evolution rules of the form*

$$u[\,v\,]_i^{\alpha} \rightarrow u'[\,v'\,]_i^{\alpha'}$$

  where $u, v, u', v' \in \Gamma^*$, i $(1 \leq i \leq q)$, $u + v \neq \lambda$ and $\alpha, \alpha' \in \{0, +, -\}$.
  - *If $(x)_{e_j}$ is the left-hand side of a rule from R_E, then none of the rules of R has a left-hand side of the form $u[v]_1^{\alpha}$, for any $u, v \in \Gamma^*$ and $\alpha \in \{0, +, -\}$, having $x \in u$.*
- *For each $r \in R$ and for each j $(1 \leq j \leq m)$, $f_{r,j} : \{1, \dots, T\} \longrightarrow [0, 1]$ is computable. These functions verify the following:*
  - *For each $u, v \in \Gamma^*$, i $(1 \leq i \leq q)$, $\alpha, \alpha' \in \{0, +, -\}$ and j $(1 \leq j \leq m)$ the sum of functions associated with j and the rules whose left-hand side is $u[v]_i^{\alpha}$ and whose right-hand side has polarization α', is the constant function 1.*
- *For each j $(1 \leq j \leq m)$, $\mathcal{M}_{1j}, \dots, \mathcal{M}_{qj}$ are strings over Γ.*

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 such that they form a directed graph G.

Each environment e_j contains a P system, $\Pi_j = (\Gamma, \mu, R_{\Pi_j}, \mathcal{M}_{1j}, \ldots \mathcal{M}_{q,j})$, of degree q, where every rule $r \in R$ has a computable function $f_{r,j}$ (specific for environment j) associated with it. The set of rules $r \in R$ of Π having included the functions $f_{r,j}$ is denoted by R_{Π_j}, for each environment e_j. All environments include an almost identical P system, sharing the same membrane structure and set of rules. The only differences between them reside in the functions associated with the rules, and in the initial multisets. As customary in Membrane Computing, the q strings $\mathcal{M}_{1j}, \ldots, \mathcal{M}_{qj}$ represent the initial multisets associated with the q regions of μ, within the environment e_j.

Communications between environments are allowed, restricted to a subset of the alphabet $\Sigma \subsetneqq \Gamma$. Note that objects from Σ located in an environment cannot participate on any evolution rule.

A *configuration* of the system at any instant t is a tuple of multisets of objects present 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. At the initial configuration of the system we assume that all environments are empty and all membranes have a neutral polarization. The evolution of the system is restricted to T transitions. That is, even if the system could keep evolving, we impose a bound on the number of steps to be simulated.

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

The P system can pass from one configuration to another by using the rules from $\bigcup_{j=1}^{m} R_{\Pi_j} \cup R_E$ 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 – that is, no rule can be further applied.

If an evolution rule of the form $u[\,v\,]_i^{\alpha} \to u'[\,v'\,]_i^{\alpha'}$ is selected to be applied on membrane i of Π_j (for some $1 \leq j \leq m$), then multisets v and u will be deleted from region i and the parent region of i, respectively, and for the next step new multisets v' and u' will be generated in region i and the parent region of i, respectively. Besides, the charge of membrane i in Π_j will be set to α'. Notice that if two rules defined over membrane i have different charges on their right-hand side, then they cannot be selected to be applied in the same step in the same environment.

When a communication rule $(x)_{e_j} \xrightarrow{p} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$ between environments 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 randomly selects one, according to their probability which is given by $p(t)$.

3 Direct Distribution Based on Consistent Blocks Algorithm (DCBA)

In this section we describe the Direct distribution based on Consistent Blocks Algorithm (DCBA), together with some auxiliary definitions and properties necessary for it. The DCBA is introduced in order to solve some distortions generated by the previous algorithm, DNDP, concerning the number of applications for competing rules (with overlapping left-hand sides). The DNDP algorithm assigns randomly the number of applications, by shuffling the list of rules, while the DCBA introduces a mechanism to distribute the number of applications proportionally. Moreover, the management of consistency in application of rules has been improved by introducing the new concept of consistent block. More details can be obtained from the following definitions and the description of the DNDP algorithm [11,12].

3.1 Definitions for Blocks and Mutual Consistency

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

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

(a) *Given a rule $r \in R_E$ of the form $(x)_{e_j} \xrightarrow{p} (y_1)_{e_{j_1}} \cdots (y_h)_{e_{j_h}}$ where $e_j \in V$ and $x, y_1, \ldots, y_h \in \Sigma$:*
- *The left-hand side of r is $LHS(r) = (e_j, x)$.*
- *The right-hand side of r is $RHS(r) = (e_{j_1}, y_1) \cdots (e_{j_h}, y_h)$.*

(b) *Given a rule $r \in R$ of the form $u[v]_i^{\alpha} \to u'[v']_i^{\alpha'}$ where $1 \leq i \leq q$, $\alpha, \alpha' \in \{0, +, -\}$ and $u, v, u', v' \in \Gamma^*$:*
- *The left-hand side of r is $LHS(r) = (i, \alpha, u, v)$.*
- *The right-hand side of r is $RHS(r) = (i, \alpha', u', v')$.*

The charge of $LHS(r)$ is the second component of the tuple (idem for $RHS(r)$).

Definition 3. *For each $e_j \in V$, $x \in \Gamma$, we denote by $B_{e_j,x}$ the block of communication rules having $(x)_{e_j}$ as left-hand side.*

Definition 4. *For each $1 \leq i \leq q$, $\alpha, \alpha' \in EC$, $u, v \in \Gamma^*$, we denote by $B_{i,\alpha,\alpha',u,v}$ the block of evolution rules having $u[v]_i^{\alpha}$ as left-hand side, and having α' in the right-hand side.*

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.

Remark 1. Note that **all** the rules $r \in B_{i,\alpha,\alpha',u,v}$ can be consistently applied, in the sense that each membrane i with charge α goes to the **same** charge α' by any rule of $B_{i,\alpha,\alpha',u,v}$.

Definition 5. *Two blocks $B_{i_1,\alpha_1,\alpha'_1,u_1,v_1}$ and $B_{i_2,\alpha_2,\alpha'_2,u_2,v_2}$ are mutually consistent with each other, if and only if $(i_1 = i_2 \wedge \alpha_1 = \alpha_2) \Rightarrow (\alpha'_1 = \alpha'_2)$.*

Definition 6. *A set of blocks $\mathcal{B} = \{B^1, B^2, \ldots, B^s\}$ is self consistent (or mutually consistent) if and only if $\mathcal{B}$ is a pairwise mutually consistent family.*

Remark 2. In such a context, a set of blocks $\mathcal{B}$ has a relation from $H \times EC$ into EC, associated with it, as follows: $((i,\alpha),\alpha')$ belongs to the relation if and only if there exists two strings $u, v \in \Gamma^*$ such that $B_{i,\alpha,\alpha',u,v} \in \mathcal{B}$. Then, a set of blocks is mutually consistent if and only if the associated relation is functional.

3.2 DCBA Pseudocode

This new simulation algorithm for PDP systems has the same general scheme than its predecessor, DNDP [11,12]. The main loop (Algorithm 1) is divided into two stages: selection and execution of rules, similarly to the DNDP and BBB algorithms.

Algorithm 1. 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 called $C_0$.
1: INITIALIZATION                                          ▷ (Algorithm 2)
2: for $t \leftarrow 1$ to $T$ do
3:     Calculate probability functions $f_{r,j}(t)$ and $p(t)$.
4:     $C'_t \leftarrow C_{t-1}$
5:     SELECTION of rules.
         • PHASE 1: distribution                           ▷ (Algorithm 3)
         • PHASE 2: maximality                             ▷ (Algorithm 4)
         • PHASE 3: probabilities                          ▷ (Algorithm 5)
6:     EXECUTION of rules.                                 ▷ (Algorithm 6)
7:     $C_t \leftarrow C'_t$
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 1. Phase 1 distributes objects to the blocks in a certain proportional way. Phase 2 assures the *maximality* by checking the maximal applications of each block. Finally, Phase 3 passes from block applications to rule applications by computing random numbers following the multinomial distribution with the corresponding *probabilities*. Recall that the DNDP algorithm uses only two micro-stages within phase 1, since it directly select rules without using blocks.

Before starting to select and execute rules in the system, some data initialization is required (Algorithm 2). For instance, the selection stage uses a table in order to distribute the objects among the blocks. This table $\mathcal{T}$,

Algorithm 2. INITIALIZATION

```
1: Construction of the *static distribution* table $\mathcal{T}$:
     – Column labels: consistent blocks $B_{i,\alpha,\alpha',u,v}$ of rules from $R$.
     – Row labels: pairs $(x, i)$, for all objects $x \in \Gamma$, and $0 \leq i \leq q$.
     – For each row, for each cell of the row: place $\frac{1}{k}$ if the object in the row label
       appears in its associated compartment with multiplicity $k$ in the LHS of the
       block of the column label.
2: **for** $j = 1$ **to** $m$ **do**                ▷ (Construct the *expanded static* tables $\mathcal{T}_j$)
3:     $\mathcal{T}_j \leftarrow \mathcal{T}$.                     ▷ (Initialize the table with the original $\mathcal{T}$)
4:     For each rule block $B_{e_j,x}$ from $R_E$, add a column labelled by $B_{e_j,x}$ to $\mathcal{T}_j$;
         place the value 1 at row $(x, 0)$ for that column.
5:     Initialize the multisets $BLOCKS_j \leftarrow \emptyset$ and $RULES_j \leftarrow \emptyset$
6: **end for**
```

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). An expanded static table $\mathcal{T}_j$ is also constructed for each environment, to consider also blocks from environment e_j communication rules. Finally, two multisets $BLOCKS_j$ and $RULES_j$, are initialized for each environment. They are used by the algorithm in order to store the selected blocks and the selected rules in the environment e_j, respectively.

The distribution of objects among the blocks is performed in Selection Phase 1 (Algorithm 3), taking into account overlapping LHS, if any. The expanded static table $\mathcal{T}_j$ is used for this purpose in each environment. Three filters are defined in order to adapt $\mathcal{T}_j$ to the configuration C_t of the system; that is, to select which blocks are going to receive objects. Filter 1 discards the columns of the table corresponding to non-applicable blocks due to mismatch charges (i.e. charges on the LHS of each block are compared with the current charges of the corresponding membranes in C_t). Filter 2 discards the columns corresponding to non-applicable blocks due to the objects from the LHS. The goal of Filter 3 is to save space in the table, by discarding irrelevant rows (associated with objects not present in the configuration). These three filters are applied at the beginning of phase 1, yielding a *dynamic table* $\mathcal{T}'_j$ for each environment j.

Filter Procedures for Selection Phase 1

```
**procedure** Filter 1(table $\mathcal{T}$, configuration $C$)          ▷ (Columns by charges)
    Discard columns from table $\mathcal{T}$, whenever the charge of the membrane in the LHS
    of the corresponding block differs from the configuration $C$.
**end procedure**
```

```
procedure Filter 2(table $\mathcal{T}$, configuration $C$)    ▷ (Columns by multiplicities)
    Discard columns from table $\mathcal{T}$, such that for any row $(o,i)$ or $(x,0)$, the multiplicity of that object in $C$ multiplied by $1/k$ (the value in the table), returns a number $\kappa, 0 \leq \kappa < 1$. If all the values for that column are null, it is also filtered.
end procedure
procedure Filter 3(table $\mathcal{T}$, configuration $C$)    ▷ (Rows by multiplicities)
    Discard rows from $\mathcal{T}$ labelled by $(o,i)$ and $(x,0)$ when the corresponding objects are not present in the multisets of $C$.
end procedure
```

Recall that the *static table* $\mathcal{T}$ collects all consistent blocks within the columns. The set of all consistent blocks is unlikely to be mutually consistent. However, two non-mutually consistent blocks, $B_{i,\alpha,\alpha'_1,u_1,v_1}$ and $B_{i,\alpha,\alpha'_2,u_2,v_2}$ (assigning a different charge to the same membrane), will not cause major troubles provided that they have different LHS (either $u_1 \neq u_2$ or $v_1 \neq v_2$) and that they are not applicable simultaneously. At each step, the non-applicable block will be discarded by Filter 2. This situation is commonly handled by the model designers, in order to avoid losing 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, after applying Filters 1 and 2, the mutual consistency is checked. If it fails, meaning that an inconsistency was encountered, the simulation process is halted, providing a warning message to the user. 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 whether to 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 updating the values in the table by the following products:

- The corresponding multiplicity of the object in the current configuration C'_t.
- 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 normalized value with respect to 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 multiplicities in the LHS of the blocks, the distribution, in fact, penalize the blocks requiring more copies of the same object, which is inspired in the amount of energy required to join individuals from the same species. In fact, this is the major difference with the DNDP algorithm, which performed a non-deterministic distribution.

After the object distribution process, the number of applications for each block is computed by selecting the minimum value in each column. This number is then used for consuming the LHS 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. As this situation may occur

Algorithm 3. SELECTION PHASE 1: DISTRIBUTION

```
 1: for $j = 1$ to $m$ do                                   ▷ (For each environment $e_j$)
 2:     Apply filters to $\mathcal{T}_j$ using $C'_t$, obtaining the dynamic table $\mathcal{T}'_j$, as follows:
        a. $\mathcal{T}'_j \leftarrow \mathcal{T}_j$
        b. FILTER 1 $(\mathcal{T}'_j, C'_t)$.
        c. FILTER 2 $(\mathcal{T}'_j, C'_t)$.
        d. Check *mutual consistency* for the blocks remaining in $\mathcal{T}'_j$. If there is at
           least one inconsistency then report the information about the error, and
           optionally halt the execution (in case of not activating step *3*.)
        e. FILTER 3 $(\mathcal{T}'_j, C'_t)$.
 3:     *(OPTIONAL)* Generate a set $S_j$ of sub-tables from $\mathcal{T}'_j$, formed by sets of
        *mutually consistent* blocks, in a maximal way in $\mathcal{T}'_j$ (by the inclusion
        relationship). Replace $\mathcal{T}'_j$ with a randomly selected table from $S_j$.
 4:     $a \leftarrow 1$
 5:     repeat
 6:         for all rows $X$ in $\mathcal{T}'_j$ do
 7:             $RowSum_X \leftarrow$ total sum of the non-null values in the row $X$.
 8:         end for
 9:         $\mathcal{TV}_j \leftarrow \mathcal{T}'_j$                ▷ (A temporary copy of the dynamic table)
10:         for all non-null positions $(X, Y)$ in $\mathcal{T}'_j$ do
11:             $mult_X \leftarrow$ multiplicity in $C'_t$ at $e_j$ of the object at row $X$.
12:             $\mathcal{TV}_j(X,Y) \leftarrow \lfloor mult_X \cdot \frac{(\mathcal{T}'_j(X,Y))^2}{RowSum_X} \rfloor$
13:         end for
14:         for all not filtered column, labelled by block $B$, in $\mathcal{T}'_j$ do
15:             $N_B \leftarrow \min_{X \in rows(\mathcal{T}'_j)}(\mathcal{TV}_j(X,B))$     ▷ (The minimum of the column)
16:             $BLOCKS_j \leftarrow BLOCKS_j + \{B^{N_B}\}$          ▷ (Accumulate the value)
17:             $C'_t \leftarrow C'_t - LHS(B) \cdot N_B$             ▷ (Delete the LHS of the block.)
18:         end for
19:         FILTER 2 $(\mathcal{T}'_j, C'_t)$
20:         FILTER 3 $(\mathcal{T}'_j, C'_t)$
21:         $a \leftarrow a + 1$
22:     until $(a > A) \vee$ *(all the selected minimums at step 15 are* $0$*)*
23: end for
```

frequently, the distribution and the configuration update process is performed A times, where A is an input parameter referring to *accuracy*. The more the process is repeated, the more accurate is the distribution, but the less could be the performance of the simulation. We have experimentally checked that $A = 2$ gives the best accuracy/performance ratio.

In order to repeat efficiently the loop for A, and also before going to the next phase (maximality), FILTERS 2 and 3 are applied again. This way, once the configuration is updated by consuming the objects on the LHS of the selected blocks, the blocks that are not applicable any more are discarded from the table.

After phase 1, some objects may be left without being consumed. This can be caused by a low A value or by rounding artefacts when calculating sums and minimums of inverse numbers in the distribution process. Due to

Algorithm 4. 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 $\mathcal{T}'_j$.
3:     for all block $B$, following the previous random order do
4:         $N_B \leftarrow$ number of possible applications of $B$ in $C'_t$.
5:         $BLOCKS_j \leftarrow BLOCKS_j + \{B^{N_B}\}$                ▷ (Accumulate the value)
6:         $C'_t \leftarrow C'_t - LHS(B) \cdot N_B$    ▷ (Delete the LHS of block $B$, $N_B$ times.)
7:     end for
8: end for
```

the requirements of P systems semantics, a maximality phase is now applied (Algorithm 4). Following a random order, a maximal number of applications is calculated for each block still applicable. As a consequence, no object that can be consumed is left in the current configuration. In order to minimize the distortion towards the most probable blocks, this phase is performed after phase 1, as a residual number of objects is expected to be consumed in this phase.

Algorithm 5. SELECTION PHASE 3: PROBABILITY

```
1: for $j = 1$ to $m$ do                                          ▷ (For each environment $e_j$)
2:     for all block $B^{N_B} \in BLOCKS_j$ do
3:         Calculate $\{n_1, \ldots, n_l\}$, a random multinomial $M(N_B, g_1, \ldots, g_l)$ with
             respect to the probabilities of the rules $r_1, \ldots, r_l$ within the block.
4:         for $k = 1$ to $l$ do
5:             $RULES_j \leftarrow RULES_j + \{r_k^{n_k}\}$.
6:         end for
7:     end for
8:     Delete the multiset of selected blocks $BLOCKS_j \leftarrow \emptyset$.
9: end for
```

After the application of phases 1 and 2, a maximal multiset of selected (mutually consistent) blocks has been computed. The output of the selection stage has to be, however, a maximal multiset of selected rules. Hence, phase 3 (Algorithm 5) 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 $M(N, g_1, \ldots, g_l)$, where N is the number of applications of the block, and $g_1, \ldots, g_l$ are the probabilities associated with the rules $r_1, \ldots, r_l$ within the block, respectively.

Once the rules to be applied on the current simulation step are selected, the execution stage (Algorithm 6) 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 RHS of the selected rules. Therefore, for each selected rule, the objects present on the RHS are added to the corresponding membranes and the indicated membrane charge is set.

Algorithm 6. EXECUTION

```
1: for $j = 1$ to $m$ do                                   ▷ (For each environment $e_j$)
2:     for all rule $r^n \in RULES_j$ do              ▷ (Apply the RHS of 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 $RULES_j \leftarrow \emptyset$.
7: end for
```

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 from the one provided in [4]. 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 wild ungulates upon which the vulture feeds.

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

$$\Pi = (G, \Gamma, \Sigma, T, R_E, \mu, R, \{f_{r,1} : \ r \in R\}, \mathcal{M}_1, \mathcal{M}_2)$$

where:

- $G = (V, S)$ with $V = \{e_1\}$ and $S = \emptyset$.
- In the alphabet Γ, we represent the seven 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 ($i = 2$)
- Female Red Deer ($i = 3$)
- Male Red Deer ($i = 4$)
- Fallow Deer ($i = 5$)
- Roe Deer ($i = 6$)
- Sheep ($i = 7$)

Note that although the male red deer and female red deer are the same species, we consider them as different species. This is because mortality of male deer is different from the female deer by reason of hunting.

- $\Sigma = \emptyset$.
- 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.
- $R_E = \emptyset$.
- $\mu = [\ [\ ]_2\ ]_1$ 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 $\alpha = \lceil \sum_{j=1}^{k_{1,4}} q_{1,j} \cdot 1.10 \cdot 682 \rceil$

    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 bearded vultures that are j years old, the goal of the 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 figure can reach higher values. Thus, to avoid problems related with the underestimation of this value the first year we have overestimated the population growth at 10%. The constant value 682 represents the amount of food needed per year for a Bearded Vulture pair to survive.
- The set R is defined as follows:
  - 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 year, 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 [4] 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

PLinguaCore is a software library for simulation that accepts an input written in P-Lingua [8] and provides simulations of the defined P systems. For each supported 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 extended the pLinguaCore library to include the DCBA simulation algorithm for PDP systems. The current version of pLinguaCore is 3.0 and can be downloaded from [18].

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 [12] 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, both obtained from [4]. In [19] 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 the year 1994. For each simulation algorithm we have made 100 simulations of 14 years, that is, 42 computational steps. The simulation workflow has been implemented on a Java program that runs over the pLinguaCore library (this Java program can be downloaded from [19]). 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 100 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 [17] to produce the population graphics.

The population graphics for each species and simulation algorithm are represented in Figures 1 to 7.

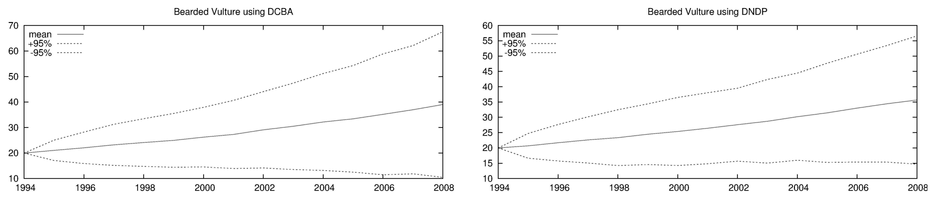

Fig. 1. Evolution of the Bearded Vulture birds

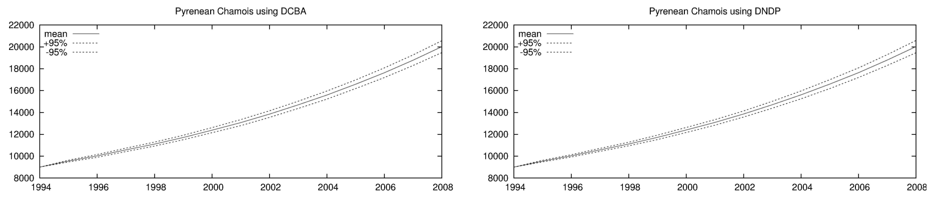

Fig. 2. Evolution of the Pyrenean Chamois

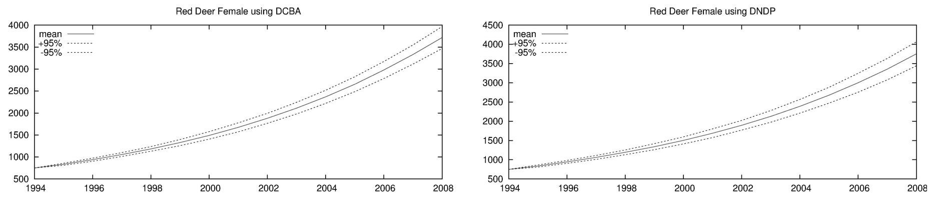

Fig. 3. Evolution of the female Red Deer

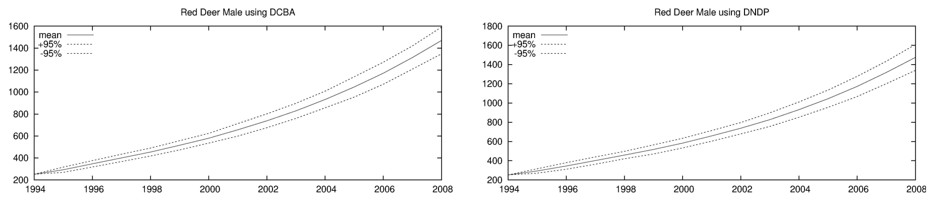

Fig. 4. Evolution of the male Red Deer

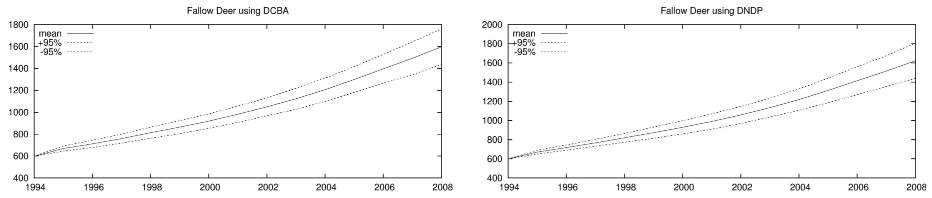

Fig. 5. Evolution of the Fallow Deer

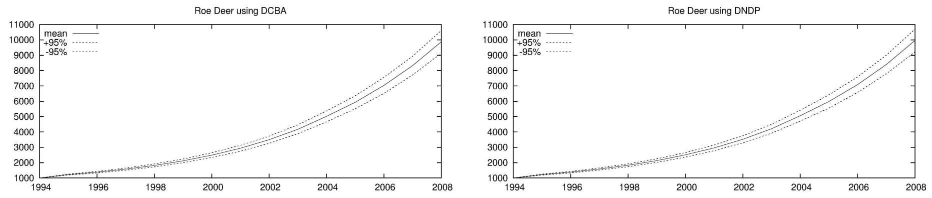

Fig. 6. Evolution of the Roe Deer

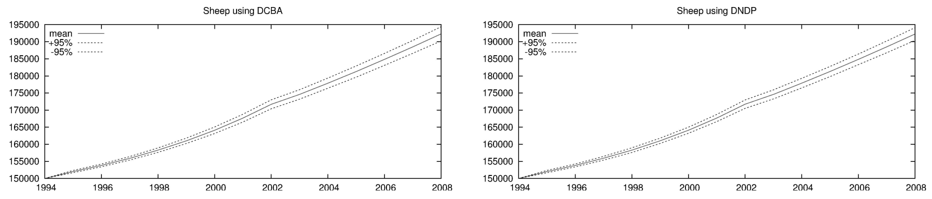

Fig. 7. Evolution of the Sheep

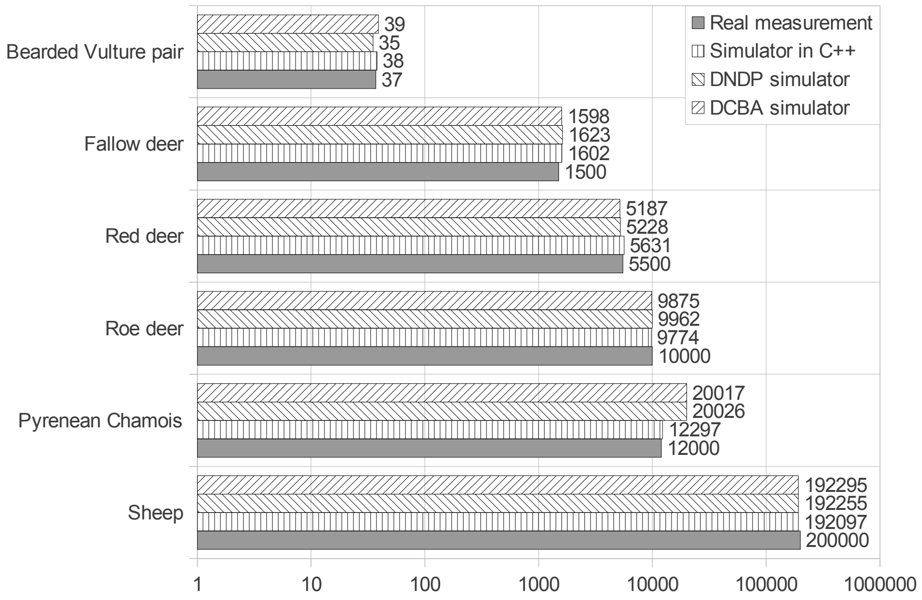

Fig. 8. Data of the year 2008 from: real measurements of the ecosystem, original simulator in C++, simulator using DNDP and simulator using DCBA.

The main difference between the DNDP and the DCBA algorithms is the way they distribute the objects between different rule blocks that compete for the same objects. In the model, the dynamics of the ungulates are modelled by using rule blocks that do not compete for objects. Therefore, similar results are obtained by the simulator for both DCBA and DNDP algorithms. However, in the case of the Bearded Vulture, there is 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 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 among vultures of different ages (rules $r_{22,1,j}$). Since there is a small initial population of bearded vultures (20 pairs), some small differences, motivated by the probabilistic component of the simulators, can be noticed between the results from DCBA, DNDP, C++ simulator and the 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). Although the total number of vultures evolves in a similar way for all simulators, the distribution of bones among vultures of different ages is performed in a more natural way by DCBA, according to the ecologists opinion.

In Figure 8 it is shown the comparison between the actual data for the year 2008 and the simulation results obtained 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 [4]. Taking this into account, one can notice 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 behaviour of actual ecosystems by means of PDP systems.

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. The algorithm is validated towards a real ecosystem model, showing that they reproduce similar results as the original simulator written in C++.

The accelerators in High Performance Computing offer new approaches to accelerate the simulation of P systems and Population Dynamics. An initial parallelization work of the DCBA by using multi-core processors is described in [9]. 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. In this respect, the massively parallel processors of graphics cards (GPUs) have been recently used to achieve higher accelerations [10]. In future work, we will improve those parallel simulators, and reconnect them to the pLinguaCore framework through efficient and robust communication protocols.

Acknowledgements. 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 Economía y Competitividad" of Spain, both co-financed by FEDER funds.

References

1. Bianco, L., Manca, V., Marchetti, L., Petterlini, M.: Psim: a simulator for biomolecular dynamics based on P systems. In: IEEE Congress on Evolutionary Computation, pp. 883–887 (2007)
2. Cardona, M., Colomer, M.A., Margalida, A., Palau, A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Sanuy, D.: A computational modeling for real ecosystems based on P systems. Natural Computing 10(1), 39–53 (2011)
3. Cardona, M., Colomer, M.A., Margalida, A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Sanuy, D.: A P System Based Model of an Ecosystem of Some Scavenger Birds. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 182–195. Springer, Heidelberg (2010)
4. Cardona, M., Colomer, M.A., Pérez-Jiménez, M.J., Sanuy, D., Margalida, A.: Modeling Ecosystems Using P Systems: The Bearded Vulture, a Case Study. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 137–156. Springer, Heidelberg (2009)
5. Cheruku, S., Păun, A., Romero-Campero, F.J., Pérez-Jiménez, M.J., Ibarra, O.H.: Simulating FAS-induced apoptosis by using P systems. Progress in Natural Science 17(4), 424–431 (2007)
6. Colomer, M.A., Lavín, S., Marco, I., Margalida, A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Sanuy, D., Serrano, E., Valencia-Cabrera, L.: Modeling Population Growth of Pyrenean Chamois (Rupicapra p. pyrenaica) by Using P-Systems. In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds.) CMC 2010. LNCS, vol. 6501, pp. 144–159. Springer, Heidelberg (2010)
7. Colomer, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Comparing simulation algorithms for multienvironment probabilistic P system over a standard virtual ecosystem. Natural Computing 11(3), 369–379 (2012)
8. García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A.: An Overview of P-Lingua 2.0. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 264–288. Springer, Heidelberg (2010)
9. Martínez-del-Amor, M.A., Karlin, I., Jensen, R.E., Pérez-Jiménez, M.J., Elster, A.C.: Parallel Simulation of Probabilistic P Systems on Multicore Platforms. In: Proceedings of the Tenth Brainstorming Week on Membrane Computing, vol. II, pp. 17–26 (2012)
10. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Gastalver-Rubio, A., Elster, A.C., Pérez-Jiménez, M.J.: Population Dynamics P Systems on CUDA. In: Gilbert, D., Heiner, M. (eds.) CMSB 2012. LNCS, vol. 7605, pp. 247–266. Springer, Heidelberg (2012)
11. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A., Sancho-Caparrini, F.: A simulation algorithm for multienvironment probabilistic P systems: A formal verification. International Journal of Foundations of Computer Science 22(1), 107–118 (2011)
12. Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A., Colomer, M.A.: A new simulation algorithm for multienvironment probabilistic P systems. In: Proceedings of the 5th IEEE International Conference on Bio-Inspired Computing: Theories and Applications, vol. 1, pp. 59–68 (2010)

13. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000); Turku Center for Computer Science-TUCS Report No 208
14. Păun, G., Romero-Campero, F.J.: Membrane Computing as a Modeling Framework. Cellular Systems Case Studies. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 168–214. Springer, Heidelberg (2008)
15. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing (2010)
16. Terrazas, G., Krasnogor, N., Gheorghe, M., Bernardini, F., Diggle, S., Cámara, M.: An Environment Aware P-System Model of Quorum Sensing. In: Cooper, S.B., Löwe, B., Torenvliet, L. (eds.) CiE 2005. LNCS, vol. 3526, pp. 479–485. Springer, Heidelberg (2005)
17. The GNUplot web page, `http://www.gnuplot.info`
18. The P-Lingua web page, `http://www.p-lingua.org`
19. The Bearded Vulture ecosystem model in P-Lingua, `http://www.p-lingua.org/wiki/index.php/bvBWMC12`