



Random walk simulation by population dynamics P systems

David Orellana-Martín^{1,2} · José A. Andreu-Guzmán^{1,2} · Carmen Graciani^{1,2} · Agustín Riscos-Núñez^{1,2} · Mario J. Pérez-Jiménez^{1,2}

Received: 7 December 2023 / Accepted: 21 March 2024
© The Author(s) 2024

Abstract

PDP systems have been widely used for real-life applications, such as systems biology, ecosystems, physics or economy, among others. Complex systems related with these areas are simulated in the framework of Membrane Computing using objects and membranes that can represent entities or places in the real-life process. In physics, the study of a particle in different fluids, depending on their composition, is really interesting for several applications. A first approximation to this field is to think that particles move randomly in the available space, without any force that constrains their movements. This behavior is known as random walk, and it is used not only in physics but in economics, genetics, and ecology among other areas. In this paper, we introduce generic PDP systems for simulating the behavior of particles, both for one-dimensional spaces and for two-dimensional spaces, using different simulators to analyze the computational resources consumed.

Keywords Membrane computing · PDP systems · Modeling · Random walk

Mathematics Subject Classification 68Q07 · 68Q15

1 Introduction

Membrane Computing is a bio-inspired paradigm based on the structure and behavior of living cells. It was first introduced in Ref. [12], trying to give an alternative perspective to fields such as formal language theory and computability theory. The main devices within this framework are the so-called P systems. Several kinds of these systems have been

defined, some of them are explained in Refs. [13, 14]. Apart from theoretical results, such as computational power and efficiency, a wide range of applications have been found by specific types of P systems. As some of them can be found in Refs. [9, 11, 16], we want to stress the impact of probabilistic-like systems, called population dynamics P systems, or *PDP systems*, in the field of ecosystems. From the first successful implementation of a model for the endangered species *Gypaetus barbatus*, or bearded vultures, in the Pyrenean and Prepyrenean mountains of Catalonia [7], passing through the Pyrenean chamois [5] and the zebra mussel [8] in the fluvial reservoir of Riba-roja, to the Giant Panda conservation in China [16], it has been proved that this framework is plausible for the simulation of real-life processes. In fact, in Refs. [1, 2], two simple models are defined to simulate two classical physics problems such as the Stern–Gerlach experiment and the Uranium 238 decay. Some tools have been developed to simulate and validate these models, such as P-Lingua [17], conceiving a general framework for simulating several different variants of membrane systems, MeCoSim [18], a user-friendly customizable user interface capable of automatizing some tasks for users outside of the community of P systems, as well as some GPU-based simulators in the PMCGPU project [19],

✉ David Orellana-Martín
dorellana@us.es

José A. Andreu-Guzmán
jandreu@us.es

Carmen Graciani
cgdiaz@us.es

Agustín Riscos-Núñez
ariscosn@us.es

Mario J. Pérez-Jiménez
marper@us.es

¹ Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

² SCORE Lab, I3US, Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

simulating both recognizer and probabilistic P systems using GPGPU technologies, more precisely, under the NVIDIA CUDA framework. In this work, we want to prove the usefulness of these kinds of systems in the simulation of the dynamics of particles moving in a free manner. The paper is organized as follows: in the next section, some references of PDP systems are given. In Sects. 3 and 4, two different models for one-dimensional and two-dimensional spaces are introduced. Next, we introduce some tools to graph the behavior of particles in the space and to show a comparison of the time spent in the computations depending on the size of the space and the number of particles. The work will be closed with some conclusions and open research lines.

2 PDP systems

PDP systems are a variant of P systems inspired by the functioning of cells. Cells are able to run multiple processes in parallel in a perfectly synchronized manner, making them good candidates to be imitated for modeling complex problems. A PDP system can be viewed as a cellular tissue in which each cell is within a special compartment called environment. The cells have a particular structure hierarchy in which there is a skin membrane that defines and distinguishes the inside from the outside. In turn, inside a cell, there are a number of hierarchically arranged membranes, where organelles or chemical substances capable of evolving according to specific reactions of the membrane may appear. PDP systems are probabilistic P systems, that is, the applications of their rules are commanded by a predefined probability on them. While having the possibility of multiple environment and both evolution and environment rules, we are going to explain the model of a single PDP system in an environment where no environment rules are allowed. For a more exhaustive explanation of this model, see Ref. [6].

A PDP system with a single P system of degree $q \geq 1$ is a tuple

$$\Pi = (\Gamma, \mu, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}),$$

where:

1. Γ is the working alphabet;
2. μ is a tree-like graph;
3. $\mathcal{M}_1, \dots, \mathcal{M}_q$ are multisets of objects over Γ , that will be placed in the q membranes in the first configuration; and
4. \mathcal{R} is the set of evolution rules of the type:
 - $u[v]_h \xrightarrow{r} u'[v']_h$, where u, v, u', v' are multisets over Γ , $h \in \{1, \dots, q\}$ and r is a probability function.

A simple PDP system of degree q can be seen as a set of q membranes structured in a tree-like way as defined by μ . Each multiset of objects \mathcal{M}_i is placed in the corresponding membrane i in the initial configuration. The system evolves as follows: a rule $r \equiv u[v]_h \rightarrow u'[v']_h$ is applicable if the multiset u is placed in the parent membrane of a membrane labeled by h and the multiset v is placed in such a membrane labeled by h . If the rule is applied, then the multiset u is removed from the parent membrane of such a membrane labeled by h , the multiset v is removed from such a membrane labeled by h , the multiset u' is placed in the parent membrane of a membrane labeled by h , and the multiset v' is placed in such a membrane labeled by h . The rules will be applied in a maximal parallel way; that is, a multiset of rules R is not applicable if it is a subset (in the sense of multisets) of another applicable multiset of rules. The sum of the probabilities of the rules that share the same left-hand side will always be equal to 1. While in this paper, we do not take into account polarizations in the membranes, in the original definition of PDP systems, electrical charges lead to the concept of consistent multisets of rules; that is, rules that charge their left-hand side must have the same polarization at their right-hand side. In this context, while it seems that the model is clearly defined, some different semantics can arise while the following question is asked: what happens when two rules do not exactly share the left-hand side but they share some objects? In this model, this is allowed and, in fact, the probabilities of the rules that do not share the whole left-hand side do not have to sum up to 1. The question then is: how are objects distributed between the different rules?

The key of software implementations of these systems are conflicts. If two or more rules compete for a resource, the algorithm has to take a strategy. The resolution of conflicts depends on the algorithm used to simulate the system. Some algorithms as the *Binomial Block-Based simulation algorithm* (BBB) [3], the *Direct Non-Deterministic distribution with Probabilities algorithm* (DNNDP) [5], and the *Direct distribution based on Consistent Blocks Algorithm* (DCBA) [10] have been developed, each of them treating these conflicts in a different way. Thus, the state of the system at any time step is determined by the state of the system at the previous time step.

Depending on the inference engine, the decision on what happens if a conflict of objects appears differs. We are going to briefly describe the behavior of each of the simulators in this sense. The BBB algorithm does not take into account this type of conflict, as the real-life cases used did not take this behavior into account. The DNNDP algorithm will try to distribute the objects in the following way: first, a multiset of consistent rules will be created, using the number of remaining objects available that fire the rule and the maximum number of applications of each rule as the parameter for a

binomial distribution. Later, the rest of objects are selected to fire rules depending on the probability of the rule (the higher the probability, the sooner the rule will be checked). Finally, the DCBA algorithm first creates a set of consistent blocks that compete for the same resources (i.e., objects). In this case, the selection process biases the rules with less probability. The process is divided in three phases: first, the objects are distributed among the different blocks of rules, keeping the consistency of the applied rules. Second, maximality is pursued by assigning objects to different blocks of rules. Lately, for each block, the objects assigned to each block are distributed among the different rules of the corresponding block using a multinomial distribution function. On the one hand, since the BBB algorithm does not check anything about the consistency of the applied multisets of rules, it is the fastest inference engine. However, if these conflicts appear, the results obtained from its simulations can lead to non-sense conclusions. On the other hand, since it is divided in three computationally intensive stages, the DCBA algorithm is not so efficient and, while the results can be close to real-life experiments, the simulation time can be extremely high and therefore, may not be very useful from a practical point of view. For this purpose, a GPGPU implementation of this algorithm can be found within the PMCGPU project.

3 The one-dimensional model

In this model, particles going in a one-dimensional space are simulated. In this sense, particles have only two movements options: either going to the left or going to the right. Since the PDP system studied has a single environment and it is not used, we are going to define directly the behavior of the P system under study. Let N be the number of particles simulated, and n_0 the space available. The corresponding P system is a tuple

$$\Pi = (\Gamma, \mu, \mathcal{M}_1, \mathcal{R}),$$

where:

1. $\Gamma = \{e_i \mid 0 \leq i \leq N - 1\} \cup \{a_{ij} \mid 0 \leq i \leq N - 1, 0 \leq j \leq n_0 - 1\}$;
2. $\mu = [\quad]_1$;
3. $\mathcal{M}_1 = \{e_i \mid 0 \leq i \leq N - 1\}$;
4. The set \mathcal{R} contains the following rules:

Objects e_i do not directly represent the particles, but they will be generated in the first step. In this case, the particles are generated in the one-dimensional space. We situate all the particles in the center of the space. Object $a_{i,j}$ will represent that particle i will be present in the point j . The first subscript is very useful for identifying the particle

from other particles in the same system. It can be used, for instance, for graphically describing the movement of the particle. Particles are not generated directly in the initial configuration since it will be useful for future research to be able to generate from the initial objects a different number of particles or to put them in a specific point of the space.

$$[e_i \rightarrow a_{i,j}]_1 \text{ for } 0 \leq i \leq N - 1, j = \lfloor n_0/2 \rfloor$$

From the second configuration, we simulate the random movement of the particle. Particles have two possible actions: either they move to the left or they move to the right. Since the extreme points of the space are limits and they are not joint with each other, there are two exceptions to this rule: when a particle is at the leftmost point or at the rightmost point. In these cases, the only option for the particle is to go to the right or to the left, respectively.

$$\left. \begin{array}{l} [a_{i,j}]_1 \xrightarrow{1/2} [a_{i,j+1}]_1 \\ [a_{i,j}]_1 \xrightarrow{1/2} [a_{i,j-1}]_1 \end{array} \right\} \text{for } \begin{cases} 0 \leq i \leq N - 1, \\ 1 \leq j \leq n_0 - 2. \end{cases}$$

$$\left. \begin{array}{l} [a_{i,n_0-1} \rightarrow a_{i,n_0-2}]_1 \\ [a_{i,0} \rightarrow a_{i,1}]_1 \end{array} \right\} \text{for } 0 \leq i \leq N - 1.$$

4 The two-dimensional model

In this model, the behavior of a free movement of particles in a two-dimensional space is simulated. We simulate a two-dimensional space with “walls”; that is, particles cannot pass through the limits of the space. Let N be the number of particles simulated. Let n_0 and n_1 be the space available in the x axis and in the y axis, respectively; that is, if a particle is situated in the (i, j) coordinate, $0 \leq i \leq n_0 - 1$ and $0 \leq j \leq n_1 - 1$. The corresponding P system is a tuple

$$\Pi = (\Gamma, \mu, \mathcal{M}_1, \mathcal{R}),$$

where:

$$\Gamma = \{e_i \mid 0 \leq i \leq N - 1\} \cup \{a_{i,j,k} \mid 0 \leq i \leq N - 1, 0 \leq j \leq n_0 - 1, 0 \leq k \leq n_1 - 1\};$$

1. $\mu = [\quad]_1$;
2. $\mathcal{M}_1 = \{e_i \mid 0 \leq i \leq N - 1\}$;
3. The set \mathcal{R} contains the following rules:

Objects e_i will be transformed into objects $a_{i,j,k}$ in the first computational step. As in the one-dimensional case, we situate all the particles in the center of the space. Object $a_{i,j,k}$ will represent that particle i will be present in the point (j, k) . It is interesting to keep the position as subscripts for having the possibility of creating higher-dimensional spaces with the same structure. Particles are not generated directly in the initial configuration since it will be useful for future research to be able to generate from the initial objects a

different number of particles or to put them in a specific point of the space.

$$[e_i \rightarrow a_{i,j,k}]_1 \text{ for } 0 \leq i \leq N - 1, j = \lfloor n_0/2 \rfloor, k = \lfloor n_1/2 \rfloor$$

From this point, the free movement of the particles will be simulated. In this case, instead of having two different actions, particles can move in two dimensions; that is, they have four possible actions (they are not able to move diagonally). There exist two exceptions to this rule as following: first, when a particle is situated in an edge of the space. In this case, either they move in one of the two directions still being in the edge or they move in perpendicular directions to the edge (three possible actions). Last, when a particle is situated in a corner of the space. In this case, it has the option to go to each of the edges that finish at that corner (two possible options).

$$\left. \begin{array}{l} [a_{i,j,k}]_1 \xrightarrow{1/4} [a_{i,j-1,k}]_1 \\ [a_{i,j,k}]_1 \xrightarrow{1/4} [a_{i,j,k+1}]_1 \\ [a_{i,j,k}]_1 \xrightarrow{1/4} [a_{i,j+1,k}]_1 \\ [a_{i,j,k}]_1 \xrightarrow{1/4} [a_{i,j,k-1}]_1 \end{array} \right\} \text{for } \begin{cases} 0 \leq i \leq N - 1, \\ 1 \leq j \leq n_0 - 2, \\ 1 \leq k \leq n_1 - 2. \end{cases}$$

$$\left. \begin{array}{l} [a_{i,j,0}]_1 \xrightarrow{1/3} [a_{i,j-1,0}]_1 \\ [a_{i,j,0}]_1 \xrightarrow{1/3} [a_{i,j,1}]_1 \\ [a_{i,j,0}]_1 \xrightarrow{1/3} [a_{i,j+1,0}]_1 \end{array} \right\} \text{for } \begin{cases} 0 \leq i \leq N - 1, \\ 1 \leq j \leq n_0 - 2. \end{cases}$$

$$\left. \begin{array}{l} [a_{i,j,n_1-1}]_1 \xrightarrow{1/3} [a_{i,j-1,n_1-1}]_1 \\ [a_{i,j,n_1-1}]_1 \xrightarrow{1/3} [a_{i,j+1,n_1-1}]_1 \\ [a_{i,j,n_1-1}]_1 \xrightarrow{1/3} [a_{i,j,n_1-2}]_1 \end{array} \right\} \text{for } \begin{cases} 0 \leq i \leq N - 1, \\ 1 \leq j \leq n_0 - 2. \end{cases}$$

$$\left. \begin{array}{l} [a_{i,n_0-1,k}]_1 \xrightarrow{1/3} [a_{i,n_0-2,k}]_1 \\ [a_{i,n_0-1,k}]_1 \xrightarrow{1/3} [a_{i,n_0-1,k+1}]_1 \\ [a_{i,n_0-1,k}]_1 \xrightarrow{1/3} [a_{i,n_0-1,k-1}]_1 \end{array} \right\} \text{for } \begin{cases} 0 \leq i \leq N - 1, \\ 1 \leq k \leq n_1 - 2. \end{cases}$$

$$\left. \begin{array}{l} [a_{i,0,k}]_1 \xrightarrow{1/3} [a_{i,0,k+1}]_1 \\ [a_{i,0,k}]_1 \xrightarrow{1/3} [a_{i,1,k}]_1 \\ [a_{i,0,k}]_1 \xrightarrow{1/3} [a_{i,0,k-1}]_1 \end{array} \right\} \text{for } \begin{cases} 0 \leq i \leq N - 1, \\ 1 \leq k \leq n_1 - 2. \end{cases}$$

$$\left. \begin{array}{l} [a_{i,0,n_1-1}]_1 \xrightarrow{1/2} [a_{i,1,n_1-1}]_1 \\ [a_{i,0,n_1-1}]_1 \xrightarrow{1/2} [a_{i,0,n_1-2}]_1 \end{array} \right\} \text{for } 0 \leq i \leq N - 1.$$

$$\left. \begin{array}{l} [a_{i,n_0-1,n_1-1}]_1 \xrightarrow{1/2} [a_{i,n_0-2,n_1-1}]_1 \\ [a_{i,n_0-1,n_1-1}]_1 \xrightarrow{1/2} [a_{i,n_0-1,n_1-2}]_1 \end{array} \right\} \text{for } 0 \leq i \leq N - 1.$$

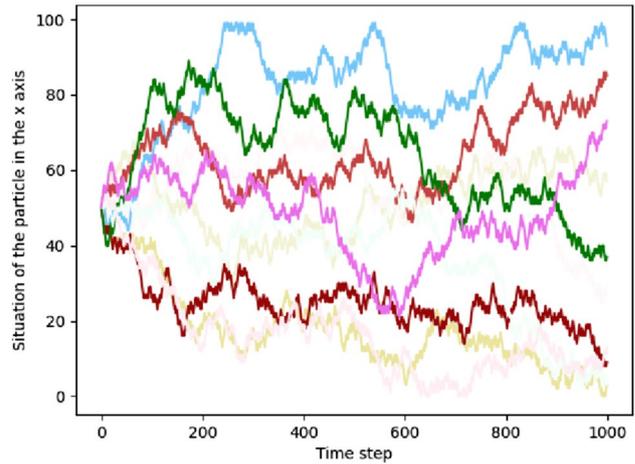


Fig. 1 Evolution of ten particles in a one-dimensional space

$$\left. \begin{array}{l} [a_{i,0,0}]_1 \xrightarrow{1/2} [a_{i,0,1}]_1 \\ [a_{i,0,0}]_1 \xrightarrow{1/2} [a_{i,1,0}]_1 \end{array} \right\} \text{for } 0 \leq i \leq N - 1.$$

$$\left. \begin{array}{l} [a_{i,n_0-1,0}]_1 \xrightarrow{1/2} [a_{i,n_0-2,0}]_1 \\ [a_{i,n_0-1,0}]_1 \xrightarrow{1/2} [a_{i,n_0-1,1}]_1 \end{array} \right\} \text{for } 0 \leq i \leq N - 1.$$

5 Evolution of the systems

The evolution of the systems is easy to follow since each particle i is represented by an object $a_{i,j}$ (respectively, $a_{i,j,k}$) that represents that the particle i is in the point (j) (resp., (j, k)) of the one-dimensional (resp., two dimensional) space. In Fig. 1, we can observe how ten particles move through the space along all the computation. This computation has been carried out using the BBB algorithm. Take into account that the x axis represent the time steps and the y axis represent the point where the particles are placed.

6 Study of the complexity of the systems

We have made simulations for different number of particles and different sizes of space. One of the first interesting points to investigate is the impact of the algorithm used to simulate this program. We are comparing the three inference motors implemented in P-Lingua: BBB, DNDP4¹ and DCBA. In Figs. 2 and 3, we can see that BBB is, as expected, the most efficient algorithm, and that the time spent while using the DCBA algorithm drastically increments as the size

¹ This is the last version of the DNDP algorithm.

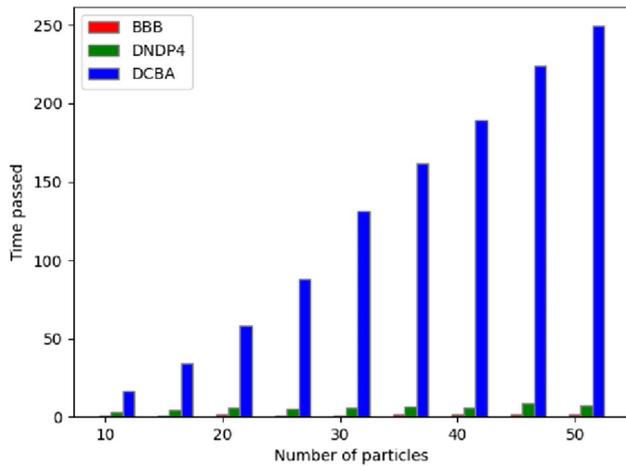


Fig. 2 Time spent in 1000 steps of computation of a one-dimensional space of size (50) depending on the number of particles

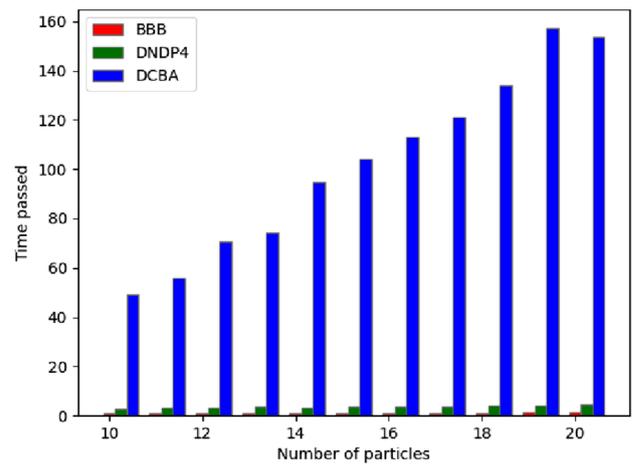


Fig. 4 Time spent in 1000 steps of computation of a two-dimensional space of size (10, 10) depending on the number of particles

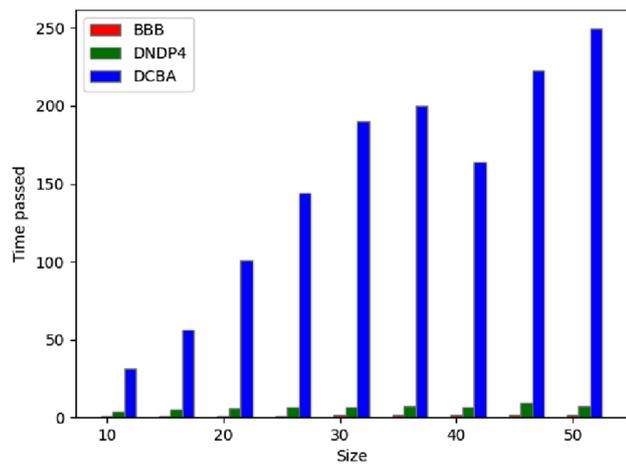


Fig. 3 Time spent in 1000 steps of computation of 50 particles in a one-dimensional space depending on the size

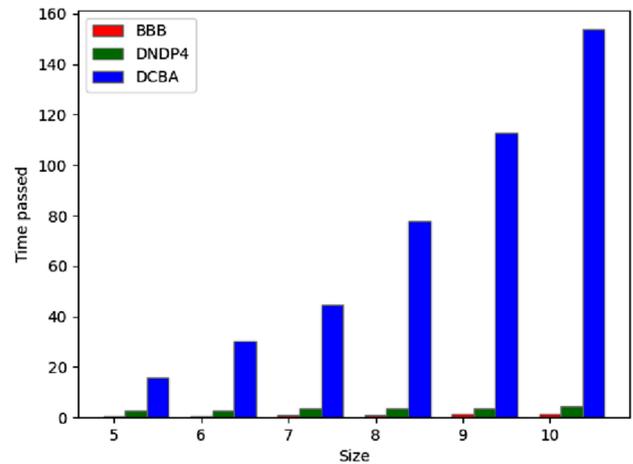


Fig. 5 Time spent in 1000 steps of computation of 20 particles in a two-dimensional space depending on the size

or the number of particles increment. The DNDP4, instead, keeps a similar time independent of the size or the number of particles. While the first two figures are referred to one-dimensional spaces, Figs. 4 and 5 give the corresponding times for the two-dimensional spaces. These times have been calculated using the running time of the simulation of the systems in a Intel Core i5-8250U CPU @ 1.60GHz processor. In the future, we will try to update this comparative with parallel algorithms, such as the ABCD simulator.² This simulator is part of the PMCGPU project [19], and it parallelizes the concepts included in the DNDP algorithm. It would be interesting to see if, for a big lattice or a high

number of particles, the parallelism improves the behavior of the compared algorithms.

7 Conclusion and future work

In the framework of PDP systems, several applications have been found using them as a modeling tool for real-life processes. In this sense, the study of particles moving in a space is interesting for Monte Carlo processes and different types of movements of particles in different fluids. Several research lines are open in this field. On the one hand, new models can arise to simulate the behavior of different types

² <https://sourceforge.net/projects/pmcgpu/files/ABCD-GPU/>

of movement such as Brownian motion. On the other hand, different approaches depending on the simulators are interesting to study.

Virus machines [4] are interesting models of computation inspired by the spread and replication of viruses between hosts. In Ref. [15], authors introduce a variant of virus machines, called stochastic virus machines, where the instructions are connected to hosts instead of channels, and one of the channels going out of the host will be opened depending on probability functions associated to them. A software for this model can be found in Ref. [20]. It would be interesting to see if virus machines are good to simulate the behavior studied in this framework, and how the performance changes with respect to the framework of membrane computing.

Besides, authors have been working lately in a stochastic version of virus machines [4], called stochastic virus machines [15], that use probabilities besides weights in the channels between the hosts.

We are working on new generators to automatically generate different systems depending on the number of dimensions, number of particles and so on.

Acknowledgements The research described in this work is supported by the Zhejiang Lab BioBit Program (Grant No. 2022BCF05). D. Orellana-Martín acknowledges Contratación de Personal Investigador Doctor. (Convocatoria 2019) 43 Contratos Capital Humano Línea 2. Paidi 2020, supported by the European Social Fund and Junta de Andalucía.

Author Contributions D.O.-M. and A.R.-N. wrote the main manuscript text, D.O.M. and M.J.P.-J. described the methodology and validated the work, D.O.-M., J.A.A.-G., and C.G. prepared the experiments and executed them, D.O.-M. and M.J.P.-J. supervised the work and all the authors revised the manuscript text.

Funding Funding for open access publishing: Universidad de Sevilla/CBUA.

Data Availability No datasets were generated or analyzed during the current study.

Declarations

Conflict of interest The authors declare no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Arazo, M., Barroso, M., De la Torre, O., Moreno, L., Ribes, A., Ribes, P., Ventura, A., & Orellana-Martín, D. (2016). Stern-Gerlach Experiment. *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*, February 1 - 5, Sevilla, Spain, 101-112.
2. Arazo, M., Barroso, M., De la Torre, O., Moreno, L., Ribes, A., Ribes, P., Ventura, A., & Orellana-Martín, D. (2016). Uranium-238 decay chain. *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*, February 1 - 5, Sevilla, Spain, 113-130.
3. Cardona, M., Colomer, M. À., Margalida, A., Palau, A., Pérez-Hurtado, I., Pérez-Jiménez, M. J., & Sanuy, D. (2011). A computational modeling for real ecosystems based on P systems. *Natural Computing*, 10(1), 39-53.
4. Chen, X., Pérez-Jiménez, M. J., Valencia-Cabrera, L., Wang, B., & Zeng, X. (2016). Computing with viruses. *Theoretical Computer Science*, 623, 146-159.
5. Colomer, M. À., Lavín, S., Marco, I., Margalida, A., Pérez-Hurtado, I., Pérez-Jiménez, M. J., Sanuy, D., Serrano, E., & Valencia-Cabrera, L. (2010). 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*. Lecture Notes in Computer Science, 6501 (2011), 144-159. A preliminary version in *Proceedings of the Eleventh International Conference on Membrane Computing*, Jena, Germany, 24-27 August Verlag ProBusiness Berlin, 2010, ISBN 978-3-86805-721-8, pp. 121-135.
6. Colomer, M. À., Margalida, A., & Pérez-Jiménez, M. J. (2013). Population dynamics P dystem (PDP) models: a standardized protocol for describing and applying novel bio-inspired computing tools. *PLOS ONE*, 8(4), e60698.
7. Colomer, M. À., Margalida, A., Sanuy, D., & Pérez-Jiménez, M. J. (2011). A bio-inspired computing model as a new tool for modeling ecosystems: the avian scavengers as a case study. *Ecol Model*, 222(1), 33-47.
8. Colomer, M. À., Margalida, A., Valencia, L., & Palau, A. (2014). Application of a computational model for complex fluvial ecosystems: the population dynamics of zebra mussel *Dreissena polymorpha* as a case study. *Ecol Complexity*, 20, 116-126.
9. Frisco, P., Gheorghe, M., & Pérez-Jiménez, M. J. (2014). *Applications of Membrane Computing in Systems and Synthetic Biology*. Complexity and Computation Series: Emergence.
10. Martínez, M. Á., Pérez-Hurtado, I., García-Quismondo, M., Macías-Ramos, L.F., Valencia-Cabrera, L., Romero-Jiménez, Á., Graciani, C., Riscos-Núñez, A., À. Colomer, M., & Pérez-Jiménez, M. J. (2012). DCBA: Simulating population dynamics P systems with proportional objects distribution. *Membrane Computing, 13th International Conference, CMC 2012 Budapest, Hungary, August 28-31, 2012, Revised Selected Papers*. Lecture Notes in Computer Science, 7762 (2013), 257-276. A preliminary version in *Proceedings of the 13th International Conference on Membrane Computing*, Budapest, Hungary, August 28-31, pp. 291-310.
11. Pan, L., Păun, Gh., Pérez-Jiménez, M. J., & Song, T. (2014). Bio-inspired Computing: Theories and Applications. *Communications in Computer and Information Science Series*
12. Păun, Gh. (1998). Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108-143, and *Turku Center for CS-TUCS Report No. 208*

13. Păun, Gh. (2002). *Membrane Computing: An introduction*. Springer Natural Computing Series
14. Păun, Gh., Rozenberg, G., & Salomaa, A. (2009). *The Oxford Handbook of Membrane Computing*. Oxford, U.K.: Oxford University Press.
15. Ramírez-de-Arellano, A., Rodríguez-Gallego, J.A., Orellana-Martín, D., & Ivanov, S. (2023). Stochastic Virus Machines. *Proceedings of the Nineteenth Brainstorming Week on Membrane Computing*, January 24 - 27, Sevilla, Spain, 79-90.
16. Zhang, G., Pérez-Jiménez, M. J., & Gheorghe, M. (2017). *Real-life applications with Membrane Computing*. Complexity and Computation Series: Emergence.
17. P-Lingua website. http://www.p-lingua.org/wiki/index.php/Main_Page
18. MeCoSim website. <http://www.p-lingua.org/mecosim/>
19. PMCGPU project. <https://sourceforge.net/projects/pmcgpu/>
20. Virus machines software repository. <https://github.com/RGNC/virusmachines>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.