Nature-Inspired Computation: two cases

University of Seville

Supervised by:

Miguel Ángel Gutiérrez-Naranjo

Miguel Cárdenas-Montes

Pedro García Victoria

December 7, 2021

Acknowledgements

I would like to thank all those people who have helped me throughout this academic year, either directly or not.

In the first place, the authors of this master's thesis, Miguel Ángel and Miguel. They have been available for any consultation even on their free time. Thanks to them, I have felt like a researcher, and that is something for which I will always be grateful.

To Matteo Cavaliere, another great source of inspiration.

To Nicole, who still has the infinite patient to live with me.

To Marta and Alberto, my best friends, who always have the right advice.

And last but not least, I have to thank my parents infinitely, who have been (and still are) the main promoters of what I am today.

Thanks to all.

Abstract

This work contains the result of two research lines that have ended up being submitted (one of them accepted and the other one is still under revision) to journals. On the one hand, *Optimizing neural networks architectures with PBIL* proposes a methodology to optimize the hyperparameters of the Inception-A block of Inception network. The main contribution is a special codification for the individuals being evolve that allows to skip the creation (hence, the evaluation) of non-valid individuals based on the requirements of this problem. Thus, this methodology is aware of the carbon footprint produced by many AI applications. In order to train and validate models, MNIST dataset is used. Results show this methodology can generate high-quality hyperparameters without explicitly search the complete space defined by the hyperparameters.

On the other hand, *Evolutionary Game Theory in a Cell: A Membrane Computing Approach* proposes a general way to encode Evolutionary Game Theory into Membrane Computing and a novel computational framework which can be used to study, analyze and simulate the spreading of behaviours in structured populations organized in communicating compartments. In order to test the framework, two classic EGT games are used: Prisoner's dilemma and Snowdrift game. The proposed approach allows to simulate populations organized in different compartments, allowing the study of the dynamics of populations that interact with each other. Results shows the spreading of behaviours in three cases: Prisoner's dilemma, Snowdrift game and both games encoded in different membranes. As expected, results show that different behaviours (cooperators and defectors) can co-exists in the Snowdrift game, while in the Prisoner's dilemma, the population is mostly composed by defectors.

Motivation

This work was to be titled *Optimizing neural networks architectures with PBIL*. Nevertheless, thanks to the opportunities that have been presented to me this year, this master's thesis includes the work carried out in two research projects in which I have been involved. On the one hand, the original purpose of this master's thesis: *Optimizing neural networks architectures with PBIL*. On the other hand, the research in which Evolutionary Game Theory and Membrane Computing are combined to propose a family of P Systems to simulate population dynamics. The work have been titled: *Evolutionary Game Theory in a Cell: A Membrane Computing Approach*.

When I started my master's studies, I decided to choose as my thesis the topic entitled *Optimizing neural networks architectures with PBIL*. Mainly, I made the decision for be a topic inspired by nature. Since I had my first contact with genetic algorithms, nature-inspired computing has always been of my interest.

In December 2020, we started working on the selected topic. After several months of hard work, we were able to complete the job with satisfactory results, so we decided to submit it for publication in the Logic Journal of the IGPL. The paper has been reviewed and accepted for publication.

After a few weeks, Miguel Ángel and Miguel, together with Matteo Cavaliere, suggested that I work with them again on a new topic. The idea was to somehow combine Evolutionary Game Theory and Membrane Computing. After several meetings and brainstormings, we managed to shape ideas into something that could be interesting for the community. Thanks to the efforts made, we were able to implement those ideas and decided to send this work for publication to the special issue about Membrane Computing of the journal Information Sciences. At the moment, this work is being reviewed, but we trust it will end up being accepted.

Why *Nature-Inspired Computation: two cases* as title? Both articles have (at least) one thing in common: they use nature-inspired methods. PBIL algorithm belong to the so-called evolutionary algorithms (like Genetic Algorithms or Evolution Strategies) which are based on the id

Finally, this master's thesis collects the content of the two articles proposed to present: *PBIL for optimizing Inception Module in Convolutional Neural Networks* and *Evolutionary Game Theory in a Cell: A Membrane Computing Approach*. In addition, methods used throughout the works are extended with relevant information about the problems.

Without a doubt, for me it has been one of the best experiences I have had, I have learned a lot, both from the topics covered and the way in which investigation is done.

Contents

1 Introduction

Since the beginning of time, humans have needed nature to survive. In order to feed themselves, humans first had to learn to gather aliments and hunt animals, later, to till the land. In order to get water to places where the rain was not enough, had to learn to channel it in a proper way. For thousands of years, human knowledge has been extracted directly from nature. For this reason, the human being is not just another natural being, the intellectual capacity together with the knowledge obtained thanks to its relationship with nature has allowed it to evolve in a more or less good way.

Humans of all ages have been inspired by nature to carry out their work and research. Today it is easy to see the mark they have left just by looking around us. Some of the modern transports, such as the train or the plane, are clearly inspired by animals. In addition, buildings structures and materials are in part deriving from nature. In computer science, there are tons of algorithms or methodologies inspired by nature. Maybe, the area of meta-heuristic algorithms could be the most influenced: Genetic Algorithms, Particle Swarn Optimization, Cuckoo Search, and so on.

In this work, the main methods used are clearly inspired by nature. On the one hand, Population-based Incremental Learning (PBIL) is inspired by the evolutionary process present in nature and proposed by Darwin. PBIL is an optimization method which combines genetic algorithms with competitive learning [1] [2]. Instead of evolve individuals like in Genetic Algorithm (GA) or similar evolutionary algorithms in which a population is generated by specific operators (e.g. reproduction, crossover, etc) over the individuals, the probability distribution of information appearance in genes is made evolve. On the other hand, Membrane Computing uses as a source of inspiration the plasma membrane structure present in cells. Membrane Computing is the result of the effort by many scientists for surpassing the barrier of the electronic computers. Bio-inspired computational models have proven to be an important alternatives that allow solving NP-complete problems in a linear time.

The *PBIL for optimizing Inception Module in Convolutional Neural Networks* paper proposes an attempt to optimize the architecture of the Inception module (presented in Inception neural networks). PBIL algorithm is used as the metaheuristic to optimize such architecture and MNIST dataset as benchmark to validate the resulting models. The main contribution is a special codification applied to the individuals in order to not process some deep learning architectures that are not considered in this work. For instance, another contribution is the awareness of the carbon footprint trying to save CPU cycles.

The paper *Evolutionary Game Theory in a Cell: A Membrane Computing Approach* proposes a general way to encode Evolutionary Game Theory into Membrane Computing. The main contribution is a novel computational framework which can be used to study, analyze and simulate the spreading of behaviours in structured populations organized in communicating compartments. To develop the proposed

model and to simulate it, P-Lingua programming language and MeCoSim are used.

This work is organized as follows: Section 2 develops the *PBIL for optimizing Inception Module in Convolutional Neural Networks* paper. In Section 3, the paper *Evolutionary Game Theory in a Cell: A Membrane Computing Approach* is depicted. In the last Section 4 the final thoughts are commented and future work proposed.

2 Optimizing neural networks architectures with PBIL

2.1 Introduction

Computer vision is one of the areas with a larger portfolio of successful applications in Deep Learning. Part of this success stems from the use of relatively simple convolutional structures which are in turn repeated with small variations, until very deep architectures are built. One of the most used structures is the Inception module [3].

From the first implementation, Inception module has been altered by new proposals, which maintain the essential of the module: several parallel branches of convolutional blocks, including stack of blocks, with different kernel sizes and a `maxpooling` or `average pooling` layers (see [4] for a review). In all the cases the choice of the kernel sizes and the number of kernels are inherited from the reference publications and in few times alternative sizes are evaluated. In part, this is restricted for the computational cost of the hyperparameters optimization process. Inception network will be introduced with details in Section 2.2.2.

In this work, an implementation of Population-based Incremental Learning (PBIL) [1, 2] is used to choose the hyperparameters of the Inception-v4 module. The final objective is to evaluate the suitability of the kernel and filter configurations in this module, and alternatively to propose other high-quality configurations. The classification of the MNIST dataset [5] of handwritten digit images is used as a benchmark for this purpose. The choice of this classification benchmark comes from its wide diffusion in the Deep Learning community. Thus, the use of MNIST as benchmark allows to explore suitable configurations found by the evolutionary algorithm used to optimize the Inception module, moreover, the use of a well-known dataset as MNIST will help to an easy dissemination among the community proposals.

PBIL is a population-based evolutionary algorithm, in which the probability of presence of certain information in genes (e.g. in a binary sequence the probability of appearance of a '1' in genes) is evolved. In the current problem, this information includes the three kernel sizes (among branches in the Inception block), the filter sizes, the max-pooling size in the Inception block, as well as the number of consecutive Inception blocks (see Section 2.2.5 and Fig. 4 for further details).

Hyperparameters under optimization by PBIL evolutionary process are binary encoded. In order to avoid Hamming cliffs[1], Gray coding is used to encode the

[1]Hamming cliff is a problem presented in some evolutionary algorithms in which adjacent dec-

hyperparameters to be optimized (see Section 2.2.5). With Gray coding, adjacent numbers differ only in one, in terms of Hamming distance. Thus, Gray coding aims at removing barriers in the hyperspace searched by the evolutionary process. Some of this concepts will be detailed in the next sections.

Green artificial intelligence pledges for reducing the carbon footprint of their algorithms [6]. The scientific literature alerts that the training of deep architectures involves the emissions of the equivalent five times the lifetime of an average car, including its manufacturing [7]. Our proposal is aware of AI carbon footprint, and for this reason, policies for minimizing the unwanted evaluations of individuals are implemented.

Inside the Inception rationale, the branches are configured with different kernel sizes. The different sizes of the receptive fields in Convolutional Neural Networks (CNN) aim to capture information at different scales: the larger the receptive field, the more generic are the features extracted from the images; whereas in the opposite sense, the smaller the kernel sizes, the more local are the features extracted.

For this reason, it is reasonable to avoid the evaluation of individuals with duplicated kernel sizes. For avoiding the evaluation of these individuals with repeated kernel sizes in different branches, policies have been implemented in association with the Gray coding (see Section 2.2.5).

This proposal is inspired by previous efforts for improving the performance of the forecasting of the ^{222}Rn time series at Canfranc Underground Laboratory (LSC) and air quality. In the past, diverse machine learning algorithms have been used for this purpose, including Multilayer Perceptron, Convolutional Neural Networks, and Recurrent Neural Networks [8, 9, 10, 11, 12].

In [9], an implementation using STL decomposition and CNN for improving the forecasting capacity is presented with promising results, but penalized by the large number of hyperparameters to select based on the practitioners expertise. In order to select the most suitable hyperparameters set, an optimization process based on PBIL was proposed in [13]. Due to the positive results achieved, in the current work we proposed to adapt the methodology to the optimization of the Inception module.

This section is organized as follows: Section 2.2 gives a brief description of the techniques used in this work. In Section 2.3 the results are shown and analysed. Finally, Section 2.4 contains the conclusions of this work.

2.2 Methods

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are Neural Networks with special emphasis in image processing [14] [15], although they are also used in time series analysis and forecasting [8, 9, 16, 17].

imal numbers have a large Hamming distance. For example, 7 and 8 have binary representations 0111 and 1000, respectively, and have a Hamming distance of 4

The CNN consist of a sequence of convolutional layers, the output of which is connected only to local regions in the input. These layers alternate convolutional, non-linear and pooling-based layers which allow extracting the relevant features of the class of objects, independently of their placement in the data sample. The CNN allows the model to learn filters that are able to recognize specific patterns, and therefore they can capture richer information.

One of the main characteristics of convolutional layers is local connectivity. In a multilayer perceptron, each input is connected with every neuron of the first layer. When processing images, every pixel is connected to every neuron, so each neuron is getting information of the whole image. This is impractical and highly expensive computationally. Instead, convolutional layers connect each neuron to local regions of the input. The spatial size (width and height) of this local regions is called kernel size and extents along the depth axis which is equal to the depth of the input. In order to control the number of parameters, convolutional layers use parameter sharing. Parameter sharing constraint the number of parameters in every local connectivity of the same depth, so the number of parameters is shared by each spatial position in every 2d-dimensional depth.

In classification problems, the last layer of a CNN is usually one or more fully-connected layers that end up in a dense layer with softmax function as the activation of neurons. In this way, the output of the network is the probability distribution over the classes.

2.2.2 Inception Network

Inception network is a deep neural network architecture with a focus on efficient rather than only accuracy. This makes the architecture suitable to use it in real world problems and devices with limited resources. Nevertheless, Inception network has been the state of the art for many mainstream problems, like classification and detection problems in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14) [3]. Inception was proposed by Christian Szegedy et al. in [3] with a main objective: to maintain constant a previously defined computational budget.

Before Inception, the state of the art for problems like classification, object detection or human pose estimation, usually stacked convolutional, pooling and fully connected layers. In larger datasets such as Imagenet, the trend has been to increase the number of layers and its size, resulting in an increased accuracy but also adding computational cost. To address the problem of overfitting dropout layers or batch normalization layers are mandatory. Although stacking layers allow to obtain a very high precision, the number of parameters is too high and the training becomes inefficient.

The Inception architecture is organized in modules or blocks. Each module is formed by convolutional and pooling layers, organized in parallel. In that way, the output of every layer is represented as a tensor that will be the input for the next module (or layer).

The first version of Inception module could capture the optimal sparse structure, but it was computationally expensive. It was formed by convolutions of 5x5, 3x3 and 1x1, resulting in a high number of parameters. Guided by this problem, the authors added 1x1 convolutions before the 3x3 and 5x5 convolutions and after pooling layers. The 1x1 convolution allows to reduce the dimension of the input in the filter dimension. This was based on the success of embeddings: even low dimensional embeddings might contain a lo of information [3]. One of the main beneficial aspects of the Inception architecture is that it allows for increasing the number of layers at each stage without an uncontrolled blow-up in computational complexity [3].

In the last version, Inception-v4, the 5x5 convolutional layer is replaced by two 3x3 convolutional layers concatenated. The reason behind this change is that convolutions with larger spatial filters are much more computationally expensive. The two 3x3 concatenated convolutions can be seen as a mini-network that can capture relevant information — similar to 5x5 convolution — but a lower number of parameters [18]. Another change incorporated in Inception-v4 was a different stem module. The stem module refers to the initial operations performed over the input data before the Inception blocks. Usually, stem module is customize depending on the problem being solve, but, in the original Inception-v4 implementation use in [4], the stem module is composed by a concatenation of some convolutional and max pooling layers (see Figure 1). In this work, stem module is only one 3x3 convolution with 32 filters (will be depicted in the next sections).

There are four types of Inception modules. Inception-A block has four branches: average pooling layers followed by 1×1 convolution, 1×1 convolution, 1×1 convolution followed by 3×3 convolution and 1×1 convolution followed by two 3×3 convolutional layers. This is the module used throughout the work and it will be optimized using PBIL algorithm. Inception-B block uses asymmetrical configurations of the kernels sizes, such as: 1×7 followed by 7×1. Inception-C block also uses asymmetrical configurations, but with kernel size 3×3. In the original Inception network, this modules are combined and formed the whole network. In [4] authors proposed a good configuration regarding the number of filters.

2.2.3 Population-based Incremental Learning

Population-based incremental learning (PBIL) is an optimization method which combines genetic algorithms with competitive learning [1] [2]. It belongs to the so-called estimation of distribution algorithms (EDAs). Instead of making evolve individuals like in Genetic Algorithm (GA), the probability distribution of information appearance in the genes is made evolve. The specific operators of PBIL operate over these probability distributions.

In order to represent the probability distribution, PBIL algorithm uses a probability vector. In the case of a binary encoded individual, the probability vector specifies the probability of appearance of a '1' in each bit position. The probability

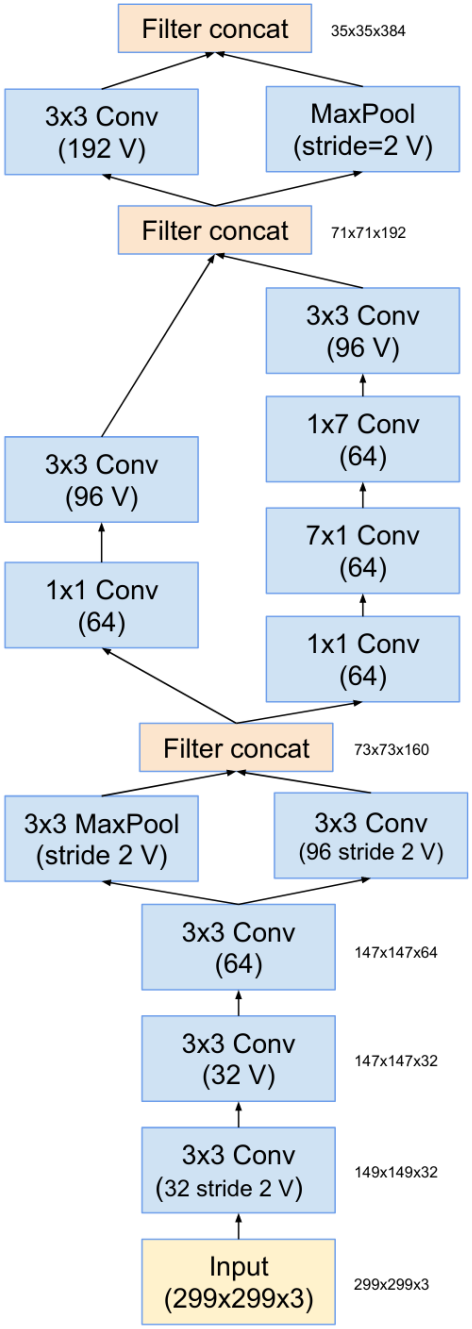

Figure 1: Stem module of the original implementation of Inception-v4 network. Figure extracted from [4].

of appearance of a '0' is just 1.0 minus the probability of a '1'. Instead of managing a population of individuals and transforming them every generation to obtain a new population as in GA, the probability vector defines the population from which new individuals can be drawn. Hence, the operations of PBIL are not defined on the population, but over the probability vector. As the population represented by the

probability vector is not unique, the diversity in search is maintained in successive generations.

Population #1	**Population #2**	**Population #3**
0011	1010	1010
1100	1100	0101
1100	1100	1010
0011	1100	0101
Representation	**Representation**	**Representation**
[0.5, 0.5, 0.5, 0.5]	[1.0, 0.75, 0.25, 0.0]	[0.5, 0.5, 0.5, 0.5]

Figure 2: The probability representation of 3 small populations of 4 individuals. Notice that the first and third representation for the population are the same, although the populations each represents are entirely different [1].

The optimization process performed by PBIL has four crucial steps:

1. **Generate population:** M individuals are generated according to probabilities in the probability vector. The probability vector is initialized with 0.5 values, so every individual is randomly chosen. This is similar to the first step in GAs, where a population of individuals is randomly initialized, ensuring the exploration of the solutions space.

2. **Evaluate individuals:** Every generated individual is evaluated using a fitness function defined according to the problem requirements. The best individual (or top N individuals) are selected based on their fitness score.

3. **Update probability vector:** The probability vector is updated using the best individual (or the top N individuals). The update rule is shown in equation 1. The top N individuals is a parameter of the algorithm. Some implementations also uses the N worst individuals to update the probability vector, receding from them.

4. **Mutate probability vector:** if mutation is applied, the probability vector is shifted by an amount in a randomly chosen direction. This amount is one of the parameters of the algorithm.

These steps are repeated until stopping criterion is satisfied. Usually, the stopping criterion is defined as the number of iterations or generations. But others, like a minimum fitness score, could be used.

The probability update rule is similar to the weight update rule in a competitive learning network when an output is moved towards a particular sample point [1]. The probability update rule is defined as follows:

$$probability_i = (probability_i \times (1.0 - lr)) + (lr \times individual_i) \quad (1)$$

where $probability_i$ is the probability of generating a '1' in position i, lr is the learning rate and $individual_i$ is the ith position in the solution vector. As it can be observed, when there is a '1' in the ith position of the solution vector, the probability is moved towards 1.0 and when there is a '1' in the ith position, the probability is moved away from 1.0. One of the problem of GA is the premature convergence and it is also presented in PBIL. Nevertheless, the learning rate parameter helps to avoid the premature convergence controlling how fast the probabilities are moved towards 0.0 or 1.0 (similar to the learning rate in neural networks).

Algorithm 1 Population-based Incremental Learning

```
p ← initialize probability vector with 0.5 values
while Generations ≠ 0 do
    individuals ← generate individuals according to probabilities in p
    evaluations ← evaluate individuals using the fitness function
    best ← find individual with maximum evaluation
    for i ← 0 to length(p) do
        p_i ← p_i × (1.0 − lr) + best_i * lr                ▷ update probabilities
    end for
    for i ← 0 to length(p) do
        if apply mutation then
            p_i ← p_i × (1.0 − ms) + random(0.0 or 1.0) * ms
        end if
    end for
end while
```

In order to perform extensive search, PBIL implements mutation. Mutation can be performed on the individuals drawn from the probability vector, in a similar way as GAs does. Another method is to perform mutation directly in the probability vector, shifting the probability of every position by an small amount. Mutation is only applied sometimes, using a parameter to decide it.

In our proposal, the individuals are represented by binary sequences of a fixed length and they are randomly initialized with a probability of 0.5. The binary sequence is formed by the concatenation of the binary codification of every hyperparameter under optimization. After their evaluation, the most suitable individuals are selected for updating the probability distribution, which represents the information under optimization. Then, based on the update probability distribution, a new generation of individuals is created, and again evaluated. The cycle is repeated and after some generations, the population converges to a set of high-quality individuals.

The fitness function is defined as the Sparse Categorical Crossentropy of a single execution over the validation set of the CNN defined with the hyperparameters codified by the individual. The evaluation of the individual, and hence of the CNN, is

performed with a single epoch. Although this seems to be prone to underfit the deep architecture, later it is shown as an appropiate strategy to save computational time, at the same time that it does not critically penalize the performance of the network. In this work, PBIL is configured as follows: population size of 10 individuals evolving during 20 generations, and the mutation probability is 0.05. When mutation is applied, the probability vector is shifted by an amount of 0.1 in a randomly chosen direction. The three best and the three worst individuals are selected for updating the probability distribution. The updated probability distribution approaches the configuration of the best individuals, at the same time that it recedes from the worse ones. Thus, the individuals of the next generation inherit more likely high-performance configurations. The choice of the population size and the number of generations stems from the computational intensity of the problems. Larger values of these parameters made the evolutionary strategy unfeasible with the computational resources available (Google Colab Free). The mutation probability and the shifted amount have been established taking into account two information sources, on the one hand the previous work with PBIL in [13], and on the other hand a restricted greedy search around the previous best parameters. Finally, the choice of the three worst and best individuals for updating the probability distribution comes from the previous choice of the population size with only 10 individuals.

2.2.4 Gray Coding

Gray coding is a type of binary coding proposed by the physicist Frank Gray. It is usually used in order to avoid Hamming cliffs. A Hamming cliff is formed when two numerically adjacent values have bit representations that differ in more than one by using the Hamming distance. For example, numbers 3 and 4 have a Hamming distance of 3 in binary representation: 0011 and 0100.

Decimal	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

Figure 3: Gray coding for the first eight numbers.

Evolutionary algorithms that deal with discrete spaces can struggle if two consecutive individuals have bit representations that are far apart by using the Hamming distance, degrading the performance of the algorithm. The change of one unit —

involving diverse bits— in a parameter under optimization requires a large amount of simultaneous modifications of the binary coding, but not in Gray coding.

In order to avoid Hamming cliffs, the Inception-module parameters are Gray-coded before the optimization process.

2.2.5 Gray-Code of Hyperparameters of Inception Module

In Fig. 4 the schema of the Inception-v4 module is depicted. In this schema, the elements that are handled by the PBIL algorithm, and therefore could be altered, are pointed with red rectangles. As it can be observed, in all layers the number of filters can be modified by PBIL.

In our approach, most of the kernel sizes are optimized by PBIL although some kernels with size 1×1 are kept frozen. Behind this decision is the own nature of 1×1 convolutional operation. It allows to reduce the computational intensity by shrinking the number of channels of the tensor of data, at the same time that capture image features at a very local scale.

Some constraints are implemented in the evolutionary algorithm. For instance, in the right-hand branch the kernels sizes of the two convolutional layers have the same configuration. Besides, except for the left-hand branch for which only the average pooling size is evolving, for the other three branches the kernel sizes are forced to be different. Thus, configurations with equal kernels sizes in any of these three branches are excluded. This aims at capturing features at different scales. Asymmetrical configurations of the kernels sizes, such as: 1×7 used in some Inception modules (see [3]) are not considered in this work (though are part of future work).

An additional hyperparameter controls the number of Inception-A blocks that are stacked in the final architecture. This hyperparameter is also handled by PBIL, and it can take values from 1 to 3.

As it has been mentioned, the three right convolutional branches of the Inception module are forced to have different kernels sizes. They are identified by a Gray-coded 3-tuple. In order to reduce the carbon footprint of the invalid tuples (i.e. tuples with repeated kernels sizes), a codification that avoids its generation is implemented. This could be done by penalizing their fitness score after their evaluation, but the computational cost would be higher and those configurations would still be processed.

The range of the possible kernels sizes is restricted to {3,5,7,9,11}. By avoiding the repetitions the search space is drastically reduced. Since 5 different kernels sizes are allowed, if repetitions are allowed, $5^3 = 125$ 3-tuples should be considered. If no repeated sizes are allowed, only $5 \times 4 \times 3 = 60$ 3-tuples are considered.

The key point for avoiding Hamming cliffs by Gray coding is to order the binary sequences in such a way that two consecutive sequences are at Hamming distance one. In this work, the same idea is considered for ordering the 3-tuples of the kernel sizes of the Inception blocks. The idea is the following: *If C_1 and C_2 are two*

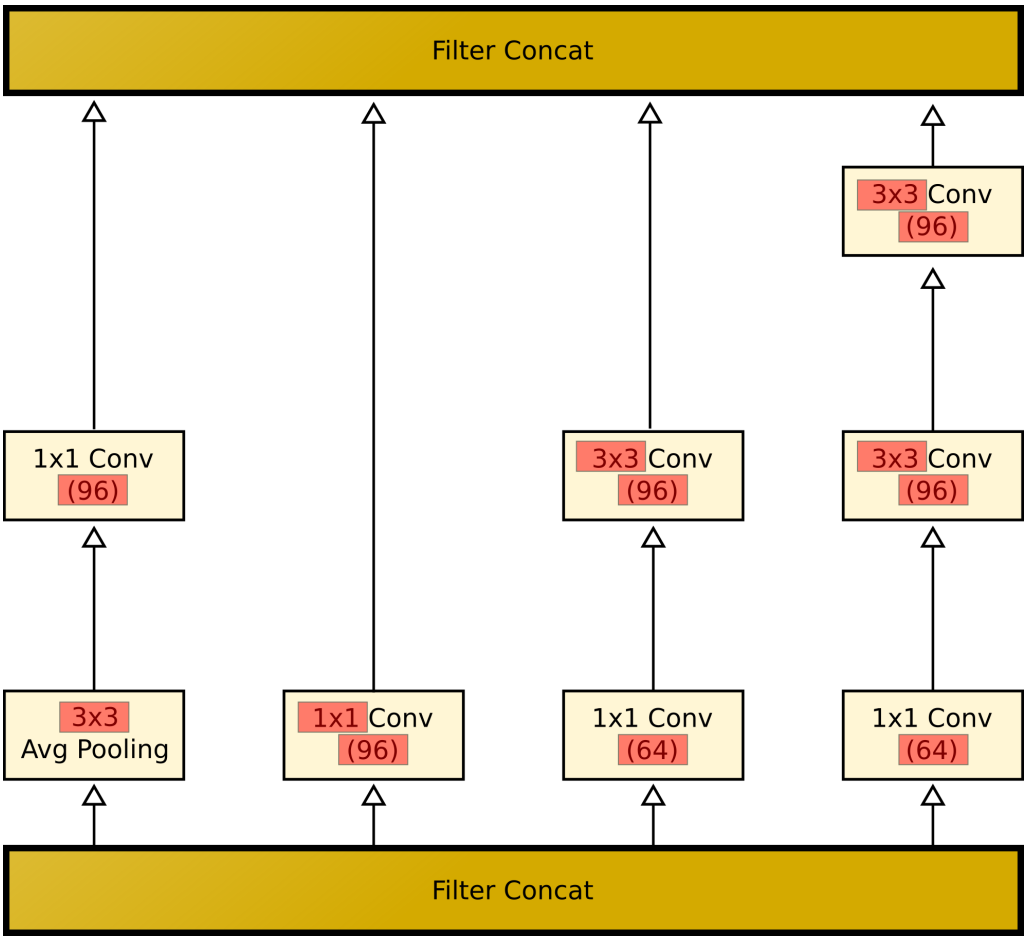

Figure 4: Schema of Inception-v4 module. Red rectangles indicate the elements of the configuration of this module that are being manipulated by the PBIL algorithm. This schema can be replicated up to 3 times for conforming the final architecture.

consecutive Gray codings, then their associated 3-tuple of sizes T_1 and T_2 are also at Hamming distance 1.

In order to reach this target, an abstract general graph G is constructed. The nodes of the graph are the 60 possible 3-tuples of sizes and there is an edge between two nodes if the corresponding 3-tuples are at Hamming distance 1 (see Fig. 5). Any possible Hamiltonian path in this graph provides an ordering of the 60 3-tuples satisfying that two consecutive tuples are at Hamming distance 1. In particular, the ordering shown in Table 2 is one of the possible Hamiltonian paths and satisfies such condition.

Finally, both sequential orderings, the six-bits Gray codings and the Hamiltonian path of 3-tuples satisfy that any the pairs of consecutive elements are at Hamming distance one. In order to compute the fitness value during the evolutionary process, each six-bit Gray coding has associated a 3-tuple and hence a neural network with a particular Inception-A configuration. In this work, 64 six-bits encodings and 60 3-tuples are considered. The approach is to map the three first six-bits encodings to

the first tuple of the Hamiltonian path and to map the three last six-bits encoding to the last 3-tuple (i.e., we consider that '000000', '000001' and '000011' are encodings of the tuple '(11,5,7)' and '100011', '100001' and '100000' are encodings of the tuple '(5,3,9)' (see Tables 1 and 2, and Fig. 5). The remaining 58 six-bit codings are bijectively mapped to the 58 3-tuple in natural order.

Table 1: Since 60 tuples must be encoded in binary form, at least six bits are necessary to reach $2^6 = 64$ encodings. This Table shows such 64 encodings ordered according to the Gray algorithm to avoid Hamming cliffs. Let us note that for each n, the n-th and the $n + 1$-th codings are at Hamming distance 1.

[000000, 000001, 000011, 000010, 000110, 000111, 000101, 000100, 001100, 001101, 001111, 001110, 001010, 001011, 001001, 001000, 011000, 011001, 011011, 011010, 011110, 011111, 011101, 011100, 010100, 010101, 010111, 010110, 010010, 010011, 010001, 010000, 110000, 110001, 110011, 110010, 110110, 110111, 110101, 110100, 111100, 111101, 111111, 111110, 111010, 111011, 111001, 111000, 101000, 101001, 101011, 101010, 101110, 101111, 101101, 101100, 100100, 100101, 100111, 100110, 100010, 100011, 100001, 100000]

Once decided the codification of the kernel sizes, the remaining hyperparameters must also be encoded. Each individual encodes all the hyperparameters needed to describe a neural network with the Inception-A module. Beyond the kernel sizes, the remaining hyperparameters to be encoded are the following (bX represents the branch X in the Inception-A module scheme —from left to right— and lY stands for the layer Y in the corresponding branch —from bottom to top— (see Figure 4).

- Branch 1: `b1_l1_pool_size`, with allowed values from 2 to 5 and `b1_l2_filters`, with allowed values the pairs from 32 to 256.
- Branch 2: `b2_l1_filters`, with allowed values the pairs from 32 to 256 and `b2_l1_kernel` with allowed values the odds from 3 to 11.
- Branch 3: `b3_l1_filters`, with allowed values the pairs from 32 to 256, `b3_l2_filters`, with allowed values the pairs from 32 to 256 and `b3_l2_kernel` with allowed values the odds from 3 to 11.
- Branch 4: `b4_l1_filters`, `b4_l2_filters`, and `b4_l3_filters`, with allowed values the pairs from 32 to 256; and `b4_l2_kernel` and `b4_l3_kernel` with the same value from odds from 3 to 11.
- Number of inception modules concatenated: `num_inception_modules`, with allowed values `[1,2,3]`.

Table 2: Ordering of the 60 3-tuples of kernel sizes obtained as a Hamiltonian path. Let us remark that each pair of consecutive 3-tuples are at Hamming distance 1.

[(11, 5, 7),	(11, 9, 7),	(11, 9, 5),	(11, 9, 3),	(11, 7, 3),	(11, 7, 9),	(11, 7, 5),
(11, 3, 5),	(11, 3, 9),	(11, 5, 9),	(11, 5, 3),	(9, 5, 3),	(9, 11, 3),	(9, 11, 7),
(9, 11, 5),	(9, 7, 5),	(9, 7, 11),	(9, 7, 3),	(5, 7, 3),	(5, 11, 3),	(7, 11, 3),
(7, 11, 9),	(7, 11, 5),	(7, 9, 5),	(7, 9, 11),	(7, 9, 3),	(7, 5, 3),	(7, 5, 11),
(9, 5, 11),	(9, 5, 7),	(9, 3, 7),	(11, 3, 7),	(5, 3, 7),	(5, 11, 7),	(5, 11, 9),
(5, 7, 9),	(5, 7, 11),	(5, 9, 11),	(5, 9, 3),	(5, 9, 7),	(3, 9, 7),	(3, 11, 7),
(3, 11, 9),	(3, 11, 5),	(3, 9, 5),	(3, 7, 5),	(3, 7, 9),	(3, 7, 11),	(3, 9, 11),
(3, 5, 11),	(3, 5, 7),	(3, 5, 9),	(7, 5, 9),	(7, 3, 9),	(7, 3, 11),	(7, 3, 5),
(9, 3, 5),	(9, 3, 11),	(5, 3, 11),	(5, 3, 9)]			

As pointed above, each individual of the population consists on the concatenation of the Gray coding of these hyperparameters. For example, if the following hyperparameters are chosen:

`b1_l1_pool_size` : 3,	`b1_l2_filters` : 116
`b2_l1_filters` : 118,	`b2_l1_kernel` : 7
`b3_l1_filters` : 228,	`b3_l2_filters` : 210
`b3_l2_kernel` : 5,	`b4_l1_filters` : 120
`b4_l2_filters` : 160,	`b4_l2_kernel` : 3
`b4_l3_filters` : 184,	`b4_l3_kernel` : 3
`num_inception_modules` : 3	

then, the concatenation of the Gray codings of the sequence [3, 116, 118, 228, 210, 120, 160, 184, 37, 3] is the corresponding individual, namely

101001110100110110010110101110111000100111100001110010011011110

It should be noted that kernel sizes are not present in the sequence in an explicit way. They are encoded as the index of the 3-tuple $(7, 5, 3)$ in the Hamiltonian path considered. Particularly, the former individual is the best individual produced by the evolutionary process (see Tabular).

In order to decode the individuals, they are divided into hyperparameters and each of them is decoded by using the Gray decoding algorithm (see Figure 7).

2.2.6 Statistics

In order to ascertain if the proposed forecasting methods applied to the test set improve the prediction, two different types of tests can be applied: parametric and non-parametric. The difference between both relies on the assumption that data

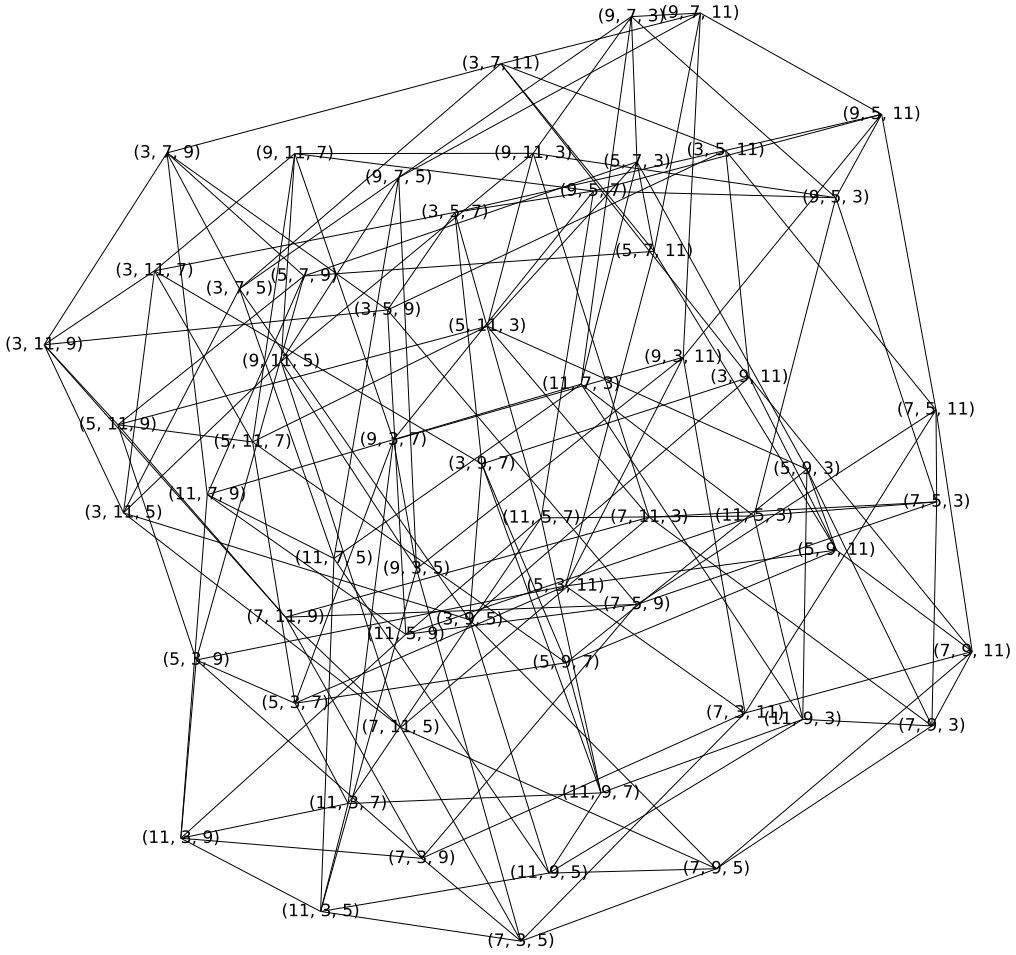

Figure 5: Graph with 60 nodes, where the nodes are labelled with the 3-tuples of kernel sizes (without repeated sizes). There is an edge between two nodes if and only if the corresponding 3-tuples are at Hamming distance 1. Any Hamiltonian path in this graph provides a sequence of the 60 3-tuples where two consecutive ones are at Hamming distance 1 (see Table 2).

are normally distributed for parametric tests, whereas non explicit conditions are assumed in non-parametric tests. For this reason, the latter is recommended when the statistical model of data is unknown [19] [20]. The Kruskal-Wallis test is a non-parametric test used to compare three or more groups of sample data. For this test, the null hypothesis assumes that the samples are from identical populations. The procedure when using multiple comparison to test whether the null hypothesis is rejected implies the use of a post-hoc test to determine which sample makes the difference. The most typical post-hoc test is the Wilcoxon signed-rank test. The Wilcoxon signed-rank test belongs to the non-parametric category. For this test, the null hypothesis assumes that the samples are from identical populations, whereas alternative hypothesis states that the samples come from different populations. It is

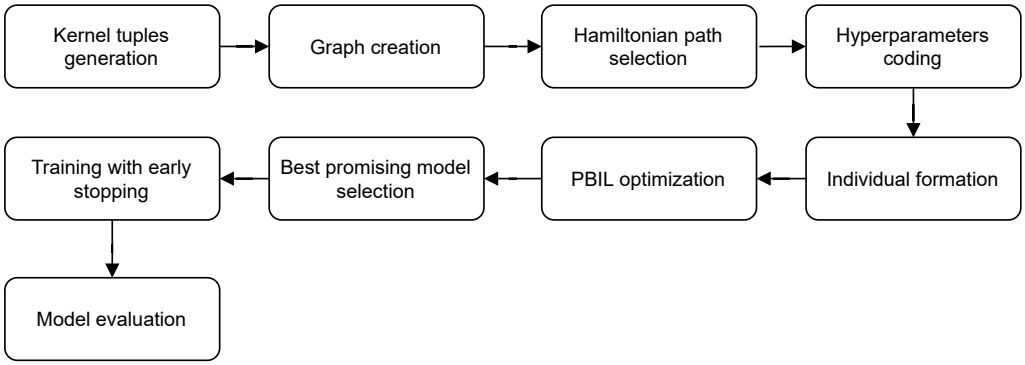

Figure 6: Steps performed by the proposed approach to reach a high quality model over the MNIST dataset.

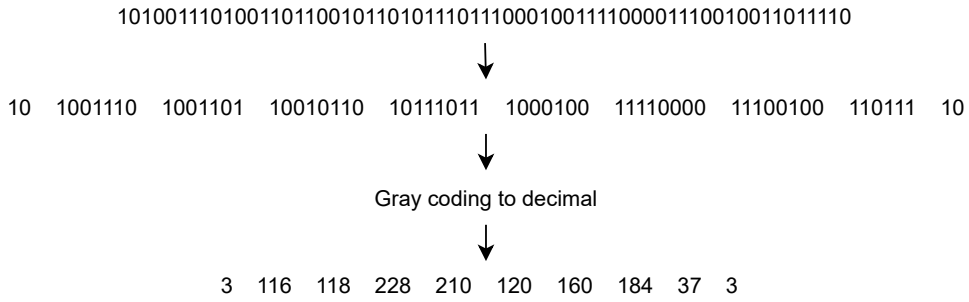

Figure 7: Decodification of an individual. The process to decode an individual perform Gray coding to decimal decodification. Particularly, this individual is the best individual produced by the evolutionary process in this work.

a pairwise test that aims to detect significant differences between two sample means. If necessary, the Bonferroni correction can be applied to control the Family-Wise Error Rate (FWER). FWER is the cumulative error when more than one pairwise comparison (e.g. more than one Wilcoxon signed-rank test) is performed.

2.3 Results

2.3.1 Evolving Inception

In this work, several hyperparameters of the Inception-A module are optimized using PBIL algorithm. As mentioned above, each combination of hyperparameters indetifies a particular neural network using this module. The first layer is a simple 2D convolutional layer that plays the role of the stem module of the Inception network. The stem module refers to the first operations performed before the Inception-A blocks. After the stem module, 1 to 3 Inception-A blocks are concatenated based on the hyperparameters represented by an individual. The output of the last block is flattened and connected to a *dropout-dense-dropout-dense* group of layers. The output is a dense layer with *softmax* as the activation function. In Figure 10 the neural

network architecture proposed is detailed. The hyperparameters of the proposed network (not including Inception-A block ones) are the same for every individual evaluated (see Section 2.2.5). In this way, can compare the effect of the hyperparameters of Inception-A block on the network performance.

Due to computational intensity, during the execution of the PBIL algorithm, a reduced data set — composed of 10^4 examples — are used as the training set. The validation and test sets are composed of 10^4 examples for the evolutionary strategy. The same validation and test sets are also used during the production (i.e. when the final individual is trained and evaluated on the complete training set). This helps to avoid data leaks, namely that examples could be in the training set in the evolutionary process and in the test or the validations sets in the production. During the production, the best individual (the most promising model) is trained using the complete training set, containing a total of $5 \cdot 10^4$ examples.

During the evolutionary process performed by PBIL algorithm, every model (i.e. an individual) is evaluated in the validation set which is a subset of the training set. Test set is separated and reserved before the process in order to avoid data leaks. As stated before, the validation loss is used as fitness value to measure the quality of an individual, whereas the accuracy of the test set — not seen before by the evolutionary process — is used as the final quality criterion (Fig. 8). Each box-plot presents the accuracy of all individuals through generations. Each individual of the PBIL algorithm is a configuration for the Inception-A module, and the accuracy of the test set is used as the final quality criterion. As it can be appreciated, the first generations contain already good individuals, while along the generations these good individuals are concentrated and new ones created. When the evolutionary process is finished, the best individual is trained with early stopping and applying a learning rate scheduler in the complete training set. The validation loss is used as a monitor metric to perform early stopping with patience 5. Results presented in Table 4 are the mean accuracy and the standard deviation on the test set of 20 independent training sessions. As it can be appreciated, the best hyperparameters obtained by PBIL produce the highest accuracy, similar to those produced by original hyperparameters of Inception-A block with three concatenated blocks. This demonstrates that hyperparameters presented in [4] are already a high quality configuration for the Inception-A module. More detail about the best configuration obtained is shown in tabular at the end of Section 2.2.5. It should be underlined that the best configuration is fully compatible with the usual Inception-A configuration, differing only in a larger number of filters (see Table 3).

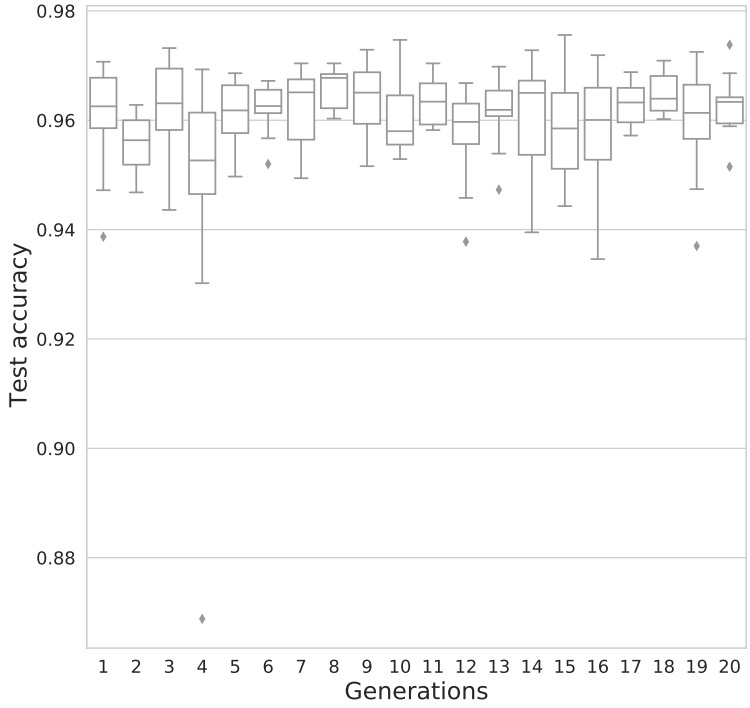

Figure 8: Evolution of the Accuracy in test set per generation.

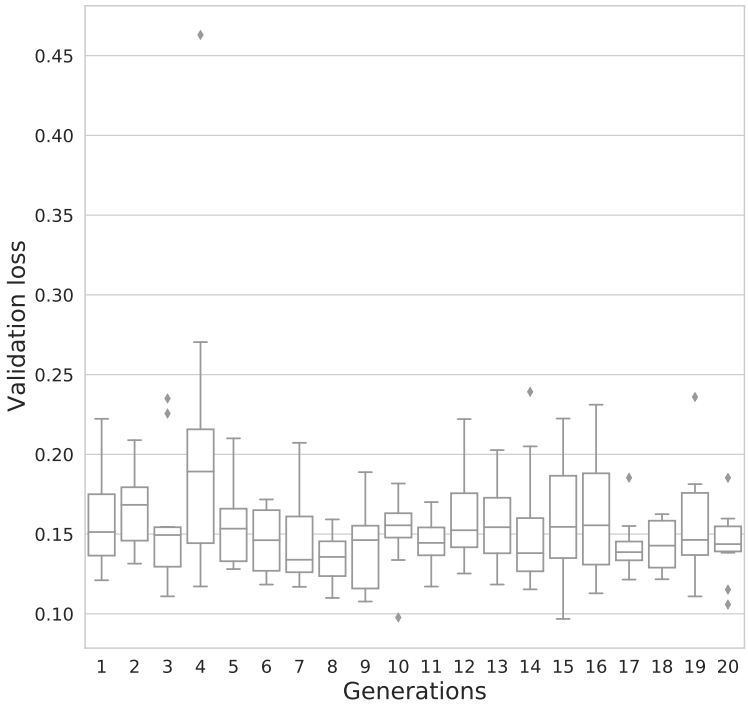

Figure 9: Evolution of the Sparse Categorical Crossentropy loss in validation set per generation.

Table 3: Comparative between original hyperparameters of Inception-A module proposed in [4] and best hyperpameters optimized by PBIL algorithm.

Hyperparameter	**Original implementation**	**Optimized by PBIL**
b1_l1_pool_size	3	3
b1_l2_filters	96	116
b2_l1_filters	96	118
b2_l1_kernel	1	7
b3_l1_filters	64	228
b3_l2_filters	96	210
b3_l2_kernel	3	5
b4_l1_filters	64	120
b4_l2_filters	96	160
b4_l2_kernel	3	3
b4_l3_filters	96	184
b4_l3_kernel	3	3
num_inception_modules	-	3

2.3.2 Results Comparison and Statistical Tests

In Table 4 the mean and standard deviation of accuracy for 20 independent evaluations on MNIST test set is shown. The results include 20 runs of the best hyperparameters set trained with early stopping; and the results of the original hyperparameters of Inception-A block (presented in [4]) with a number of blocks ranging from 1 to 3.

The application of the Kruskal-Wallis test to the accuracy shown in Table 4 indicates that the differences between the medians of the best result obtained in the independent runs are significant for a confidence level of 95% ($p - value$ under 0.05) is used in this analysis. This means that the differences are unlikely to have occurred by chance with a probability of 95%.

The statistical analysis using the Wilcoxon signed-rank test with Bonferroni correction of the results obtained with the 20 independent runs (Table 4) indicates that the differences between the best hyperparameter set obtained by PBIL (trained with early stopping) and the original hyperparameters with 1 or 2 blocks are significant for a confidence level of 95% ($p-value$ under 0.05). The corresponding p-values are $p-value = 0.0001$ for 1-block configuration, and $p-value = 0.003$ for 2-blocks configuration. This means that the differences are unlikely to have occurred by chance with a probability of 95%. Otherwise, the comparison with original hyperparameters and 3-blocks configuration ($p - value = 0.35$) indicates that the differences are not significant for a confidence level of 95% (p-value under 0.05).

This last comparison demonstrates that our approach with a single epoch in the evaluation of the population of PBIL does critically not penalize the performance of the CNN, at the same time that diminish the computational intensity of the evolutionary algorithm.

The mean accuracy achieved, 0.9922 ± 0.0008, with this architecture is among the best ones reported at https://paperswithcode.com/sota/image-classification-on-mnist for deep architectures in the order of 10^5 trainable parameters. Two comparisons are emphasized with the implementations reported in this repository.

- On the one hand, the one with the highest quality implementation reported on the web. This implementation is a CapsNet one, with more than 1.5 millions of trainable parameters and a reported accuracy of 0.9987.
- On the other hand, the one with highest quality implementation with a number of trainable parameters similar to our implementation, in the order of 10^5 trainable parameters [21]. This is also a CapsNet implementation with a reported accuracy of 0.9984.
- In comparison to our implementation, the cited implementations are trained with more than two orders of magnitudes of epochs. In our implementation the early stopping does not progress beyond 10 epochs, with the best accuracy around 5 epochs. This underlines the low carbon footprint of our work, at the same time that achieves a competitive accuracy.

Table 4: Mean and standard deviation of accuracy for 20 independent runs on MNIST test set. The values correspond to the results of the best individuals of PBIL separated by the configuration of the number of blocks, and the execution of the overall best hyperparameters set raised from PBIL.

Hyperparameters	**Accuracy**
Best Inception-A 1-block individual	0.9899 ± 0.0014
Best Inception-A 2-blocks individual	0.9913 ± 0.0009
Best Inception-A 3-blocks individual	0.9920 ± 0.0011
Best hyperparameters set with early stopping	0.9922 ± 0.0008

2.4 Conclusions

In this work, a first attempt to optimize the architecture of the Inception module has been proposed. For this purpose, Population-based Incremental Learning as optimizer and MNIST as benchmark are used. The relevance of this task stems from the wide use of the Inception module in computer vision, where it holds a large number of success cases.

Regarding the contributions of this proposal, it is aware of a low carbon footprint in the artificial intelligence area. Unacceptable deep learning architectures are no longer evaluated thanks to the codification of these architectures. This allows saving CPU cycles, and therefore, reducing their carbon footprint. Of course, this allows to obtain high quality networks in a reasonable time.

The analyses of results demonstrate that the optimized architecture of Inception-A module using only 3 blocks, and therefore a low number of trainable parameters, achieves an excellent performance. This performance supports the use of evolutionary strategies for optimizing deep architectures while reducing the carbon footprint through the use of a single epoch during the parameters optimization. Furthermore, the best configuration is fully compatible with the usual configuration of Inception-A module.

As part of the Future Work, more elements of the Inception architecture are intended to be handled by the evolutionary algorithm. This will allow to propose novel architectures for this module. Use the evaluation after more than one epoch as fitness value could be interesting, although the optimizer used should also be considered. The use of simpler architectures with a lower number of parameters and the same (or better) performance is a good thing to try. This will reduce the execution time of the optimization process and the carbon footprint. Although the impact should be studied, the use of another metaheuristics like Integer-Particle Swarm Optimization could be of interest. Other benchmarks, such as Fashion MNIST, CIFAR-100 or The Street View House Numbers (SVHN) are suggested for in-depth evaluation of the best hyperparameters set, and for the evolutionary algorithm proposal as a deep architecture optimizer.

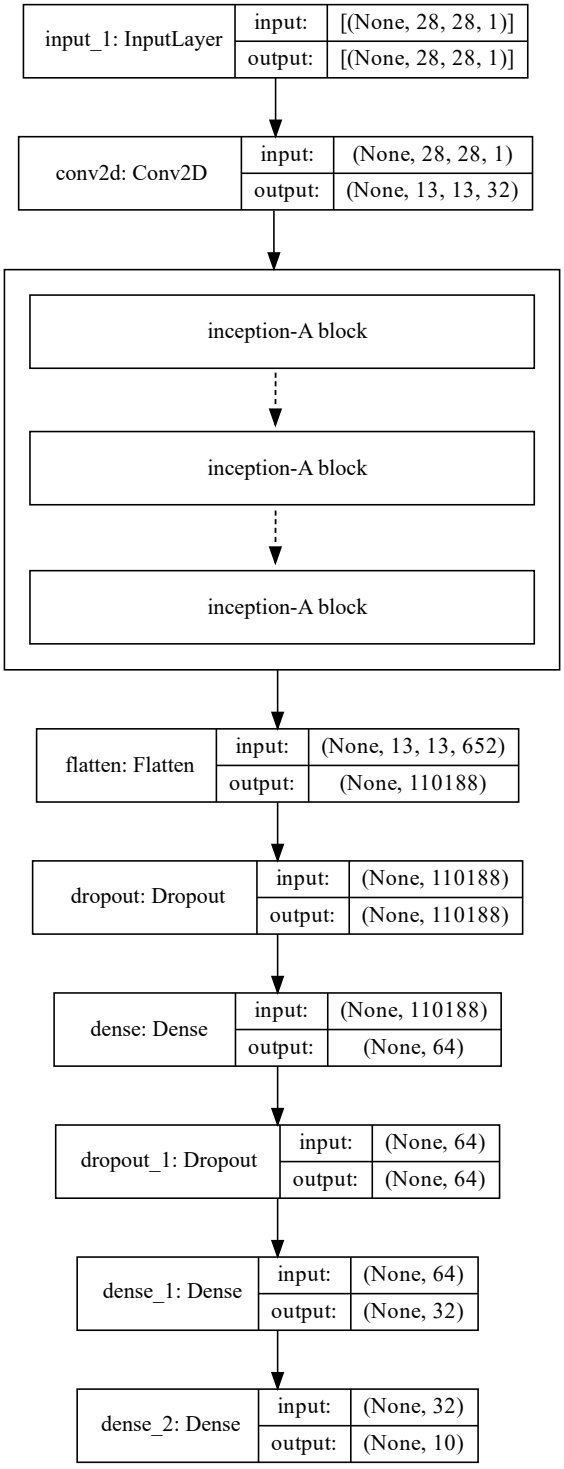

Figure 10: Model architecture. Stripped lines indicate that the number of blocks varies from 1 to 3.

3 Evolutionary Game Theory and Membrane Computing

3.1 Introduction

Evolutionary Game Theory (EGT, for short) is a mathematical and computational framework which is used to study the spreading of behaviours (strategies) in evolving populations [22, 23]. While classical game theory is used to describe the behaviour of completely rational players, in EGT, the individual strategies are not associated to rational choices, but they are assumed to be encoded into inherited programs that can be passed to the offsprings [23].

The main driving principle of EGT is that individuals that perform better (they get a higher payoff/fitness) will tend to replicate more often, so their encoded strategy will spread in the population [23]. The success (fitness) of an individual depends not only on its own adopted strategy but also on the strategies of the other individuals with whom is interacting in the population [23]. This means that there could not be an universal optimal strategy but the optimal choice may depend on the strategies that are adopted by the other components of the population (this makes EGT different from standard optimization and is technical referred as frequency-dependent selection in the area of evolutionary dynamics [23]).

An important problem studied using EGT is the resilience of cooperation and the conflict between cooperative and non-cooperative (cheating/defecting) individuals [24]. This issue is present in many systems, at different scales ranging from technological systems [25] to microbial systems [26] and human societies [27] and it is considered one of the most relevant problems in science [28]. The study of the resilience of cooperation in structured populations has received a strong attention [29, 30, 31, 32, 33, 34], suggesting that the population structure (e.g., the social network) is a crucial part of the problem [31, 33]. Therefore the search of appropriate, efficient and general mathematical and computational frameworks to study the interplay of population structure and spreading of certain strategies has become an important research area [29, 30, 35, 36].

The dynamics of an EGT model can be studied analytically [22] but very often is simulated using ad-hoc agent-based computational models [36]. In these models, the replication and death of individuals (agents) are explicitly simulated using a system updated by a series of discrete events [36].

In the paper *Evolutionary Game Theory in a Cell: A Membrane Computing Approach*, we propose a novel computational approach to study the spreading of behaviours in structured populations by combining EGT and membrane computing. Using the proposed approach, we show that there is an effective and general way to encode EGT into probabilistic membrane computing and P-Lingua [37, 38]. This work therefore enhances the area of membrane computing, extending the line of research focused on the simulation and study of population/ecological dynamics [38, 39] allowing the study of the spreading of strategies in populations organized

in compartments using probabilistic simulators such as P-Lingua [38]. At the same time, the proposed combination enriches the EGT area with a novel cellular-inspired framework to study, analyze and simulate the spreading of behaviours in structured evolving populations organized in communicating nested compartments. Our proposal will make possible to tackle further systems and, for instance, those related to the cooperation concerns.

To sum up, the main contributions of this work are the following:

- A novel approach to study the spreading of behaviours in structured populations by combining Evolutionary Game Theory and Membrane Computing is proposed.
- We show that there is a general way to encode Evolutionary Game Theory into Membrane Computing, leading to a computational framework which can be used to study, analyze and simulate the spreading of behaviours in structured populations organized in communicating compartments.
- The proposed approach allows to extend the works on membrane systems, population and ecological dynamics, and, at the same time, suggests a novel bio-inspired framework, based on formal languages theory, to investigate the dynamics of evolving structured populations.

The next sections are organized as follows: Section 3.2.1 recall some basics of the P system model used along the paper. Among the possible models, our choice has been probabilistic P systems, since probability (in replication, deletion or migration) is one of the pillars of EGT. In Section 3.2.2, we present the design of a family of probabilistic P systems which simulates the behaviour of EGT and a short overview of the computation. It deserves to be remarked that all the simulation is performed by P system rules, without considering oracles or external functions. In this way, MeCoSim simulator [40] has been used for the experiments. Section 3.3 show some examples of EGT cases and how the compartmental structure of P systems can help in the development of the interaction among populations. Finally, some conclusions and hints for future work are presented.

3.2 Methods

3.2.1 Probabilistic P systems

Since Gh. Păun introduced in 1998 the first model of membrane computing [41] till now, many different variants of P systems have been presented. Among them, spiking neural P systems (SNPS) [42] is one of the most widely extended. From the starting model of SNPS, many other features have been added and explored. Among the most recent, the articles *Homogeneous spiking neural P systems with structural plasticity* [43], *Delayed Spiking Neural P Systems with Scheduled Rules* [44] or *Spiking neural P systems with autapses* [45]. Among the recent contributions we can cite

the SNP systems with communication on requests [46, 47] or SNP systems variant used in optimization and for building an arithmetic calculator [48, 49, 50].

Beyond SNPS, many other P system variants have proved their efficiency in order to model real life problems. In particular, probabilistic/stochastic models [37] have shown its efficiency in problems from biological processes.

In this work, it is considered the so-called *Probabilistic P systems* [38] introduced by Cardona *et al.* in 2011. Next, it is briefly summarized some of the main features.

Considering a working alphabet Γ and a membrane structure where different membranes have different labels. Membranes have electrical charges from the set $\{0, +, -\}$ and $\mathbb{R}$ is a finite set of evolution rules of the form

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

where u, u', v and v' are multisets over Γ, i and j are labels[2] and $\alpha, \alpha' \in \{0, +, -\}$.

The representation of information as multisets placed on a membrane structure and the use of biologically inspired rules for performing the evolution of such multisets of objects is common to many P system variants. The main feature of probabilistic P systems is that the rules are endowed with a computable function f_r, $(r \in \mathbb{R})$ such that $dom(f_r) \subseteq \{1, \ldots, T\}$ (with T a natural number) and $range(f_r) \subseteq [0, 1]$ verifying that if $r_1, \ldots, r_z$ are the rules from $\mathbb{R}$ with the same left-hand side (i.e., $u\,[\,v\,]_i^{\alpha}$) then $\sum_{j=1}^{z} f_{r_j}(a) = 1$ for $a = 1, \ldots, T$. Intuitively, such $f_r(a)$ represents the probabilistic constant associated with rule r. In general, it is written as $r : u\,[\,v\,]_i^{\alpha} \xrightarrow{f_r(a)} u'\,[\,v'\,]_j^{\alpha'}$. If $f_r(a) = 1$, then it is denoted by $r : u\,[\,v\,]_i^{\alpha} \rightarrow u'\,[\,v'\,]_j^{\alpha'}$.

The key point is that in this P systems applicable rules are non-deterministic chosen as usual, but, in this case the probability of each rule can be fixed at the beginning of the computation.

The semantics of the P system follow the next principles:

I1 When an object cross a membrane, its polarization may change. In such way, the electrical charges can act as traffic lights, i.e., rules can only be applied if the polarization of the rule is the appropriate.

I2 If a rule can be applied inside a membrane and, in the same step a send-in or send-out rule which changes the polarization of the membrane can also be applied, both rules are applied. In a certain sense, we can consider that the change of the polarization is performed *after* the application of the evolution rules.

3.2.2 Evolutionary Game Theory in P Systems

In this section, the description of a P system which computes the evolution of a population according to Game Theory is provided. The main contribution of

[2] In the original description of probabilistic P systems $i = j$.

this approach is that different sub-populations can be encapsulated in P system membranes. Individuals inside such membranes evolve according to Game Theory principles:

- Each individual gets a *fitness value* obtained by meetings with other individuals according to a chosen strategy (*cooperator* or *defector*).
- One of the individuals is replicated. The probability of replication of an individual is proportional to its fitness value.
- In order to keep constant the total number of individuals, after replication one individual is randomly chosen to be deleted.

Each sub-population evolves following these principles in a membrane. After the execution of these steps, one individual in each membrane is randomly chosen to migrate to another membrane. These migration between membranes is possible due to the intrinsic nature of P systems and it represents a big chance for exploring the behaviour of other populations. Next, the technical details of the proposed model are described.

Let consider a population of $n \times Q$ individuals distributed on Q membranes. In each membrane, there are n individuals identified by $1, \ldots, n$. Due to technical reasons, in this proposed model, n must be even. Each individual choose between two different strategies to follow: *cooperator* or *defector*. When a cooperator meets another cooperator, they both get R (Reward). If a cooperator meets a defector, the cooperator gets S (the Sucker's payoff) and the defector T (Temptation). If two defectors meet, they both get P (Punishment). This can be encoded in a payoff matrix

	cooperate	defect
cooperate	R	S
defect	T	P

or in a 4-tuple $\langle R, S, T, P \rangle$. In this model, each compartment (which is represented by an elementary membrane) has a different payoff matrix. As said previously, individuals can move between compartments (i.e. between sub-populations). In this way, the evolution of the different populations can be studied according to the different payoff matrices and the influence of the communication among compartments. In this version, the number of individuals in each compartment keeps constant along the computation, but the proportion of defectors and cooperators in each compartment can change due to the processes of replication, deletion and migration.

For each compartment $k \in \{1, \ldots, Q\}$, we have a payoff matrix represented by $\langle R_k, S_k, T_k, P_k \rangle$. In this approach, each individual have exactly M meetings per cycle. In that way, a cooperator in the compartment k will have, after A meetings, an accumulated payoff which can be calculated as the sum of A values taken from R_k

and S_k in all the possible ways, i.e., the possible accumulated payoff of a cooperator after A meetings in the membrane k is one of the values of

$$V_{c,k,A} = \{(r \times S_k) + ((N - r) \times R_k) \,|\, r \in \{0, \ldots, A\}\}$$

Analogously, the possible accumulated payoff of a defector after A meetings in the membrane k is one of the values of

$$V_{d,k,A} = \{(r \times T_k) + ((N - r) \times P_k) \,|\, r \in \{0, \ldots, A\}\}$$

In EGT, the *fitness* F of an individual depends on: (1) The accumulated payoff Ac; (2) The number of meetings of the individual M and; (3) A parameter w which we call *influence*. The fitness is computed by the formula

$$F = (1 - w) + w \times \frac{Ac}{M} \tag{2}$$

In this way, each individual will obtain a fitness depending on Ac, M and w and its probability of replication will be proportional to its fitness. In our model, the number of meetings M is fixed and also the influence w and therefore, the fitness F_i of an individual i with accumulated payoff Ac_i is computed as

$$F_i = (1 - w) + w \times \frac{Ac_i}{M} \tag{3}$$

From this equation we have

$$M \times F_i = M \times (1 - w) + w \times Ac_i \tag{4}$$

We can consider that $New\_F_i = M \times F_i$ is the fitness F_i scaled by a constant M and then

$$New\_F_i = M + w \times (Ac_i - M) \tag{5}$$

Since P systems deal with integer numbers, we can approximate Eq. 5 by considering

$$New\_F_i = M + \lfloor w \times (Ac_i - M) \rfloor \tag{6}$$

where $\lfloor x \rfloor$ is the greatest integer less than or equal to x. We define the following set $LF_{c,k}$ of pairs (Ac, New_F_i), where Ac is the accumulated payoff obtained by a cooperator after M meetings in the membrane k and New_F_i the fitness value obtained from Ac according to Eq. 6,

$$LF_{c,k} = \{(Ac, New\_F) \,|\, Ac \in V_{c,k,M}\}$$

We will denote by $MaxF_c$ the maximum value of the fitness that can be reached by a cooperator in any membrane, i.e.,

$$MaxF_c = \max\{B \,|\, (A, B) \in \cup_{k=1}^{k=Q} LF_{c,k}\}$$

In order to formally define a P system which simulates the evolution of the Game Theory, we will consider different type of objects:

- An *individual* will be represented by a tuple $\langle Id, t, Am, Ap\rangle$ where
  - $Id \in \{1, \ldots, n\}$ is the identifier of the individual;
  - $t \in \{c, d\}$ is the strategy of the individual: cooperator (c) or defector (d);
  - Am is the number of meetings in the current configuration;
  - Ap is the accumulated payoff in the current configuration.

  At the beginning of the computation, each compartment has n individuals with identifiers $1, \ldots, n$ and each individual has a strategy. After each cycle, each compartment has again n individuals with identifiers $1, \ldots, n$, but the proportion of defectors and cooperators can change due to the processes of deletion, replication and migration.

- Two special kind of objects are $p_{Id,j}$ and $q_{Id,j}$. Each of these objects represent a payoff unit of the individual with identifier Id: $p_{Id,j}$ if it is a cooperator and $q_{Id,j}$ if it is a defector.

- Other auxiliary objects will also be used: z_i, f_i, h_i, K, L, L_a, L_b, V, def, $coop$ ...

Broadly speaking, the computation is performed according to the following stages:

1. **Meetings stage**. Each individual have M meetings with other randomly chosen individuals. It can have more than one meeting with the same individual. Each meeting modifies the accumulated payoffs of the individuals according to their strategies and increases by 1 the number of meetings of the individuals. From a P system point of view, these meetings will be performed along M steps. In order to be sure that all individuals can be paired and have a meeting in each step, we will consider that the number of individuals n in each compartment is even. The end of this stage will be controlled by changing the polarization of the membrane where the meetings occur. This stage is performed in parallel along the Q elementary membranes.

2. **Replication and killing stage**. After M steps, the meetings between individuals stop and the replication stage begins. In each elementary membrane, only one individual is chosen to replicate. The probability of an individual to be chosen is proportional to its fitness. As pointed out above, the correspondence between the accumulated payoff and the fitness is stored in the sets $LF_{c,k}$ and $LF_{d,k}$. Each of the pairs (A, B) in these sets will produce rules of type

$$[\, \langle Id, c, M, A\rangle \to p_{Id,1} \ldots p_{Id,B}\, \langle Id, c, 0, 0\rangle]^{+}_{k}$$

$$[\,\langle Id, d, M, A\rangle \to q_{Id,1} \ldots q_{Id,B}\,\langle Id, d, 0, 0\rangle]_k^+$$

Rule $[\,\langle Id, c, M, A\rangle \to p_{Id,1} \ldots p_{Id,B}\,\langle Id, c, 0, 0\rangle]_k^+$ takes a cooperator with identifier Id such that, after M meetings, has obtained an accumulated payoff A and produces B different objects $p_{Id,1} \ldots p_{Id,B}$, where B is the fitness associated[3] to the accumulated payoff A, according to the corresponding $LF_{c,k}$. This rule also resets to 0 the accumulated payoff and the number of meetings. Finally, the rule produces $\langle Id, c, 0, 0\rangle$. The case of a defector is analogous.

In parallel with the production of objects $p_{Id,j}$ and $q_{Id,j}$, a set of rules is applied in order to compute the *mutation* process. A unique object L (L for *life*) is produced in each elementary membrane. In the next step of computation, one and only one of the following rules is triggered:

$[L \to L_a]_k^+$ with probability U (the probability of mutation)
$[L \to L_b]_k^+$ with probability V (the probability of *no* mutation)

Let us notice that both rules have the same left-hand-side and the sum of the probabilities is $U + V = 1$. If L_a is produced, the replication process inside membrane k will occur with mutation. If L_b is produced, the replication will be without mutation.

Let us suppose that L_b is generated, i.e., the replication of the randomly chosen individual will be performed without mutation. In such case one and only one of the following rules is applied.

$$[\,p_{Id,j}\, L_b \to coop\, K\,]_k^+ \qquad [\,q_{Id,j}\, L_b \to def\, K\,]_k^+$$

Since the amount of copies of $p_{Id,j}$ and $q_{Id,j}$ represents the fitness of the individual Id (cooperator and defector, respectively), the probability of triggering one of these rules is proportional to B, i.e., proportional to the fitness of the individual. Since there is only one object L_b, only one of these rules is applied. Since L_b denotes no mutation, the object $p_{Id,j}$ produces a new object *coop* and $q_{Id,j}$ produces a new object def. Such objects *coop* and def can be considered flags in order to recall that the individual which will be produced at the end of the replication process will be a cooperator or a defector (respectively).

If L_a is generated, the replication will be performed with mutation. It means that L_a together with a copy of $p_{Id_1,j}$ (i.e., an object which represents a payoff unit of a cooperator) will produce a defector or, analogously, L_a together with a copy of $q_{Id_1,j}$ (i.e., an object which represents a payoff unit of a defector) will produce a cooperator. The rules are the following:

$$[\,p_{Id,j}\, L_a \to def\, K\,]_k^+ \qquad [\,q_{Id,j}\, L_a \to coop\, K\,]_k^+$$

[3] If $B = 0$, then these rules are simply $[\,\langle Id, c, M, A\rangle\langle Id, c, 0, 0\rangle]_k^+$ and $[\,\langle Id, d, M, A\rangle \to \langle Id, d, 0, 0\rangle]_k^+$.

In any case, the application of one of these replication rules produce an object K (K for *killer*). In the next step, K chooses randomly one of the individuals and delete it by application of one of the rules

$$[\, K\, \langle Id, c, 0, 0\rangle \to K_{Id}]_k^+$$
$$[\, K\, \langle Id, d, 0, 0\rangle \to K_{Id}]_k^+$$

The application of such rule produces an object K_{Id} which can be considered a flag for recalling the identifier of the object which has been deleted. Finally we combine the flag K_{Id} with the flag *coop* or def and produce a new individual with the strategy obtained by the probabilistic process of mutation and with the identifier of the deleted individual. The rules are

$$[\, K_{Id}\, coop \to \langle Id, c, 0, 0\rangle]_k^+$$
$$[\, K_{Id}\, def \to \langle Id, d, 0, 0\rangle]_k^+$$

In any case, after the replication and killing stage, in each compartment k there are n individuals with identifiers $1, \ldots, n$.

3. **Migration stage**. At the end of the replication and killing stage, a new object V (V stands for voyager) appears in each elementary membrane k. In parallel, each of these objects V chooses randomly one of the individuals of the compartment and sends it out of the elementary membrane, by the application of one of the rules

$$[\, V\, \langle Id, c, 0, 0\rangle]_k^+ \to V_{k,Id}\, \langle Id, c, 0, 0\rangle\, [\,]_k^-$$
$$[\, V\, \langle Id, d, 0, 0\rangle]_k^+ \to V_{k,Id}\, \langle Id, d, 0, 0\rangle\, [\,]_k^-$$

One of these rules is applied in parallel in each of the Q membranes and therefore, after the application, there are Q individuals in the membrane surrounding the Q elementary membranes. Objects $V_{k,Id}$ can be considered as flags for remembering that the individual sent out from membrane k had identifier Id. In the next step of computation, each of these individuals is randomly paired to an object V_{k,Id_2} and one of the following rules is randomly applied

$$\langle Id_1, c, 0, 0\rangle\, V_{k,Id_2}\, []_k^+ \to [\, \langle Id_2, c, 0, 0\rangle\,]_k^-$$
$$\langle Id_1, d, 0, 0\rangle\, V_{k,Id_2}\, []_k^+ \to [\, \langle Id_2, d, 0, 0\rangle\,]_k^-$$

Object V_{k,Id_2} brings the individual $\langle Id_1, c, 0, 0\rangle$ inside the membrane k and modifies the identifier in order to keep the unicity of the identifiers in the membrane. Let us notice that sending $\langle Id_1, c, 0, 0\rangle$ inside the membrane k changes its identifier to Id_2, but the individual keeps its strategy. With this step, the migration process finishes.

Algorithm 2 Summary

The following cycle of three stages is performed C times

Stage 1 (Meeting stage):

• It is performed in parallel in the Q elementary membranes;

• It takes M steps of the P system;

• Each meeting modifies the accumulated payoff of the involved individuals.

Stage 2 (Replication and killing stage):

• In each elementary membrane:

  ⋆ One individual is chosen with probability proportional to its fitness;

  ⋆ The individual is replicated (maybe with mutation in its strategy).

  ⋆ An individual is removed (killed).

Stage 3 (Migration stage):

• In each elementary membrane, one individual is randomly chosen and sent into another membrane;

The algorithm finishes with a **Stopping stage**.

After computing these stages Meeting, Replication/Killing and Migration, the P system is again ready for a new Meeting stage.

4. **Stopping stage**. The cycle of the previous three stages is performed C times. After these C cycles, the computation stops.

These stages are summarized in Algorithm 2 and a diagram is shown in Figure 11.

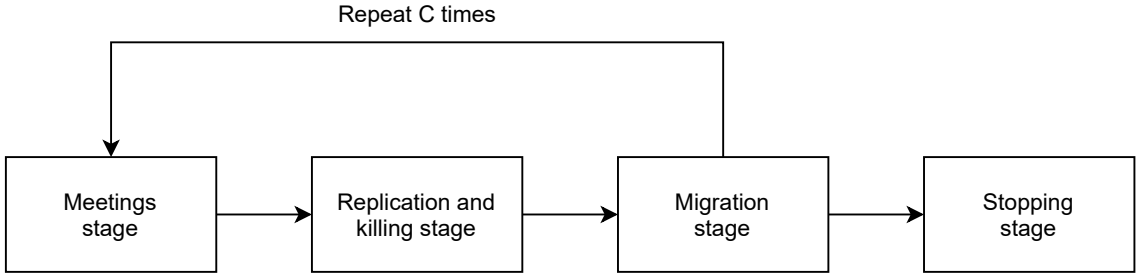

Figure 11: Diagram of the four stages performed in the P System.

Definition of the P system

Let us consider a EGT system with the following parameters[4]:

For each compartment i:

[4]Let us remark that the choice of these parameters (number of compartments Q, number of individuals in each compartment n, etc.) depends on the concrete instance of the problem to be simulated. Once fixed these parameters, the P system Π is completely defined and its evolution, and hence the simulation of the EGT problem, follows the membrane computing principles.

R_i : Reward $\quad S_i$: Sucker's payoff
T_i : Temptation $\quad R_i$: Punishment

Other parameters:

n : Number of individuals in each compartment
M : Number of meetings per cycle
w : Influence
U : Prob. of mutation
V : Prob. of no mutation
C : Number of cycles
Q : Number of compartments

Let us also consider the finite sets of tuples $LF_{c,k}$ and $LF_{d,k}$. In these conditions, let us consider the following P system

$$\Pi = \langle \Gamma, H, EC, \mu, w_1, \ldots, w_Q, w_g, w_s, \mathbb{R} \rangle$$

where

- The alphabet Γ of objects is

$$\begin{aligned}
\Gamma = \quad & \{\langle Id, t, Am, Ap\rangle \,|\, Id \in \{1,\ldots,n\},\ t \in \{c,d\},\\
& \quad Am \in \{0,\ldots,M\},\ Ap \in \cup_{i=1}^{i=Q}\}\\
\cup \quad & \{p_{Id,j} \,|\, Id \in \{1,\ldots,n\}\, j \in \{1,\ldots,MaxF_c\}\}\\
\cup \quad & \{q_{Id,j} \,|\, Id \in \{1,\ldots,n\}\, j \in \{1,\ldots,MaxF_d\}\}\\
\cup \quad & \{z_k \,|\, k \in \{0,\ldots,M+3\}\}\\
\cup \quad & \{L, L_a, L_b, K, V, coop, def\}\\
\cup \quad & \{f_{i,k} \,|\, i \in \{1,\ldots,7\}\, k \in \{1,\ldots,Q\}\}\\
\cup \quad & \{V_{k,Id} \,|\, Id \in \{1,\ldots,n\}\, k \in \{1,\ldots,Q\}\}\\
\cup \quad & \{h_s \,|\, s \in \{0,\ldots,CM+7C-3\}\}
\end{aligned}$$

- $H = \{1,\ldots,Q\} \cup \{g, s\}$ is the set of labels;
- $EC = \{0, +, -\}$ is the set of electrical charges;
- The membrane structure has Q elementary membranes with labels $1,\ldots,Q$; the skin with label s; an intermediary membrane with label g;

$$\mu = [\ [\ [\ ]_1^0\ \cdots\ [\ ]_Q^0\ ]_g^0\ ]_s^0$$

- The initial multisets are $w_j = z_0\, \langle Id, t_{Id}^j, 0, 0\rangle$ for $j \in \{1,\ldots,Q\}$, $Id \in \{1,\ldots,n\}$ and $t_{Id}^j \in \{c,d\}$ is the strategy followed by the Id-th individual in the j-th elementary membrane in the initial configuration. We also have $w_g = \emptyset$ and $w_s = h_0$.

We will also consider the following sets of rules $\mathbb{R}$ (λ represents the empty multiset)

Meeting rules: Let $\langle R_k, S_k, T_k, P_k\rangle$ is the encoding of the payoff matrix inside the compartment $k \in \{1,\ldots,Q\}$.

$RS_1 \equiv [\,\langle Id_1, c, A, B\rangle\langle Id_2, c, A, C\rangle \to \langle Id_1, c, A+1, B+R_k\rangle\langle Id_2, c, A+1, C+R_k\rangle]_k^0$
for $k \in \{1,\ldots,Q\}$, $Id_1, Id_2 \in \{1,\ldots,n\}$, $A \in \{0,\ldots,M-1\}$, $B, C \in V_{c,k,A}$

$RS_2 \equiv [\,\langle Id_1, c, A, B\rangle\langle Id_2, d, A, C\rangle \to \langle Id_1, c, A+1, B+S_k\rangle\langle Id_2, c, A+1, C+T_k\rangle]_k^0$,
for $k \in \{1,\ldots,Q\}$, $Id_1, Id_2 \in \{1,\ldots,n\}$, $A \in \{0,\ldots,M-1\}$,
$B \in V_{c,k,A}$, $C \in V_{d,k,A}$

$RS_3 \equiv [\,\langle Id_1, d, A, B\rangle\langle Id_2, d, A, C\rangle \to \langle Id_1, d, A+1, B+P_k\rangle\langle Id_2, d, A+1, C+P_k\rangle]_k^0$
for $k \in \{1,\ldots,Q\}$, $Id_1, Id_2 \in \{1,\ldots,n\}$, $A \in \{0,\ldots,M-1\}$, $B, C \in V_{d,k,A}$

Replication rules:

$RS_4 \equiv [\,\langle Id, c, M, A\rangle \to p_{Id,1}\ldots p_{Id,B}\langle Id, c, 0, 0\rangle]_k^+$ for $k \in \{1,\ldots,Q\}$, $Id \in \{1,\ldots,n\}$,
$A \in V_{c,k,M}$ and $(A, B) \in LF_{c,k}$

$RS_5 \equiv [\,\langle Id, d, M, A\rangle \to q_{Id,1}\ldots q_{Id,B}\langle Id, d, 0, 0\rangle]_k^+$ for $k \in \{1,\ldots,Q\}$, $Id \in \{1,\ldots,n\}$,
$A \in V_{d,k,M}$ and $(A, B) \in LF_{d,k}$

$RS_6 \equiv [L \xrightarrow{U} L_a]_k^+$, i.e., $[L \to L_a]_k^+$ with probability U (for $k \in \{1,\ldots,Q\}$)
$RS_7 \equiv [L \xrightarrow{V} L_b]_k^+$, i.e., $[L \to L_b]_k^+$ with probability V (for $k \in \{1,\ldots,Q\}$)

$RS_8 \equiv [\,p_{Id,j}\,L_a \to def\,K\,]_k^+$ for $k \in \{1,\ldots,Q\}$,
$Id \in \{1,\ldots,n\}$ and $j \in \{1,\ldots,MaxF_c\}$

$RS_9 \equiv [\,q_{Id,j}\,L_a \to coop\,K\,]_k^+$ for $k \in \{1,\ldots,Q\}$,
$Id \in \{1,\ldots,n\}$ and $j \in \{1,\ldots,MaxF_d\}$

$RS_{10} \equiv [\,p_{Id,j}\,L_b \to coop\,K\,]_k^+$ for $k \in \{1,\ldots,Q\}$,
$Id \in \{1,\ldots,n\}$ and $j \in \{1,\ldots,MaxF_c\}$

$RS_{11} \equiv [\,q_{Id_1,j}\,L_b \to def\,K\,]_k^+$ for $k \in \{1,\ldots,Q\}$,
$Id \in \{1,\ldots,n\}$ and $j \in \{1,\ldots,MaxF_d\}$

$RS_{12} \equiv [\,K_{Id}\,coop \to \langle Id, c, 0, 0\rangle\,]_k^+$ for $k \in \{1,\ldots,Q\}$ and $Id \in \{1,\ldots,n\}$
$RS_{13} \equiv [\,K_{Id}\,def \to \langle Id, d, 0, 0\rangle\,]_k^+$ for $k \in \{1,\ldots,Q\}$ and $Id \in \{1,\ldots,n\}$

Killing rules:

$RS_{14} \equiv [\,K\,\langle Id, t, 0, 0\rangle \to K_{Id}]_k^+$ for $k \in \{1,\ldots,Q\}$, $t \in \{c, d\}$ and $Id \in \{1,\ldots,n\}$

Migration rules:

$RS_{15} \equiv [\, V\, \langle Id, t, 0, 0\rangle]_k^+ \to V_{k,Id}\, \langle Id, t, 0, 0\rangle\, [\,]_k^-$
for $k \in \{1, \ldots, Q\}$, $Id \in \{1, \ldots, n\}$, $t \in \{c, d\}$

$RS_{16} \equiv \langle Id_1, t, 0, 0\rangle\, V_{k,Id_2}\, []_k^- \to [\, \langle Id_2, t, 0, 0\rangle\,]_k^-$
for $k \in \{1, \ldots, Q\}$, $Id_1, Id_2 \in \{1, \ldots, n\}$, $t \in \{c, d\}$

Control rules:

For $k \in \{1, \ldots, Q\}$

$RS_{17} \equiv [\, z_i \to z_{i+1}]_k^0$ for $i \in \{0, \ldots, M-3\} \cup \{M, M+1, M+2\}$
$RS_{18} \equiv [\, z_{M-2} \to z_{M-1}\, f_{0,k}]_k^0$
$RS_{19} \equiv [\, z_{M-1} \to z_M\, L\,]_k^0$
$RS_{20} \equiv [\, z_{M+3} \to V\,]_k^+$

$RS_{21} \equiv [\, f_{0,k}\,]_k^0 \to f_{1,k}\, [\,]_k^+$
$RS_{22} \equiv [\, f_{i,k} \to f_{i+1,k}\,]_g^0$ for $i \in \{1, \ldots, 6\}$
$RS_{23} \equiv f_{7,k}\, [\,]_k^- \to [\, z_0\,]_k^0$

Stopping rules:

$RS_{24} \equiv [\, h_i \to h_{i+1}]_s^0$ for $i \in \{0, \ldots, CM + 7C - 4\}$
$RS_{25} \equiv h_{CM+7C-3}\, [\,]_g^0 \to [h_{CM+7C-3}]_g^+$

Cleaning rules:

$RS_{26} \equiv [\, p_{Id,j} \to \lambda]_k^0$ for $k \in \{1, \ldots, Q\}$,
$Id \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, max\{B \mid (A, B) \in LF_{c,k}\}$

$RS_{27} \equiv [\, q_{Id,j} \to \lambda]_k^0$ for $k \in \{1, \ldots, Q\}$,
$i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, max\{B \mid (A, B) \in LF_{d,k}\}$

$RS_{28} \equiv [\, L_a \to \lambda]_k^0$ for $k \in \{1, \ldots, Q\}$
$RS_{29} \equiv [\, L_b \to \lambda]_k^0$ for $k \in \{1, \ldots, Q\}$

Overview of the Computation

We start with the initial configuration. It consists on Q elementary membranes, the skin (with label s) and the intermediate one (with label g). All of them have polarization 0. Each elementary membrane contains the object z_0, plus n individuals with identifiers $1, \ldots, n$. All the individuals have accumulated meetings and accumulated payoff equals to 0. Each of them has also a strategy c or d (t_i^j denotes the strategy of the individual with identifier i in the elementary membrane j). The skin contains an object h_0.

$$\mathbb{C}_0 = \left[\left[\begin{array}{c} \left[\langle Id_1, t_1^1, 0, 0\rangle \ldots \langle Id_n, t_n^1, 0, 0\rangle \, z_0 \right]_1^0 \\ \cdots \\ \left[\langle Id_1, t_1^Q, 0, 0\rangle \ldots \langle Id_n, t_n^Q, 0, 0\rangle \, z_0 \right]_Q^0 \end{array} \right]_g^0 \; h_0 \right]_s^0$$

From this starting configuration only meeting rules can be applied (together with rules $[\, z_0 \rightarrow z_1]_k^0$ from the set of rules RS_{17} and the rule $[\, h_0 \rightarrow h_1]_s^0$ from the set of rules RS_{24}). Each individual has a meeting with another individual and their accumulated payoff change according to their strategies. In order to fix ideas let us consider a concrete example in the elementary membrane 1 with the payoff matrix settled to $R_1 = 5$, $S_1 = 1$, $T_1 = 10$, $P_1 = 3$. Let us also consider that there are six individuals, four cooperators with identifiers $1, \ldots, 4$ and two defectors with identifiers 5 and 6. In this case, the initial configuration in the elementary membrane 1 is:

$$\left[\begin{array}{l} \langle 1, c, 0, 0\rangle \, \langle 2, c, 0, 0\rangle \, \langle 3, c, 0, 0\rangle \\ \langle 4, c, 0, 0\rangle \, \langle 5, d, 0, 0\rangle \, \langle 6, d, 0, 0\rangle \, z_0 \end{array} \right]_1^0$$

Let us suppose that the non-deterministically chosen rules has been

$$[\, \langle 1, c, 0, 0\rangle \langle 3, c, 0, 0\rangle \rightarrow \langle 1, c, 1, 5\rangle \langle 3, c, 1, 5\rangle]_k^0$$
$$[\, \langle 2, c, 0, 0\rangle \langle 5, d, 0, 0\rangle \rightarrow \langle 2, c, 1, 1\rangle \langle 5, d, 1, 10\rangle]_k^0$$
$$[\, \langle 4, c, 0, 0\rangle \langle 6, d, 0, 0\rangle \rightarrow \langle 4, c, 1, 1\rangle \langle 6, d, 1, 10\rangle]_k^0$$

From a Game Theory point of view, cooperators 1 and 3 meet; cooperator 2 and defector 5 meet; and cooperator 4 with defector 6. In this way, the configuration at time 1 in this elementary membrane is

$$\left[\begin{array}{l} \langle 1, c, 1, 5\rangle \, \langle 2, c, 1, 1\rangle \, \langle 3, c, 1, 5\rangle \\ \langle 4, c, 1, 1\rangle \, \langle 5, d, 1, 10\rangle \, \langle 6, d, 1, 10\rangle \, z_1 \end{array} \right]_1^0$$

These meetings occur in parallel in the Q elementary membranes. Since the polarization of these elementary membranes has not changed, the next step is similar to this one: only rules from the meeting set can be applied together with the corresponding rules from RS_{17} and RS_{24}. The P system goes on with the evolution till reaching the configuration $M - 2$. The configuration at time $M - 2$ is (in the general case) is

$$\mathbb{C}_{M-2} = \left[\left[\begin{array}{c} \left[\langle Id_1, t_1^1, M-2, Y_{M-2}^{1,1}\rangle \ldots \langle Id_n, t_n^1, M-2, Y_{M-2}^{n,1}\rangle \, z_{M-2} \right]_1^0 \\ \cdots \\ \left[\langle Id_1, t_1^Q, M-2, Y_{M-2}^{1,Q}\rangle \ldots \langle Id_n, t_n^Q, M-2, Y_{M-2}^{n,Q}\rangle \, z_{M-2} \right]_Q^0 \end{array} \right]_g^0 \; h_{M-2} \right]_s^0$$

In the next step of computation, the meeting rules are still applied. We also applied the rules RS_{18} and a rule of the set RS_{24}.

$$\mathbb{C}_{M-1} = \left[\left[\begin{array}{l} \left[\ldots \langle Id_j, t_j^1, M-1, Y_{M-1}^{j,1} \rangle \ldots \; z_{M-1} \, f_{0,1} \right]_1^0 \\ \qquad \ldots \\ \left[\ldots \langle Id_j, t_j^Q, M-1, Y_{M-1}^{j,Q} \rangle \ldots \; z_{M-1} \, f_{0,Q} \right]_Q^0 \end{array} \right]_g^0 \; h_{M-1} \right]_s^0$$

In each elementary membrane k in the configuration $\mathbb{C}_{M-1}$ a new object $f_{0,k}$ appears. From this configuration, meeting rules can be applied together with RS_{19}, rules from RS_{21} and the corresponding rule from the set RS_{24}. The application of rules from R_{21} changes the polarization of the elementary membranes, but according to the principle **I2**, we consider that the evolution inside this membrane can be performed in the same computational step. In this way, the configuration at time M is

$$\mathbb{C}_M = \left[\left[\begin{array}{l} \left[\ldots \langle Id_j, t_j^1, M, Y_M^{j,1} \rangle \ldots \; z_M \, L \right]_1^+ \; f_{1,1} \\ \qquad \ldots \\ \left[\ldots \langle Id_j, t_j^Q, M, Y_M^{j,Q} \rangle \ldots \; z_M \, L \right]_Q^+ \; f_{1,Q} \end{array} \right]_g^0 \; h_M \right]_s^0$$

Since the polarization of the elementary membranes has changed to $+$, now the meeting rules cannot be applied and the replication and killing stage starts. Each individual $\langle Id, c, M, A \rangle$ in the membrane k evolves to an individual $\langle Id, c, 0, 0 \rangle$ together with B objects $p_{Id,1}, \ldots p_{Id,B}$ where (A, B) is a pair in $LF_{c,k}$. This evolution is performed by the application of the corresponding rule from RS_4. As pointed out above, the fitness B corresponding to the accumulated payoff A is expressed by the object $p_{Id,1} \ldots p_{Id,B}$. The application of the rule also resets the accumulated meetings and payoff of the individual to 0. The case of the defectors and the application of the rules from the set RS_5 is analogous.

In each elementary membrane, a new objects L has appeared. L starts the replication process. In each elementary membrane, one and only one of the rules RS_6 or RS_7 is triggered according to the probability of mutations. If L_a is produced, then there is mutation in the replication process; otherwise, if L_b is produced, there is no mutation. Rules RS_{17} and the corresponding from the sets RS_{22} and RS_{24} are also applied.

$$\mathbb{C}_{M+1} = \left[\left[\begin{array}{l} \left[\ldots \langle Id_j, c, 0, 0 \rangle q_{Id_j,1} \ldots p_{1,B_j^1} \; \ldots \; z_{M+1} \, L_{a/b} \right]_1^+ \; f_{2,1} \\ \qquad \ldots \\ \left[\ldots \langle Id_j, d, 0, 0 \rangle q_{Id_j,1} \ldots q_{1,B_j^1} \; \ldots \; z_{M+1} \, L_{a/b} \right]_1^+ \; f_{2,Q} \end{array} \right]_g^0 \; h_{M+1} \right]_s^0$$

In the next step the replication stage starts. Let us consider that L_b was produced in the previous step. The object L_b is interpreted as the *no-mutation* object. This object is involved in the application of rules from RS_{10} and RS_{11}. Since $p_{Id,j}$ is a fitness unit of a cooperator and L_b denotes that there is no mutation, the application of the rule $RS_{10} \equiv [\, p_{Id,j}\, L_b \rightarrow coop\, K\,]_k^+$ will produce a flag *coop* which will be used to recall the strategy of the offspring. Rules for $_8$ to RS_{11} consider all the possibilities of producing a flag *coop* or def depending on the strategy of the individual chosen to be replicated (denoted by $p_{Id,j}$ or $q_{Id,j}$ and the probability of mutation. Let us notice that, in any case, a new object K is produced. Rules from RS_{17}, RS_{22} and RS_{24} are also triggered.

$$\mathbb{C}_{M+2} = \left[\left[\begin{array}{l} \Big[\ldots \langle Id_j, c, 0, 0\rangle q_{Id_j,1} \ldots p_{1,B_j^1} \ \ldots \ z_{M+2}\, coop/def\, K \Big]_1^+ \ f_{3,1} \\ \qquad \ldots \\ \Big[\ldots \langle Id_j, d, 0, 0\rangle q_{Id_j,1} \ldots q_{1,B_j^1} \ \ldots \ z_{M+2}\, coop/def\, K \Big]_1^+ \ f_{3,Q} \end{array} \right]_g^0 \ h_{M+2} \right]_s^0$$

In any case, regardless which of the rules from RS_8 to RS_{11} is triggered, an object K will appear in the elementary membrane. As we will see below, such object K will be used to delete one of the individuals of the membrane randomly chosen. Let us notice the special case where all the individuals in the membrane are defectors all of them and the *punishment* P_k of the payoff matrix associated to the membrane is $P_k = 0$. In such case, all the possible meetings couple two defectors and their accumulated payoffs are constant (increased by $P_k = 0$). In such case, objects $q_{i,j}$ are never produced by RS_5 and therefore, RS_9 nor RS_{11} are applied. This means that in such membrane there is no replication. As a secondary effect, since RS_9 and RS_{11} are not applied, then the object K is not produced, an none of the individuals of the membrane is deleted. There is no replication an no deletion in this elementary membrane, but it is still possible the migration and then, the possibility of introducing cooperators in the membrane.

In the next step, one of the killing rules from RS_{14} is applied in each elementary membrane. The object K randomly chooses an individual and removes it. The result of the application of the rule is a new object K_{Id} which recalls the identifier of the deleted individual. Rules from RS_{17}, RS_{22} and RS_{24} are also triggered.

$$\mathbb{C}_{M+3} = \left[\left[\begin{array}{l} \Big[\langle Id_j, c, 0, 0\rangle q_{Id_j,1} \ldots p_{1,B_j^1} \ \ldots \ z_{M+3}\, coop/def\, K_{Id_1} \Big]_1^+ \ f_{4,1} \\ \qquad \ldots \\ \Big[\langle Id_j, d, 0, 0\rangle q_{Id_j,1} \ldots q_{1,B_j^1} \ \ldots \ z_{M+3}\, coop/def\, K_{Id_Q} \Big]_1^+ \ f_{4,Q} \end{array} \right]_g^0 \ h_{M+3} \right]_s^0$$

In the next step the process of replication and killing finished with the application of one rule from RS_{12} or RS_{13}. The flag which recalls the strategy of the

offspring together with the flag which recalls the identifier of the deleted individual and combined in order to produce a new individual with the obtained strategy and the same identifier as the deleted individual. In this case, after the replication and killing stage, the set of identifiers $1, \ldots, n$ is kept in each elementary membrane. Rules from RS_{20}, RS_{21} and RS_{24} are also applied. Rules from RS_{20} produce a new object V inside each elementary membrane and the migration stage starts.

$$\mathbb{C}_{M+4} = \left[\left[\begin{array}{c} \left[\langle Id_j, c, 0, 0 \rangle q_{Id_j,1} \ldots p_{1,B_j^1} \ \ldots \ V \right]_1^+ \ f_{5,1} \\ \ldots \\ \left[\langle Id_j, d, 0, 0 \rangle q_{Id_j,1} \ldots q_{1,B_j^1} \ \ldots \ V \right]_1^+ \ f_{5,Q} \end{array} \right]_g^0 \ h_{M+4} \right]_s^0$$

In the next step, rules form RS_{15} are applied. The object V chooses randomly an individual and sends it out of the elementary membrane. This happens in parallel in all the Q elementary membranes, so in the next configuration there are Q individuals in the membrane g which surrounds the elementary membranes. Each rule from RS_{15} also produce a flag $V_{k,Id}$ which recalls the identifier Id of the individual which has been sent out from the elementary membrane k. Rules from RS_{22} and RS_{24} are also applied.

$$\mathbb{C}_{M+5} = \left[\left[\begin{array}{c} \left[\langle Id_j, c, 0, 0 \rangle q_{Id_j,1} \ldots p_{1,B_j^1} \right]_1^- \langle \, Id_j, c, 0, 0 \rangle \\ \ldots \\ \left[\langle Id_j, d, 0, 0 \rangle q_{Id_j,1} \ldots q_{1,B_j^1} \right]_1^- \langle \, Id_j, c, 0, 0 \rangle \\ f_{6,1} \ldots f_{6,Q} \quad V_{1,Id_1} \ldots V_{Q,Id_Q} \end{array} \right]_g^0 \ h_{M+5} \right]_s^0$$

Next, rules from RS_{16} are applied. Each object $V_{k,Id}$ chooses randomly an individual from the membrane g and sends it into the membrane k. Since there are Q objects different $V_{k,Id}$ and Q individuals, only one individual is sent into each elementary membrane k. The application of the rule also changes the identifier of the chosen individual and takes the identifier associated to $V_{k,Id}$. In this way, the individual which arrives to membrane k will take the same identifier as the individual which was sent out in the previous step. In this way, all the individuals inside each membrane will have the identifiers $1, \ldots, n$ again. Rules from RS_{22} and RS_{24} are also applied.

$$\mathbb{C}_{M+6} = \left[\left[\begin{array}{c} \left[\langle Id_j, c, 0, 0 \rangle q_{Id_j,1} \ldots p_{1,B_j^1} \ \ldots \right]_1^- \ f_{7,1} \\ \ldots \\ \left[\langle Id_j, d, 0, 0 \rangle q_{Id_j,1} \ldots q_{1,B_j^1} \ \ldots \right]_1^- \ f_{7,Q} \end{array} \right]_g^0 \ h_{M+6} \right]_s^0$$

In the next step, rules from RS_{23} and the corresponding from RS_{24} are also

applied. Rules from RS_{23} send an object z_0 inside each elementary membrane and also changes the polarization, so the meeting stage can start again.

Let us notice that the configuration $\mathbb{C}_{M+7}$ is similar to $\mathbb{C}_0$. The elementary membranes have n individuals with identifiers $1, \ldots, n$ and accumulated payoffs and meetings equal to 0. All the elementary membranes also have an object z_0 and its electrical charge is 0. With this configuration the first cycle has concluded. The differences between $\mathbb{C}_{M+7}$ and $\mathbb{C}_0$ are:

- The number of the individuals in each elementary membrane is n, but it may be changed the proportion of cooperators and defectors (due to the deletions, replications and migrations).
- The counter h has reached h_{M+7}.
- In $\mathbb{C}_{M+7}$ several objects $p_{Id,j}$ and $q_{Id,j}$ appear. These objects can be considered garbage and will be deleted in the next step by the set of rules RS_{26} and RS_{27}. As pointed above, in the case of that L_a or L_b has been produced but it has not been consumed, such object can also be deleted by rules RS_{28} and RS_{29}.

After other $M+7$ steps, the configuration $\mathbb{C}_{2\times(M+7)}$ is reached and this is the end of the second cycle. Analogously, the $(C-1)$-th cycle is finished after $(C-1)\times(M+7)$ steps. Let us consider the configuration at the step $M+4+((C-1)\times(M+7)) = CM+7C-3$ which is analogous to the configuration at step $M+4$

$$\mathbb{C}_{CM+7C-3} = \left[\left[\begin{array}{c} \left[\langle Id_j,c,0,0\rangle q_{Id_j,1}\ldots p_{1,B_j^1}\ \ldots\ V\right]_1^+\ f_{5,1} \\ \cdots \\ \left[\langle Id_j,d,0,0\rangle q_{Id_j,1}\ldots q_{1,B_j^1}\ \ldots\ V\right]_1^+\ f_{5,Q} \end{array} \right]_g^0 h_{CM+7C-3} \right]_s^0$$

The transition from this configuration to the next one is similar to the transition from $\mathbb{C}_{M+4}$ to $\mathbb{C}_{M+5}$. The difference is that no more rules from the set RS_{24} are applied and now rule RS_{25} is triggered. The application of this rule changes the polarization of the membrane with label g from 0 to $+$.

$$\mathbb{C}_{CM+7C-2} = \left[\left[\begin{array}{cc} \left[\langle Id_j,c,0,0\rangle q_{Id_j,1}\ldots p_{1,B_j^1}\right]_1^-\langle\, Id_j,c,0,0\rangle & \\ \cdots & \\ \left[\langle Id_j,d,0,0\rangle q_{Id_j,1}\ldots q_{1,B_j^1}\right]_1^-\langle\, Id_j,c,0,0\rangle & h_{CM+7C-3} \\ f_{6,1}\ldots f_{6,Q}\quad V_{1,Id_1}\ldots V_{Q,Id_Q} & \end{array} \right]_g^+ \right]_s^0$$

Since the polarization of the membrane g is now positive, rules from RS_{22} are no longer applied and objects $f_{4,k}$ do not evolve any more. Rules from RS_{16} are applied and the migration of the C-th cycle finishes.

$$\mathbb{C}_{CM+7C-1} = \left[\left[\begin{array}{cc} \left[\langle Id_j, c, 0, 0\rangle q_{Id_j,1} \ldots p_{1,B_j^1} \ \ldots\right]_1^- & f_{6,1} \\ \ldots & h_{CM++7C-3} \\ \left[\langle Id_j, d, 0, 0\rangle q_{Id_j,1} \ldots q_{1,B_j^1} \ \ldots\right]_1^- & f_{6,Q} \end{array} \right]_g^+ \right]_s^0$$

Since the polarization of the elementary membranes is now negative, the meeting and rules cannot be applied. No more rules in general can be applied and $\mathbb{C}_{CM+7C-1}$ is a halting configuration. In each of the Q membranes there are n individuals which can be considered the output of the computation.

3.2.3 MeCoSim

In order to implement and simulate the proposed P System, P-Lingua and MeCoSim are used. P-Lingua is a programming language for Membrane Computing. It is designed to implement P Systems and it incorporate some built-in simulators for the supported models [51]. MeCoSim is a *General Purpose Application* that implements a version of P-Lingua, including its core and simulators. In MeCoSim it is possible to model, design, simulate, analyze and verify P system models [52]. One of the main features of MeCoSim is the ability to run multiple simulations with different initial conditions using input files and parameters in the model definition. Another important features is the step by step execution of simulations, which allows to browse the content of every membrane present in the P System.

All the experiments carried out in this work have been run in MeCoSim using the simulator named *binomial*.

3.3 Experimentation and Results

In this section, the implementation of the proposed model by using the probabilistic P systems simulator MeCoSim [39] is presented. The model have been tested using the classical games involving cooperation, Prisoner's dilemma and Snowdrift game[5]. The aim of this section is not to do a systematic analysis of these two games but to demonstrate a way to use a standard P systems simulator to run relevant examples of EGT, including the possibility of using compartments to simulate structured populations and migration, that can be used to enrich the study of cooperation and population dynamics [31, 54, 55, 56, 57], and, at the same time, enhance the study of P systems and ecological dynamics [39].

Simulations have been conducted using a computer with CPU Intel Core i7 and 16Gb of RAM. Due to the limited computational resources available, some parameters in the experiments have been limited. The number of individuals (cooperators

[5]A detailed description of Prisoner's dilemma and Snowdrift game, among many other classical games involving cooperation, can be found in [53].

or defectors) are restricted to a maximum of 20 and the number of cycles to a maximum of 20000, depending on the experiment. For the experiments in which Prisoner's dilemma is simulated, the number of individuals is 20, divided in 10 cooperators and 10 defectors. The number of cycles is 10000. For the Snowdrift game experiment, the number of individuals is also 20 (10 cooperators and 10 defectors) and the number of cycles is 20000. The last experiment uses two compartments, one for the Prisoner's dilemma and the other one for the Snowdrift game. The total number of individuals is 40 (20 in each compartment) and the number of cycles is 20000.

The implementation of the model in P-Lingua is the following. As it can be seen, the *Model* function receives the parameters needed to define the P System. The code provided in this work it is adapted for two games.

```
@model<probabilistic>
def Model(R{1},S{1},T{1},P{1},R{2},S{2},T{2},P{2},cs,ds,M,w,U,C)
{
  /*
  R: Reward
  S: Sucker
  T: Temptation
  P: Punishment
  cs: Initial number of cooperators
  ds: Initial number of defectors
  M: Number of meetings per cycle
  w: Influence
  U: Prob. of mutation
  C: Number of cycles
  */

  let MAX_AC_C = M*R{1};
  let MAX_AC_D = M*T{2};
  let MAX_FC = (M + @floor(w * (MAX_AC_C - M)));
  let MAX_FD = (M + @floor(w * (MAX_AC_D - M)));
  let n = cs + ds;

  /* Number of membranes */
  let Q = 2;

  /* Membrane structure */
  @mu = [[[]'1[]'2]'g]'s;

  /* Input multiset for membrane 1 */
  @ms(1) += c{ci,0,0} : 1 <= ci <= cs;
  @ms(1) += d{di,0,0} : cs+1 <= di <= cs+ds;
  @ms(1) += z{0};

  /* Input multiset for membrane 2 */
  @ms(2) += c{ci,0,0} : 1 <= ci <= cs;
  @ms(2) += d{di,0,0} : cs+1 <= di <= cs+ds;
  @ms(2) += z{0};

  /* Input multiset for membrane g */
  @ms(g) += w;

  /* Input multiset for membrane s */
  @ms(s) += h{0};

  /* Rules */
  /* R1 */
  [c{Id1, A, (r1 * S{k}) + ((A-r1) * R{k})}, c{Id2, A, (r2 * S{k}) + ((A-r2) * R{k})} --> c{Id1, A + 1, (r1 * S{k}) + ((A-r1) * R{k}) + R{k}}, c{Id2, A + 1, (r2 * S{k}) + ((A-r2) * R{k}) + R{k}}]'{k} :: 1.0 : 1 <= k <= Q, 0 <= Id1 <= n, 0 <= Id2 <= n, 0 <= r1 <= A, 0 <= r2 <= A, 0 <= A <= M-1;

  /* R2 */
  [c{Id1, A, (r1 * S{k}) + ((A-r1) * R{k})}, d{Id2, A, (r2 * T{k}) + ((A-r2) * P{k})} --> c{Id1, A + 1, (r1 * S{k}) + ((A-r1) * R{k}) + S{k}}, d{Id2, A + 1, (r2 * T{k}) + ((A-r2) * P{k}) + T{k}}]'{k} :: 1.0 : 1 <= k <= Q, 0 <= Id1 <= n, 0 <= Id2 <= n, 0 <= r1 <= A, 0 <= r2 <= A, 0 <= A <= M-1;

  /* R3 */
  [d{Id1, A, (r1 * T{k}) + ((A-r1) * P{k})}, d{Id2, A, (r2 * T{k}) + ((A-r2) * P{k})} --> d{Id1, A + 1, (r1 * T{k}) + ((A-r1) * P{k}) + P{k}}, d{Id2, A + 1, (r2 * T{k}) + ((A-r2) * P{k}) + P{k}}]'{k} :: 1.0 : 1 <= k <= Q, 0 <= Id1 <= n, 0 <= Id2 <= n, 0 <= r1 <= A, 0 <= r2 <= A, 0 <= A <= M-1;

  /* R4 */
  +[c{Id, M, (r1 * S{k}) + ((A-r1) * R{k})} --> &{p{Id,j}, c{Id,0,0}}:{1<=j<=(M + @floor(w * (((
```

```
    r1 * S{k}) + ((A-r1) * R{k})) - M)))}]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n, 0 <= r1 <= A,
     1 <= A <= M;

  /* R5 */
  +[d{Id, M, (r1 * T{k}) + ((A-r1) * P{k})} --> &{q{Id,j}, d{Id,0,0}}:{1<=j<=(M + @floor(w * (((
    r1 * T{k}) + ((A-r1) * P{k})) - M)))}]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n, 0 <= r1 <= A,
     1 <= A <= M;

  /* R6 */
  +[L --> La]'{k} :: U : 1 <= k <= Q;

  /* R7 */
  +[L --> Lb]'{k} :: 1.0 - U : 1 <= k <= Q;

  /* R8 */
  +[p{Id,jj},La --> D,K]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n, 1 <= jj <= MAX_FC;

  /* R9 */
  +[q{Id,jj},La --> C,K]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n, 1 <= jj <= MAX_FD;

  /* R10 */
  +[p{Id,jj},Lb --> C,K]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n, 1 <= jj <= MAX_FC;

  /* R11 */
  +[q{Id,jj},Lb --> D,K]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n, 1 <= jj <= MAX_FD;

  /* R12 */
  +[K{Id},C --> c{Id,0,0}]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n;

  /* R13 */
  +[K{Id},D --> d{Id,0,0}]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n;

  /* R14 */
  +[K,c{Id,0,0} --> K{Id}]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n;
  +[K,d{Id,0,0} --> K{Id}]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n;

  /* R15 */
  +[V, c{Id,0,0}]'{k} --> V{k,Id},c{Id,0,0}-[c{Id,0,0}]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n;
  +[V, d{Id,0,0}]'{k} --> V{k,Id},d{Id,0,0}-[d{Id,0,0}]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n;

  /* R16 */
  c{Id1,0,0}, V{k,Id2} -[]'{k} --> -[c{Id2,0,0}]'{k} :: 0 : 1 <= k <= Q, 1 <= Id1 <= n, 1 <= Id2
    <= n;
  d{Id1,0,0}, V{k,Id2} -[]'{k} --> -[d{Id2,0,0}]'{k} :: 0 : 1 <= k <= Q, 1 <= Id1 <= n, 1 <= Id2
    <= n;

  /* R17 */
  [z{i} --> z{i+1}]'{k} :: 1.0 : 1 <= k <= Q, 0 <= i <= M-3;
  +[z{i} --> z{i+1}]'{k} :: 1.0 : 1 <= k <= Q, M <= i <= M+2;

  /* R18 */
  [z{M-2} --> z{M-1}, f{0,k}]'{k} :: 1.0 : 1 <= k <= Q;

  /* R19 */
  [z{M-1} --> z{M},L]'{k} :: 1.0 : 1 <= k <= Q;

  /* R20 */
  +[z{M+3} --> V]'{k} :: 1.0 : 1 <= k <= Q;

  /* R21 */
  [f{0,k}]'{k} --> f{1,k}+[]'{k} :: 1.0 : 1 <= k <= Q;

  /* R22 */
  [f{i,k} --> f{i+1,k}]'g :: 1.0 : 1 <= k <= Q, 1 <= i <= 6;

  /* R23 */
  f{7,k}-[]'{k} --> [z{0}]'{k} :: 1.0 : 1 <= k <= Q;

  /* R24 */
  [h{i} --> h{i+1}]'s :: 1.0 : 0 <= i <= C*M + 7*C - 4;

  /* R25 */
  h{C*M + 7*C - 3}[]'g --> +[h{C*M + 7*C - 3}]'g :: 1.0;

  /* R26 */
  [p{Id,jj} --> ]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n, 1 <= jj <= MAX_FC;

  /* R27 */
  [q{Id,jj} --> ]'{k} :: 1.0 : 1 <= k <= Q, 1 <= Id <= n, 1 <= jj <= MAX_FD;

  /* R28 */
  [La -->]'{k} :: 1.0 : 1 <= k <= Q;

  /* R29 */
  [Lb -->]'{k} :: 1.0 : 1 <= k <= Q;
}
```

In Figures 12 and 13 we compute the long-term average of the number of cooperators by calculating the average number of cooperators considering the entire

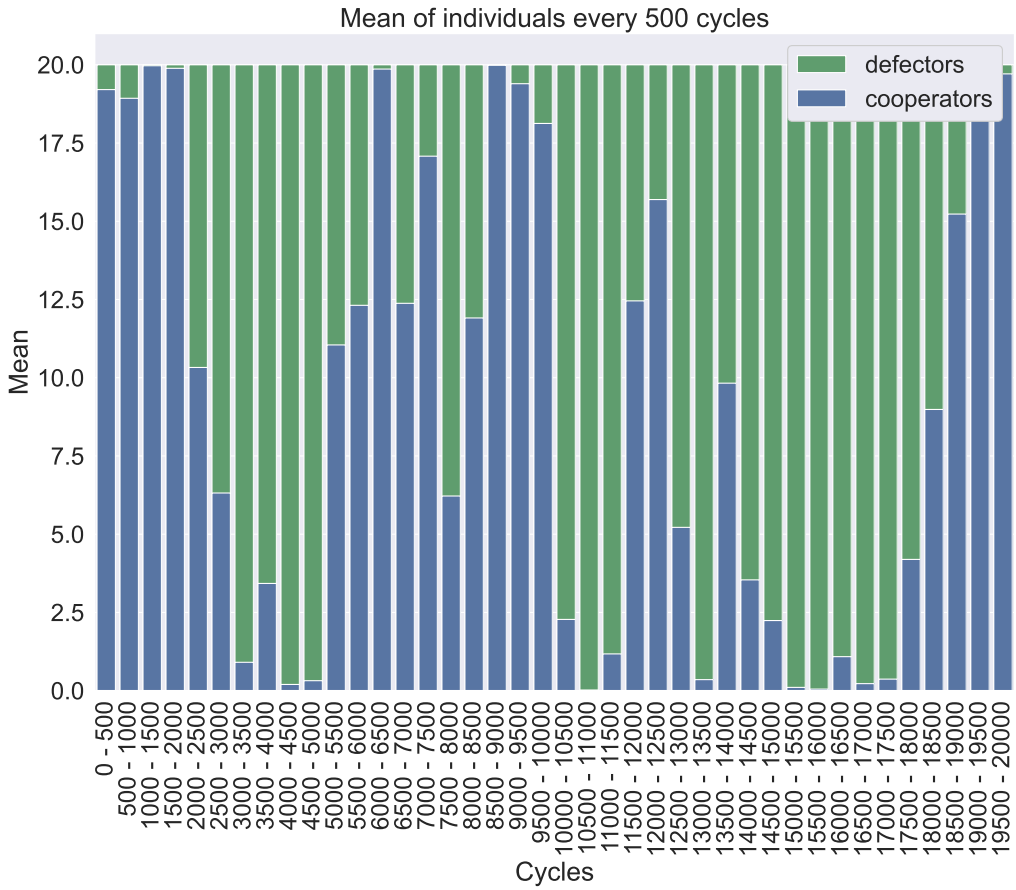

Figure 12: Evolution of cooperators and defectors in Snowdrift game during 20000 cycles (each bar represents the mean every 500 cycles). We can observe that cooperators and defectors can co-exist, as we can also observe in the long-term average of cooperators (skipping first 1000 cycles, in order to avoid adding the transient effects of the initial configuration in the calculated long-term average) which is 8.73. Cooperators can invade a region full of defectors, while defectors can invade a region full of cooperators. Payoff matrix used in this experiment is composed of Reward = 5, Sucker's payoff = 4, Temptation = 6 and Punishment = 3. Initial individuals are 10 cooperators and 10 defectors. Mutation probability is 0.01 and parameter influence w is 0.5.

simulation (skipping the first 1000 cycles). Intuitively, this provides the expected number of cooperators present in the population in the long term. As one would expect, the simulator correctly shows that cooperators and defectors can co-exists only in the Snowdrift game (Figure 12), while in the Prisoner's dilemma, the population is dominated by the defectors in the early cycles (Figure 13).

Figure 14 shows the evolution of a population of 40 individuals split into 2 membranes with 20 individuals each. In the first one (membrane 1) the individuals evolve with the Prisoner's dilemma matrix and the individuals in membrane 2 evolves according to the Snowdrift game. The system is evolved for 20000 cycles (each bar

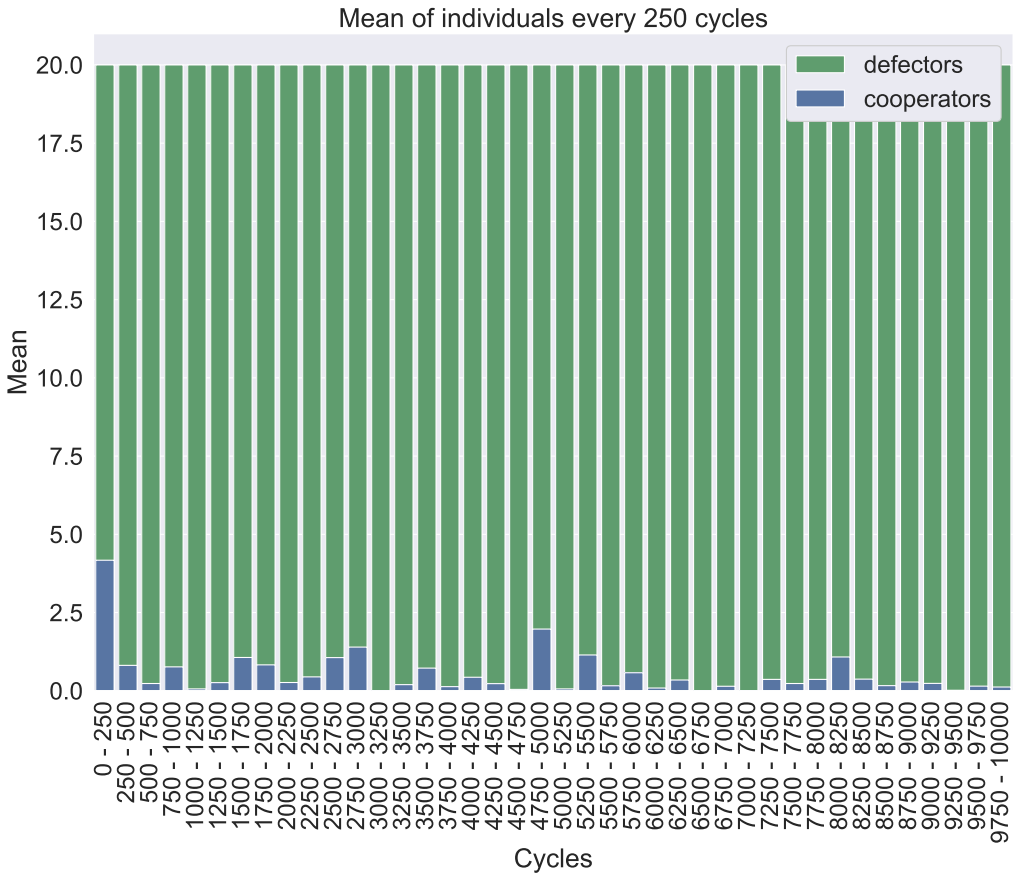

Figure 13: Evolution of cooperators and defectors in Prisoner's dilemma game during 10000 cycles (each bar represents the mean every 250 cycles). We can observe that the population is composed by mostly defectors as we can also observe in the long-term average of cooperators (skipping first 1000 cycles) which is just 0.42. As it can be seen, cooperators have a lot of difficulties invading a region full of defectors. Payoff matrix is composed of Reward = 3, Sucker's payoff = 1, Temptation = 7 and Punishment = 2. Initial individuals are 10 cooperators and 10 defectors. Mutation probability is 0.01 and parameter influence w is 0.5.

represents the mean every 500 cycles). In this case, the two previous games are encoded in different elemental membranes. As stated before, membrane 1 encodes the Snowdrift game whereas membrane 2 encodes Prisoner's dilemma. In this configuration, migration rules are applied in order to interchange individuals between membranes. In Figure 14 we can see that the long-term average number of cooperators is distinct from the one obtained in the two scenarios (Prisoner's dilemma and Snowdrift) when studied independently, (Figure 13 and 14) which highlights the relevance of combining compartments. More generally, the presented approach allows to study systems with complex structures where in different regions can be applied different games, with distinct payoff matrices and individuals migrating across membranes. In this way, the cellular-inspired structure of membrane computing can be

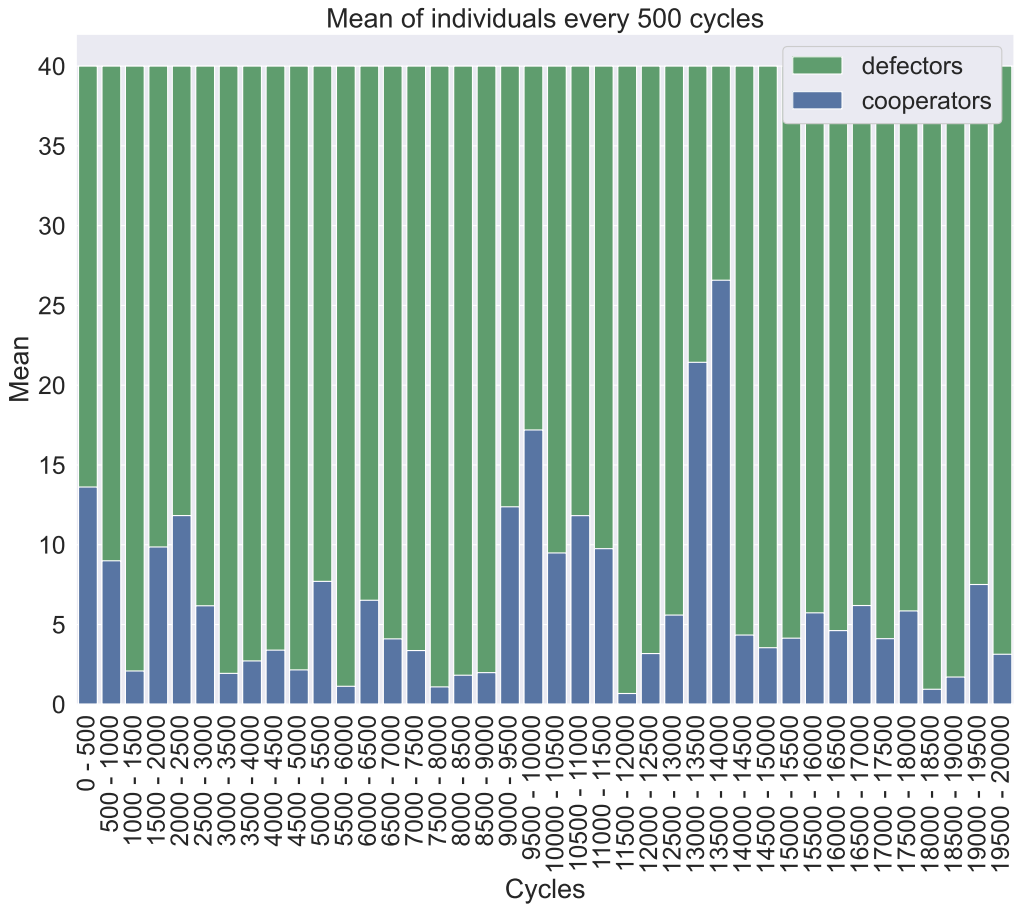

Figure 14: This figure shows the evolution of a population of 40 individuals split into 2 membranes with 20 individuals each. In membrane 1, the individuals evolve with the Prisoner's dilemma payoff matrix and the individuals in membrane 2 evolves according to the Snowdrift game. The system has evolved during 20000 cycles (each bar represents the mean every 500 cycles). Long-term average of cooperators for the overall system (skipping first 1000 cycles) is 6.26. In this configuration, each membrane encode one of the games. Membrane 1 has a payoff matrix composed of Reward = 5, Sucker's payoff = 4, Temptation = 6 and Punishment = 3 whereas membrane 2 has Reward = 3, Sucker's payoff = 1, Temptation = 7 and Punishment = 2 (i.e., these payoffs correspond to the Prisoner's dilemma and Snowdrift games). Both membranes has the same configuration for the rest of parameters: mutation probability is 0.01 and parameter influence w is 0.5.

used to study the spreading of behaviours in structured populations.

3.4 Conclusions

EGT is a mathematical and computational framework which is used to study the spreading of behaviours (often referred as strategies) in evolving populations. An

important problem approached using EGT is the study of the resilience of cooperation in structured populations, i.e., organized according to specific structures. Very often, the problem is studied using ad-hoc computational models.

In this work, it is proposed a general, flexible, way to encode and simulate EGT problems in P systems. This allows us to provide a formal way to computationally study the dynamics of evolving populations, and, at the same time, use well-known simulators and languages for P systems (such as MeCoSim and P-Lingua) to investigate, in-silicio, the spreading of strategies in structured populations, organized in compartments where individuals can meet, replicate and migrate. To demonstrate the feasibility of the proposed approach, it is encoded two well-known games to study cooperation (Prisoner's dilemma and Snowdrift game) into P systems and simulated their dynamics using MeCoSim. As expected, cooperators and defectors can co-exists in the Snowdrift game, while in the Prisoner's dilemma, the population is mostly composed by defectors (even in the early cycles). The proposed approach allows also to simulate populations organized in different compartments with migration across the different membranes, where different games can be associated to different membranes (i.e. it is possible to study the dynamics of populations that interact with each other). Despite it is not the aim of this work to focus on any specific scenario, we believe that the proposed encoding of EGT into P systems will constitute an innovative framework to study and simulate the dynamics of ecological processes related to the spreading of new behaviours in structured populations, a key general research question in EGT. More generally, the proposed approach can provide an alternative general framework to study the evolutionary dynamics in structured populations, enriching the research area that studies ecological dynamics using P systems, and the area that studies the spreading of behaviours in structured populations.

4 Final thoughts

In this work, the two research works carried out during the academic year have been developed. Both of the papers have been submitted to journals. Paper *PBIL for optimizing Inception Module in Convolutional Neural Networks* have been submitted to the Logic Journal of the IGPL and have already been accepted for publication. Paper *Evolutionary Game Theory in a Cell: A Membrane Computing Approach* have been submitted to the special issue about Membrane Computing of the Information Sciences journal and is currently being reviewed. Both papers have a strong inspiration in nature.

The paper *PBIL for optimizing Inception Module in Convolutional Neural Networks* proposes a methodology to optimize the hyperparameters of the Inception-A block using PBIL algorithm. In order to not generate individuals with incompatible kernel sizes (based on the requirements of the problem) a special codification is implemented. In addition, the use of a single epoch during the hyperparameters optimization allows to reduce the execution time drastically. Hence, the carbon footprint is also reduced. Results demonstrate that the optimized architecture of Inception-A module using only 3 blocks, and therefore a lower number of trainable parameters, achieves an excellent performance. One of the best things of this paper is that the future work is abundant. There are ton of things that can be tested: others neural network architectures, others optimization algorithms, the use of more than one epoch to compute the fitness of individuals, the optimization of more hyperparameters, etc.

The paper *Evolutionary Game Theory in a Cell: A Membrane Computing Approach* proposes a general way to encode Evolutionary Game Theory problems into Membrane Computing. A novel computational framework which can be used to study, analyze and simulate the spreading of behaviours in structured populations organized in communicating compartments is also proposed. The proposed approach provides an alternative general framework to study the evolutionary dynamics in structured populations, enriching the research area that studies ecological dynamics using P systems, and the area that studies the spreading of behaviours in structured populations. As future work, optimization of the P System can lead to more ambitious experiments in which more than two compartments are used to hold different populations.

These two papers are only two cases of nature-inspired computation. In the computer science area (and mathematics), there are tons of examples of nature inspiration. Thus, we can say nature has had a great impact in the way research is done.

As stated before, the two papers open up a broad range of future work. Hence, new research lines can derive to continue the work.

References

[1] Shumeet Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, January 1994.

[2] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. In *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 38–46, 1995.

[3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[4] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017.

[5] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[6] Estrategia Nacional de Inteligencia Artificial. Technical report, Gobierno de España, 2020.

[7] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics.

[8] Iván Méndez-Jiménez and Miguel Cárdenas-Montes. Modelling and forecasting of the ^{222}Rn radiation level time series at the Canfranc Underground Laboratory. In *Hybrid Artificial Intelligent Systems - 13th International Conference, HAIS 2018, Oviedo, Spain, June 20-22, 2018, Proceedings*, volume 10870 of *Lecture Notes in Computer Science*, pages 158–170. Springer, 2018.

[9] Iván Méndez-Jiménez and Miguel Cárdenas-Montes. Time series decomposition for improving the forecasting performance of convolutional neural networks. In *Advances in Artificial Intelligence - 18th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2018, Granada, Spain, Proceedings*, volume 11160 of *Lecture Notes in Computer Science*, pages 87–97. Springer, 2018.

[10] Miguel Cárdenas-Montes and Iván Méndez-Jiménez. Ensemble deep learning for forecasting ^{222}Rn radiation level at Canfranc Underground Laboratory. In

14th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2019) - Seville, Spain, May 13-15, 2019, Proceedings, volume 950 of *Advances in Intelligent Systems and Computing*, pages 157–167. Springer, 2019.

[11] Miguel Cárdenas-Montes. Forecast daily air-pollution time series with deep learning. In *Hybrid Artificial Intelligent Systems - 14th International Conference, HAIS 2019, León, Spain, September 4-6, 2019, Proceedings*, volume 11734 of *Lecture Notes in Computer Science*, pages 431–443. Springer, 2019.

[12] Miguel Cárdenas-Montes. Uncertainty estimation in the forecasting of the ^{222}Rn radiation level time series at the Canfranc Underground Laboratory. *Logic Journal of the IGPL*, 11 2020. jzaa057.

[13] Roberto A. Vasco-Carofilis, Miguel A. Gutiérrez-Naranjo, and Miguel Cárdenas-Montes. PBIL for optimizing hyperparameters of convolutional neural networks and STL decomposition. In Enrique A. de la Cal, José Ramón Villar Flecha, Héctor Quintián, and Emilio Corchado, editors, *Hybrid Artificial Intelligent Systems - 15th International Conference, HAIS 2020, Gijón, Spain, November 11-13, 2020, Proceedings*, volume 12344 of *Lecture Notes in Computer Science*, pages 147–159. Springer, 2020.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] Y. LeCun. Generalization and network design strategies. Technical report, University of Toronto, 1989.

[16] John Cristian Borges Gamboa. Deep learning for time-series analysis. *CoRR*, abs/1701.01887, 2017.

[17] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. *CoRR*, abs/1611.06455, 2016.

[18] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[19] Salvador García, Daniel Molina, Manuel Lozano, and Francisco Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. *J. Heuristics*, 15(6):617–644, 2009.

[20] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. *Soft Comput.*, 13(10):959–977, 2009.

[21] Vittorio Mazzia, Francesco Salvetti, and Marcello Chiaberge. Efficient-capsnet: Capsule network with self-attention routing, 2021.

[22] Josef Hofbauer and Karl Sigmund. *Evolutionary Games and Population Dynamics*. Cambridge University Press, 1998.

[23] Martin A Nowak and Karl Sigmund. Evolutionary dynamics of biological games. *science*, 303(5659):793–799, 2004.

[24] Martin A Nowak. Five rules for the evolution of cooperation. *science*, 314(5805):1560–1563, 2006.

[25] Kevin Lai, Michal Feldman, Ion Stoica, and John Chuang. Incentives for cooperation in peer-to-peer networks. In *Workshop on economics of peer-to-peer systems*, pages 1243–1248, 2003.

[26] Matteo Cavaliere, Song Feng, Orkun S Soyer, and José I Jiménez. Cooperation in microbial communities and their biotechnological applications. *Environmental microbiology*, 19(8):2949–2963, 2017.

[27] Simon Levin. Crossing scales, crossing disciplines: collective motion and collective action in the global commons. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365(1537):13–18, 2010.

[28] Elizabeth Pennisi. How did cooperative behavior evolve? *Science*, 309(5731):93–93, 2005.

[29] Carlos P Roca, José A Cuesta, and Angel Sánchez. Evolutionary game theory: Temporal and spatial effects beyond replicator dynamics. *Physics of life reviews*, 6(4):208–249, 2009.

[30] Matjaž Perc and Attila Szolnoki. Coevolutionary games—a mini review. *BioSystems*, 99(2):109–125, 2010.

[31] Erez Lieberman, Christoph Hauert, and Martin A Nowak. Evolutionary dynamics on graphs. *Nature*, 433(7023):312–316, 2005.

[32] Corina E Tarnita, Tibor Antal, Hisashi Ohtsuki, and Martin A Nowak. Evolutionary dynamics in set structured populations. *Proceedings of the National Academy of Sciences*, 106(21):8601–8604, 2009.

[33] Matteo Cavaliere, Sean Sedwards, Corina E Tarnita, Martin A Nowak, and Attila Csikász-Nagy. Prosperity is associated with instability in dynamical networks. *Journal of theoretical biology*, 299:126–138, 2012.

[34] Martin A Nowak, Corina E Tarnita, and Tibor Antal. Evolutionary dynamics in structured populations. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365(1537):19–30, 2010.

[35] Hisashi Ohtsuki, Martin A Nowak, and Jorge M Pacheco. Breaking the symmetry between interaction and replacement in evolutionary dynamics on graphs. *Physical review letters*, 98(10):108106, 2007.

[36] Christoph Adami, Jory Schossau, and Arend Hintze. Evolutionary game theory using agent-based methods. *Physics of life reviews*, 19:1–26, 2016.

[37] Paolo Cazzaniga, Marian Gheorghe, Natalio Krasnogor, Giancarlo Mauri, Pario Pesciani, and Frnacisco José Romero-Campero. Probabilistic/stochastic models. In Păun et al. [69], pages 455 – 474.

[38] Mónica Cardona, M. Ángels Colomer, Antoni Margalida, Antoni Palau, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, and Delfí Sanuy. A computational modeling for real ecosystems based on P systems. *Natural Computing*, 10(1):39–53, 2011.

[39] Maria Àngels Colomer, Antoni Margalida, and Mario J Pérez-Jiménez. Population dynamics p system (pdp) models: a standardized protocol for describing and applying novel bio-inspired computing tools. *PloS one*, 8(4):e60698, 2013.

[40] Ignacio Pérez-Hurtado, Luis Valencia-Cabrera, Mario J. Pérez-Jiménez, Maria Angels Colomer, and Agustin Riscos-Núñez. Mecosim: A general purpose software tool for simulating biological phenomena by means of P systems. In *Fifth International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2010, University of Hunan, Liverpool Hope University, Liverpool, United Kingdom / Changsha, China, September 8-10 and September 23-26, 2010*, pages 637–643. IEEE, 2010.

[41] Gheorghe Păun. Computing with membranes. Technical Report 208, Turku Centre for Computer Science, Turku, Finland, November 1998.

[42] Mihai Ionescu, Gheorghe Păun, and Takashi Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2-3):279–308, 2006.

[43] Ren Tristan A. de la Cruz, Francis George C. Cabarle, Ivan Cedric H. Macababayao, Henry N. Adorna, and Xiangxiang Zeng. Homogeneous spiking neural P systems with structural plasticity. *J. Membr. Comput.*, 3(1):10–21, 2021.

[44] Qianqian Ren and Xiyu Liu. Delayed spiking neural p systems with scheduled rules. *Complexity*, 2021:1–13, 2021.

[45] Xiaoxiao Song, Luis Valencia-Cabrera, Hong Peng, and Jun Wang. Spiking neural p systems with autapses. *Information Sciences*, 570:383–402, 2021.

[46] Linqiang Pan, Gheorghe Paun, Gexiang Zhang, and Ferrante Neri. Spiking neural p systems with communication on request. *International Journal of Neural Systems*, 27, 08 2017.

[47] Tingfang Wu, Florin-Daniel Bîlbîe, Andrei Paun, Linqiang Pan, and Ferrante Neri. Simplified and yet turing universal spiking neural p systems with communication on request. *International Journal of Neural Systems*, 28, 04 2018.

[48] Gexiang Zhang, Haina Rong, Ferrante Neri, and Mario Pérez-Jiménez. An optimization spiking neural p system for approximately solving combinatorial optimization problems. *International journal of neural systems*, 24:1440006, 08 2014.

[49] Ming Zhu, Qiang Yang, Jianping Dong, Gexiang Zhang, Xiantai Gou, Haina Rong, Prithwineel Paul, and Ferrante Neri. An adaptive optimization spiking neural p system for binary problems. *International Journal of Neural Systems*, 31, 06 2020.

[50] Gexiang Zhang, Haina Rong, Prithwineel Paul, Yang yang He, Ferrante Neri, and Mario J. Pérez-Jiménez. A complete arithmetic calculator constructed from spiking neural p systems and its application to information fusion. *International journal of neural systems*, page 2050055, 2021.

[51] P-Lingua. `http://www.p-lingua.org/wiki/index.php/Main_Page`.

[52] P-Lingua. `http://www.p-lingua.org/mecosim/`.

[53] Martin A Nowak. *Evolutionary dynamics: exploring the equations of life*. Harvard university press, 2006.

[54] Rui Cong, Bin Wu, Yuanying Qiu, and Long Wang. Evolution of cooperation driven by reputation-based migration. *PLoS One*, 7(5):e35776, 2012.

[55] Luo-Luo Jiang, Wen-Xu Wang, Ying-Cheng Lai, and Bing-Hong Wang. Role of adaptive migration in promoting cooperation in spatial games. *Physical Review E*, 81(3):036108, 2010.

[56] Genki Ichinose and Takaya Arita. The role of migration and founder effect for the evolution of cooperation in a multilevel selection context. *Ecological Modelling*, 210(3):221–230, 2008.

[57] Dirk Helbing and Wenjian Yu. Migration as a mechanism to promote cooperation. *Advances in Complex Systems*, 11(04):641–652, 2008.

[58] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[59] R. B. Cleveland, W. S. Cleveland, J.E. McRae, and I. Terpenning. STL: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, pages 3–73, 1990.

[60] Josh Montague. STLDecompose. `https://github.com/jrmontag/STLDecompose`, 2017.

[61] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[62] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 2960–2968, 2012.

[63] Dougal Maclaurin, David K. Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2113–2122, 2015.

[64] Gonzalo I. Diaz, Achille Fokoue-Nkoutche, Giacomo Nannicini, and Horst Samulowitz. An effective algorithm for hyperparameter optimization of neural networks. *IBM Journal of Research and Development*, 61(4):9, 2017.

[65] Elad Hazan, Adam R. Klivans, and Yang Yuan. Hyperparameter optimization: A spectral approach. *CoRR*, abs/1706.00764, 2017.

[66] Risto Miikkulainen, Jason Zhi Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. *CoRR*, abs/1703.00548, 2017.

[67] Ang Li, Ola Spyra, Sagi Perel, Valentin Dalibard, Max Jaderberg, Chenjie Gu, David Budden, Tim Harley, and Pramod Gupta. A generalized framework for population based training. *CoRR*, abs/1902.01894, 2019.

[68] Reza Rastegar and Arash Hariri. The population-based incremental learning algorithm converges to local optima. *Neurocomputing*, 69(13-15):1772–1775, 2006.

[69] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Oxford, England, 2010.

[70] Daniel Díaz-Pernil, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, and Agustin Riscos-Núñez. A p-lingua programming environment for membrane computing. In David W. Corne, Pierluigi Frisco, Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing - 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers*, volume 5391 of *Lecture Notes in Computer Science*, pages 187–203. Springer, 2008.