Computers and Mathematics with Applications 66 (2013) 201–212

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Playing with complexity: From cellular evolutionary algorithms with coalitions to self-organizing maps

Juan C. Burguillo [*]

Telematics Engineering Department, University of Vigo, 36310-Vigo, Spain

ARTICLE INFO

Keywords:
Game theory
Coalitions
Cellular evolutionary algorithms
Self organized networks (SOM)
Optimization
Prediction

ABSTRACT

Since its origins, Cellular Automata (CA) has been used to model many type of physical and computational phenomena. Interacting CAs in spatial lattices combined with evolutionary game theory have been very popular for modeling genetics or behavior in biological systems. Cellular Evolutionary Algorithms (cEAs) are a kind of evolutionary algorithm (EA) with decentralized population in which interactions among individuals are restricted to the closest ones. The use of decentralized populations in EAs allows to keep the population diversity for longer, usually resulting in a better exploration of the search space and, therefore in a better performance of the algorithm. A new adaptive technique (EACO) based on Cellular Automata, Game Theory and Coalitions uses dynamic neighborhoods to enhance the quality of cEAs. In this article we compare the characteristics EACO with classical Self-organizing Maps (SOM), and we discuss the possibilities for using Game Theory and Coalitions in the SOM scenario.

1. Introduction

A Cellular Automata (CA) is a regular grid of cells, or lattice, each one having a finite number of states. CAs have been used as models for describing physical and computational phenomena with many applications [1] in physics, computational theory, mathematics, complexity theory, urban traffic modeling, artificial intelligence, biology, etc. Every cell, also denoted as cellular automaton, has a defined neighborhood to interact with. Time is discrete, and in every iteration any cell interacts with its neighborhood to find its new state depending on its own state and its neighbors' state.

Game Theory [2] provides useful mathematical tools to understand the possible strategies that self-interested agents may follow when choosing a course of action. Evolutionary Game Theory (EGT) [3] models the application of interaction dependent strategies in populations along generations. EGT differs from classical game theory by focusing on the dynamics of strategy change more than the properties of strategy equilibrium. In EGT participants do not possess unfailing Bayesian rationality. Instead, they play with limited computing and memory resources. The only requirement is that the players learn by trial and error, incorporate what they learn in their future behavior, and die or somehow 'change' if they do not.

Evolutionary algorithms (EA) are well-known population based metaheuristics [4]. They work on a set of solutions (called *population*), evolving them simultaneously towards (hopefully) better ones by applying some stochastic operators (typically called *evolutionary operators*, e.g., selection, recombination, and mutation). Cellular EAs (cEAs) [5] are structured population algorithms with a high explorative capacity. The individuals composing their population are arranged into a (usually) two dimensional toroidal mesh, and only neighbor individuals are allowed to interact during the breeding loop. Structuring the

[*] Tel.: +34 986 813869; fax: +34 986 812116.
E-mail address: J.C.Burguillo@uvigo.es.

0898-1221/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2013.01.020

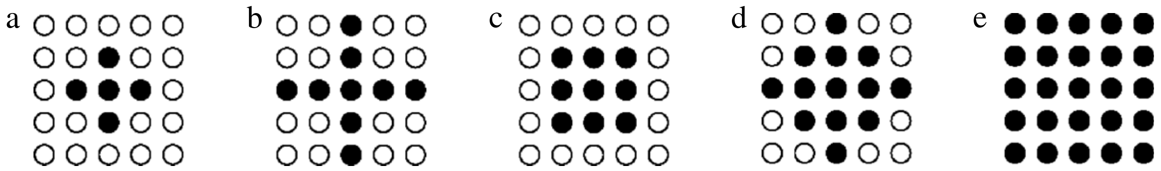

Fig. 1. Five possible neighborhoods used in CA. Among them, we can see von Neumann's neighborhood in (a) and Moore's neighborhood in (c).

population in this way we can achieve a good exploration/exploitation trade off on the search space, thus improving the capacity of the algorithm for solving complex problems [6].

This article first introduces the areas of Cellular Automata, Evolutionary Game Theory, Coalitions and Cellular Evolutionary Algorithms applied to function optimization. For this, we describe the EACO algorithm that integrates all these topics in a way to obtain a synergy in the development of Evolutionary Algorithms, with the advantages of different classical distributed models, but avoiding some of their limitations. This can be done thanks to the use of spatial cellular approaches with neighborhoods, allowing the formation of coalitions among cells as a way to create dynamic islands of evolution in order to preserve diversity. Besides, EACO uses Game Theory to consider every cell as a player of a game arranged in a two dimensional torus. Cells will be able to evolve depending on their payoff with respect to their neighbors, and have also the support provided by their coalition members. This approach allows the payoff of a given solution to be defined in terms of how much such solution has improved in the last generations. The idea is to speed up the evolution of individuals grouped in high-quality coalitions that are quickly converging to promising solutions. Once EACO has been introduced, we do so with Self-organizing Maps (SOM), a type of self-organizing neural networks with many applications described along the last two decades [7]. Finally, we compare both approaches to explore the possibilities of merging both techniques to get a richer and eclectic combination to apply in the areas of optimization and prediction.

Therefore, the main contribution of this work is: first to compare these two literature independent models (cEAs and SOM) to evaluate their similarities; and second to consider the possibility of using game theory and coalitions, as dynamic neighborhoods in SOM networks.

The next sections are structured as follows. Section 2 introduces Cellular Automata. Section 3 provides a short introduction to Game Theory and Coalitions. Then, Section 4 introduces Evolutionary Algorithms (EAs), Cellular Evolutionary Algorithms (cEAs) and describes a recent cEA with Coalitions (EACO) and its promising results. Section 5 describes the main characteristics of Self-organizing Maps (SOM), and afterwards Section 6 discusses the promising possibilities of using game theory and coalitions in SOM models. Finally Section 7 outlines some conclusions and future work.

2. Cellular automata

A Cellular Automata (CA) is a regular grid of cells, or lattice, each one having a finite number of states [8]. CAs have been used as models for describing physical and computational phenomena with many applications in physics, computational theory, mathematics, complexity theory, urban traffic, biology, etc. Every cell, also denoted as cellular automaton, has a defined neighborhood to interact with. Time is discrete, and in every iteration any cell interacts with its neighborhood to find its new state depending on its own state and its neighbors' state.

CAs are simulated by a finite grid, which can be a line in one dimension, or a rectangle in 2D. In 2D usually cells are represented by squares, but triangles and hexagons can also be used. In these domains neighborhoods are usually based on the Euclidean distance, but other conventions can be considered as stochastic distance functions, social networks and many more. The same happens with the rules that define the transition function from one state to the next, where many types of functions can be defined.

The CA designer has to define the behavior of the cells in the frontier and, to solve this, several alternatives have been usually applied:

- *Open frontier*: the cells outside the lattice have a constant state.
- *Periodic frontier*: the frontiers of the lattice are in contact, i.e., a circumference in 1D, a toroidal shape in 2D, etc.
- *Reflection frontier*: outside cells take the value of their internal mirror cells.
- *No frontier*: the frontier expands when needed.

The origins of CA go back to the 40s when John von Neumann desired to design a machine able to self-reproduce and to provide complex computation. Together with his colleague Stanislaw Ulam, von Neumann developed a theory of CAs considering a model in 2D where cells can be in 29 possible states. In this model, the neighborhood defined for any cell was the cell itself plus the four orthogonal cells around, i.e., above, below, right and left; known as von Neumann's neighborhood (see Fig. 1). The result was a universal copier and constructor for a particular initial configuration [8].

In 1970 Conway produced the most popular CA, the Game of Life, which was published by Martin Gardner at Scientific American [9]. Life is a 2D CA game, where the neighborhood of a cell is composed by the cell itself and the 8 ones surrounding it, known as Moore's neighborhood (see Fig. 1). In Life cells can be dead or alive (2 states) and evolve using 3 very

current pattern	111	110	101	100	011	010	001	000
new middle state	0	1	1	0	1	1	1	0

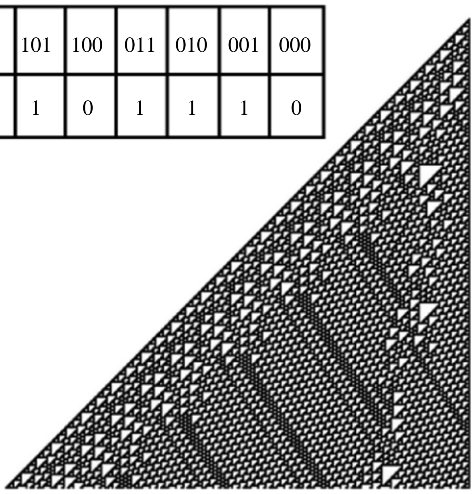

Fig. 2. One-dimensional CA evolution using rule 110.

simple rules:

1. *Birth*: a dead cell gets back to live if it has exactly 3 neighbors.
2. *Death*: an alive cell dies if it has less than 2 cells around (isolation) or more than 3 (overpopulation).
3. *Survive*: alive cells stay alive with 2 or 3 cells around.

Life can be used for universal computation, i.e., with an initial appropriate distribution of cells, Life becomes a Turing Machine [10].

Moving forward in time, in the 80s Stephen Wolfram systematically investigates a very basic, but essentially unknown class of cellular automata, which he denotes as elementary cellular automata [11]. The unexpected complexity in the behavior of these simple models led Wolfram to consider if complexity in Nature may be due to similar mechanisms. In 2002 Wolfram published a book titled 'A New Kind of Science' [1], which extensively argues that the discoveries about cellular automata are not only mathematical or isolated facts, but are robust and have significance for all disciplines of science.

Wolfram defines four classes into which cellular automata, and several other simple computational models, can be divided depending on their behavior. These four classes are:

1. *Class* 1 (*Stable*): patterns evolve quickly into stable and homogeneous final states.
2. *Class* 2 (*Oscillating*): patterns evolve quickly into oscillating structures.
3. *Class* 3 (*Chaos*): patterns evolve in a pseudo-random or chaotic manner.
4. *Class* 4 (*Complexity*): patterns evolve into stable and repetitive structures that interact in very complicated ways. These types of patterns have been usually denoted as computation at the Edge of Chaos [12], as they are not ordered and not chaotic.

As an example of an elementary cellular automata investigated by Wolfram, consider a one-dimensional CA, with two possible states per cell, and two adjacent cells behaving as neighbors. Then, a cell and its two neighbors form a neighborhood of 3 cells, so there are $2^3 = 8$ possible patterns for a neighborhood (see table in Fig. 2). In this model, a rule consists of deciding, for each pattern, whether the center cell will be a 1 or a 0 in the next generation. Thus, there are then $2^8 = 256$ possible rules. These 256 CAs are generally denoted by their Wolfram code, a naming convention proposed by Wolfram that gives each rule a number from 0 to 255. Among them, rule 110 is particularly interesting. Fig. 2 shows the time evolution of rule 110, when the starting configuration of the one-dimensional CA consists of a 1 (at the top of the image) surrounded by 0's. In the figure each row of pixels represents a generation in the history of the automaton, being $t = 0$ the top row, and each pixel is colored white for 0 and black for 1. Rule 110 exhibits class 4 complex behavior. Wolfram has conjectured that many, if not all, class 4 cellular automata are capable of universal computation. This has been proven for Conway's Game of Life and for rule 110 by Matthew Cook, a research assistant to Wolfram [13].

3. Game theory

Game Theory [2] provides useful mathematical tools to understand the possible strategies that self-interested players may follow when competing or collaborating in games. This branch of applied mathematics is used nowadays in the social sciences (mainly economics), biology, engineering, political science, international relations, computer science, philosophy, etc.

The roots and popularity of game theory trace back to the 50s, when von Neumann and Morgenstern analyze competitions in which one individual does better at another's expense: zero sum games [14]. From that moment, traditional applications of game theory attempt to find equilibria in these games. In any equilibrium each player of the game adopts a strategy that is unlikely to change. Many equilibrium concepts have been developed, among them we find the famous Nash equilibrium [15].

Another approach which has been very popular and interesting is the combination of Evolutionary Techniques with Game Theory, which gave birth to Evolutionary Game Theory (EGT) [3]. EGT models the application of interaction dependent strategies in populations along generations, and differs from classical game theory by focusing on the dynamics of strategy change more than the properties of strategy equilibrium. In evolutionary games, participants do not possess unfailing Bayesian rationality, instead they play with limited resources. The only requirement is that the players learn by trial and error, incorporate what they learn in future behavior, and die or somehow change if they do not. An interesting set of spatial and social evolutionary games, based on the iterative version of the Prisoner's Dilemma [16] had been suggested and deeply analyzed by Nowak and other authors [17,18] trying to understand the role of local or social interactions in the maintenance of cooperation.

Presently, we can consider two main branches in game theory: non cooperative and cooperative. On the one hand, non cooperative game theory, or competitive games [19], assumes that each participant acts independently, without collaboration with the others; and it chooses its strategy for improving its own benefit. On the other hand, cooperative game theory studies the behavior of players when they cooperate. Within cooperative games, we find coalition games, in which a set of players seek to form cooperative groups to improve their performance. Coalitions enable players to accomplish goals they may not accomplish independently.

Coalitions usually emerge as a natural way to achieve better conditions for defending its members against the outside players. Game theory provides a natural framework to analyze the partitions that can be formed in a multiplayer game, and the relative power or influence they can achieve over the whole community. Unfortunately, many times coalition formation can become intractable since all possible coalitions combinations in a game depend exponentially on the number of players. Therefore, finding the optimal partition by checking the whole space may be too expensive from a computational point of view. In the literature, authors have applied evolutionary techniques to try to avoid the coalition formation problem [20]. Other recent approaches use local decisions to create dynamic coalitions that emerge and evolve adapting to the present needs of every player [21].

4. Cellular evolutionary algorithms

Evolutionary algorithms (EA) are well-known population based metaheuristics [4] that work on a set of solutions denoted as the *population*. EAs evolve such a population towards a better one, from an evolutive point of view, by applying some *evolutionary operators*, usually selection, recombination, and mutation. Most EAs use a single population (*panmixia*) of individuals and apply the operators over it as a whole. However, when dealing with complex problems, EAs perform a fast convergence, but sometimes getting stuck in local optimal solutions. In order to avoid this problem, it is usual to decentralize the population, keeping the diversity of solutions for longer [6]. There are two main ways for decentralizing the population in EAs: distributed EAs (dEAs, also denoted as island models) [6] and cellular EAs (cEAs) [5].

On the one hand, in dEAs the population is split into several smaller sub-populations that are independently evolved by EAs, exchanging some information (typically the best solution found so far) with other sub-populations. Thus, the EAs in the different islands will perform a fast convergence to hopefully distinct regions of the search space, preserving this way the overall population diversity. The information exchange among islands allows them to benefit from the exploration performed by the others and to introduce diversity in the local sub-populations.

Cellular EAs [5] are structured population algorithms with a high explorative capacity. The individuals composing their population are arranged into a (usually) two dimensional toroidal mesh, and only neighbor individuals are allowed to interact during the breeding loop (see Fig. 3). This way, we are introducing some kind of isolation in the population that depends on the distance between individuals. Hence, the genetic information of a given individual can be spread slowly through the grid (since neighborhoods are overlapped), and it will need a certain number of generations to reach distant individuals (thus preventing the population from premature convergence). Structuring the population in this way we can achieve a good exploration/exploitation trade off on the search space, thus improving the capacity of the algorithm for solving complex problems [6].

4.1. Canonical cellular evolutionary algorithms

A canonical cEA follows the pseudo-code included in Algorithm 1. In this basic cEA, the population is usually structured in a regular grid of d dimensions, and a neighborhood is defined on it. The algorithm iteratively considers each individual in the grid (line 3), and individuals may only interact with the ones belonging to their neighborhood (line 4), so parents are chosen among the neighbors (line 5) with a given criterion. Crossover and mutation operators are applied to the individuals in lines 6 and 7, with probabilities P_c and P_m, respectively. Afterwards, the algorithm computes the fitness value of the new offspring individual (or individuals) (line 8), and inserts it (or one of them) instead of the current individual in the population (line 9) following a given replacement policy. This loop is repeated until a termination condition is met (line 2).

The cEA described in Algorithm 1 is asynchronous, since the population is updated with the next generation individuals just after creating them. This way, these new individuals can interact with those belonging to their parent's generation.

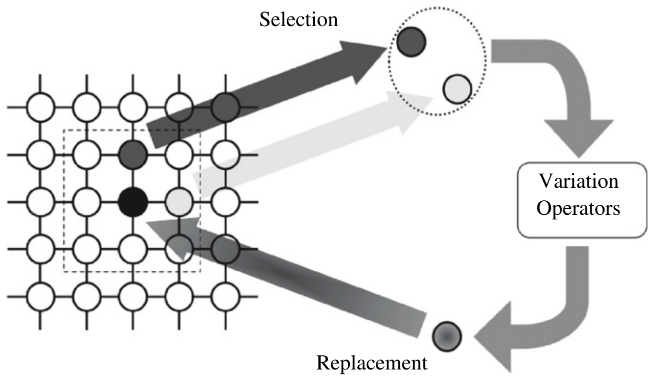

Fig. 3. In cellular EAs, individuals are only allowed to interact with their neighbors.

On the contrary, there is also the possibility of storing all the offspring individuals in an auxiliary population, and then replacing all the individuals in the population at the same time. This last version belongs to the synchronous cEA model. As it was studied in [5], the use of asynchronous policies allows faster convergence of the population than in the case of the synchronous one.

Algorithm 1 Pseudocode for a canonical cEA

```
1: //Algorithm parameters in 'cea'
2: while ! StopCondition() do
3:    for individual ← 1 to cea.popSize do
4:       n_list←Get_Neighborhood(cea,position(individual));
5:       parents←Selection(n_list);
6:       offspring←Recombination(cea.Pc,parents);
7:       offspring←Mutation(cea.Pm,offspring);
8:      Evaluation(offspring);
9:      Add(position(individual),offspring,cea);
10:   end for
11: end while
```

4.2. *Evolutionary algorithms with coalitions*

The work presented in [22] describes a new EA with Coalitions (EACO), designed on a dynamic topology that is taking advantage of both cellular and island population models. It is basically a cellular EA in which coalitions of individuals are automatically formed and maintained. Individuals in the population will choose a coalition to join at every time in terms of some rewards they can expect from such coalition. Neighborhoods are defined by these coalitions, and coalitions can be understood as small panmictic subpopulations (i.e., islands) that emerge in the cellular topology providing dynamic neighborhoods.

4.2.1. *EACO description*

EACO merges recent approaches in the areas of Cellular Automata, Evolutionary Game Theory and Coalitions. Mainly it integrates all these topics in a way to obtain a synergy in the development of Evolutionary Algorithms, with the advantages of dEAs and cEAs. At the same time, EACO removes some classical required parameters, like the fixed neighborhoods, or the islands connectivity topology, the migration frequency, or the policies to exchange and discard individuals (in dEAs). This is achieved thanks to the use of spatial cellular approaches with dynamic neighborhoods, that allows the formation of coalitions among cells as a way to create islands of evolution in order to preserve diversity. Besides, it uses Game Theory to consider every cell as a player of a game arranged in a two dimensional toroidal grid. Cells evolve depending on their payoff with respect to their neighbors, and have also the support provided by their coalition members. This approach allows the payoff of a given solution to be defined in terms of how much such a solution has improved in the last generations. The idea is to speed up the evolution of individuals grouped in high-quality coalitions that are quickly converging to promising solutions.

Individuals can only belong to one single coalition, and they can join to other coalitions from which they expect to get a maximum profit. In the same way, they can leave their current coalition if the benefit of belonging to it is low. When an individual leaves a coalition, it can either join another existing one or create a completely new one, depending on the expected gain. Therefore, individuals are considered to be selfish entities that are able to collaborate with others to look for benefits.

All the individuals belonging to the same coalition can interact among them, like in the subpopulations of an island EA. In order to further evolve to better solutions, individuals will be interested in mating with diverse solutions of good quality. Therefore, it would be beneficial for them to belong to coalitions composed by a large number of high quality individuals, representing a diverse set of solutions at the same time. The quality of a coalition is consequently evaluated in terms of its size, the average quality of the solutions forming it, and their diversity. This way, belonging to a high quality coalition will generally be beneficial for individuals.

Inside a coalition, the evolution is performed as in regular EAs with panmictic populations; typically, the parents are selected from the population using some selection scheme, and then some variation operators are applied to them in order to generate a set of offspring solutions that are inserted into the population following some policy.

Every coalition (C_i) has a valuation, which is determined as follows,

$$\text{Valuation } (C_i) = \alpha \cdot \text{Size}(C_i) + \beta \cdot \text{Var}(C_i) + \gamma \cdot \text{Avg}(C_i), \tag{1}$$

where $\alpha + \beta + \gamma = 1$ and these coefficients model the weight given to the size of the coalitions, the variance and the average quality of the solutions, respectively. This valuation will be used by cells to evaluate the quality of a coalition and to move from one coalition to another.

Algorithm 2 Pseudocode for EACO

```
1: //Algorithm parameters in 'eaco'
2: while ! StopCondition() do
3:    coa_value_list ← Coalition_Valuation(coa_list);
4:    for individual ← 1 to eaco.popSize do
5:       n_list ← Get_Coalition_Neighborhood(eaco,position(individual));
6:       parents ← Selection(n_list);
7:       offspring ← Recombination(eaco.Pc,parents);
8:       offspring ← Mutation(eaco.Pm,offspring);
9:       Add(position(individual),offspring,eaco);
10:      coa_individuals ← Change_Coalition(eaco,position(individual),coa_value_list, $Ind_c$);
11:   end for
12: end while
```

Algorithm 2 describes the pseudocode of the EACO algorithm, which is relatively similar to the one described before for the canonical cEA. In EACO, as happens in cEA, the population is structured in a regular grid of d dimensions, and a neighborhood is defined on it. The algorithm first evaluates the valuation of every coalition (line 3). At the beginning, all the cells are independent, so their valuation differences will be based only in the quality of those cell solutions. Then, the algorithm iteratively considers as current each individual in the grid (line 4), and individuals may only interact with the ones belonging to their von Neumann neighborhood (line 5), in the independent case, or within the coalition neighborhood, i.e., its member cells. Then, parents are chosen among those neighbors (line 6) with a given criterion. Crossover and mutation operators are also applied to the individuals in lines 7 and 8, with probabilities P_c and P_m, respectively. Afterwards, the algorithm computes the fitness value of the new offspring individual (or individuals) (line 9), and inserts it (or one of them) instead of the current individual in the population (line 9) following a given replacement policy. Finally, the cell can change from one coalition to another (line 10) depending on its present valuation and the independence coefficient (Ind_c). This loop is repeated until a termination condition is met (line 2).

4.2.2. EACO results

Here we present the results obtained over a set of problems chosen for a study detailed in [22]. The election of this benchmark is justified by both their difficulty and their application domains (combinatorial optimization, continuous optimization, telecommunications, scheduling, etc.). The benchmark is representative because it contains many different interesting features in optimization, such as epistasis, multimodality, deceptiveness, use of constraints, parameter identification, and problem generators. These are important ingredients in any work trying to evaluate algorithmic approaches with the objective of getting reliable results, as stated in [23].

Table 1 presents the results as a table obtained in the experiments, which compare the performance of EACO versus the canonical cellular GA, which was demonstrated to outperform other GAs with panmictic and decentralized populations in [5]. The results were obtained after performing 100 independent runs of every algorithm for all the problems. We did an exhaustive evaluation of the EACO algorithm, and we heuristically found that mainly it works better using: $Ind_c = 0.05$, $\alpha = 0.1$, $\beta = 0.1$ and $\gamma = 0.8$. This means that the coefficient with bigger weight in the valuation is the average functional value of the coalition.

For the two algorithms, in the table we show the best result found (or the percentage of runs in which the optimum was found), the average best solution found (unless the optimum is found in every run), and the average number of generations required to find the optimum. The best results are emphasized in **bold font**. In the last two columns, we present the results of the Wilcoxon unpaired signed rank sum statistical test in the comparison of both algorithms for the

Table 1
EACO computational results. Simbol (__) means that the optimum has been found in all the executions.

Problem	cGA			EACO			Wilcoxon test	
	Best	Avg. Result	Avg. Gen.	Best	Avg. Result	Avg. Gen.	Results	Gen.
ECC	**100.0%**	–	$1.3103E2$ $\pm 2.3662E1$	**100.0%**	–	**1.1256E2** **±3.4095E1**	–	**3.858E − 8**
MAXCUT100	(52.0%)	$4.6656E-4$ $\pm 8.6721E-6$	**1.9892E2** **±1.9213E2**	(**78.0**%)	**2.1297E − 4** **±9.2710E−6**	$3.5442E2$ $\pm 2.6644E2$	**3.162E − 2**	0.1631
MAXCUT20_01	**100.0%**	–	7.8700 ±3.0966	**100.0%**	–	**5.0600** **±1.8793**	–	**2.335E − 11**
MAXCUT20_09	**100.0%**	–	$1.3530E1$ ± 4.5002	**100.0%**	–	**8.3700** **±3.1096**	–	**2.2E − 16**
MMDP	(**2.0**%)	**2.5484E − 2** **±0.0**	**3.6800E0** **±3.2527E1**	$2.5227E-2$ (0.0%)	$2.6026E-2$ ± 0.0	– –	0.6894	–
MTTP100	**100.0%**	–	**1.3733E2** **±1.9499E1**	**100.0%**	–	$1.7657E2$ $\pm 5.4288E1$	–	**3.774E − 10**
MTTP200	(**88.0**%)	**6.0900E1** **±0.0**	$2.5469E2$ $\pm 3.2804E1$	(46.0%)	$2.7156E2$ ± 0.0	**1.5520E2** **8.5397E1**	0.4963	**2.194E − 3**
P-PEAKS	**100.0%**	–	$5.7400E1$ ± 4.2545	**100.0%**	–	**4.2210E1** **±5.0578**	–	**2.2E − 16**

results found and the number of generations required. Those values lower than 0.05 (emphasized in **bold font in the table**) mean that differences between the algorithms are significant with 95% confidence. Those cases when EACO is significantly outperforming cGA according to this test are emphasized with dark gray background in Table 1, while light gray background stands for significantly better behavior of cGA.

We can see in the table that the two algorithms find always the optimal solution for ECC, MAXCUT20_01, MAXCUT20_09, MTTP100, and P-PEAKS. For the other 3 problems, EACO is more effective for MAXCUT100, but worse for the other two ones. However, if we check the average best solutions reported by the algorithms in the 100 independent runs, we can see that EACO is significantly more accurate than cGA for MAXCUT100, and there are no statistically significant differences for the other two problems.

Regarding the number of generations required to find the optimum, EACO is faster than cGA with statistical significance for most problems. The exceptions are MAXCUT100 and MMDP, for which cGA is statistically faster.

Summarizing the results, the new EACO algorithm clearly outperforms the cGA in terms of efficiency (i.e., the number of generations required to find the optimum) and also in accuracy (i.e., the quality of solutions when the optimum is not found), while the two algorithms perform similarly in terms of efficacy (i.e., the percentage of runs in which the optimum is found).

5. Self-organizing maps (SOM)

Self-organizing Maps (SOM) [7], also denoted as Self-organizing Feature Maps (SOFM) or Kohonen Neural Networks, derive from the seminal work done by Professor Teuvo Kohonen in auto-associative memories during the 70s and early 80s. The basic idea of the SOM is to make available a certain amount of classificatory resources, usually denoted as neurons, cells or units, which self-organize based on input patterns. The concept of organizing information spatially constitutes a differential element between SOM and other type of neural networks, and it is believed to be one of the learning paradigms used by the human brain.

From a topological point of view, a SOM is a single layer neural network, where the neurons are set along a d-dimensional grid. In most applications this grid is 2-dimensional and rectangular, but hexagonal grids or other dimensional spaces have been also used in the literature. Associated with each neuron is a weight vector of the same dimension as the input vectors (patterns) and a position in the map space (see Fig. 4). The self-organizing map describes a mapping from a higher dimensional input space to a lower dimensional map space. The procedure for placing a vector from the input space onto the map is to find the neuron with the closest (smallest distance metric) weight vector. After several iterations, the result is an ordered network in which nearby neurons will share certain similarities. Therefore similar input patterns activate similar areas in the SOM producing a local specialization in the global self-organized network.

Before training, the neurons may be initialized randomly, and like most artificial neural networks, SOMs operate in two modes: training and mapping. *Training* builds the map using input examples (a competitive process, also called vector quantization), while *Mapping* automatically classifies a new input vector.

Fig. 5 describes a 2-dimensional to 1-dimensional mapping presented to visualize the training process [24]. In this problem, 2-dimensional input data points are uniformly distributed in a triangle, and a 1-dimensional SOM is trained with these patterns. Fig. 5 represents the evolution of the neurons in the input space. As training proceeds, the line first unfolds (steps 1 to 100), and then fine-tunes itself to cover the input space.

We can think about a SOM as a surface that is stretched and bent all over the input space (see Fig. 6) in a process that is achieved iteratively over a large number of epochs. In this sense, a SOM is similar to the input layer of a RBF (Radial Basis Function) neural net, a neural gas model, or a K-means algorithm [7]. The big difference is that while in these methods

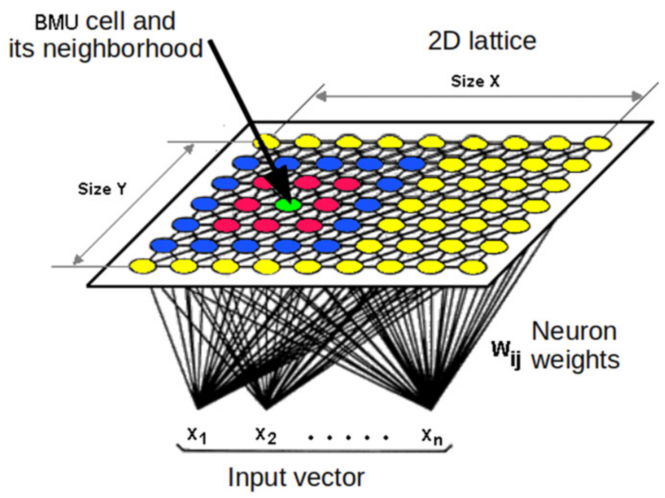

Fig. 4. A SOM map with an input vector, a winner cell and its neighborhood.

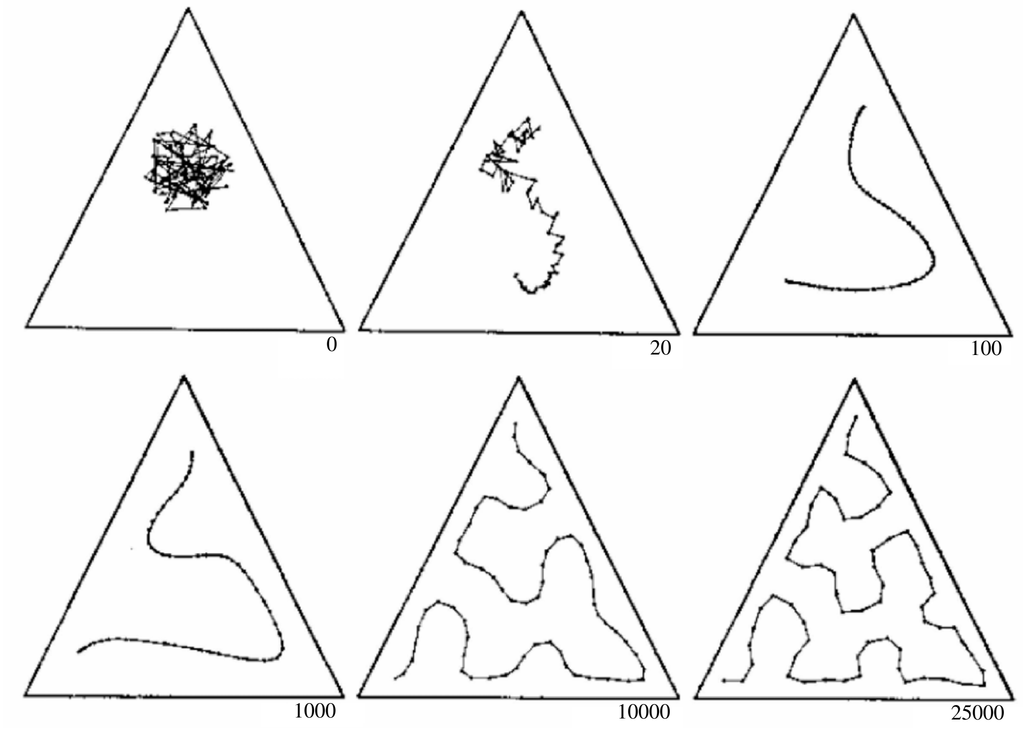

Fig. 5. 2D to 1D mapping by a SOM [24].

there is no notion of output space neighborhood (all neurons are independent from each other), in SOM the neurons are tied together in the output mapping space.

5.1. *Formal definition*

The SOM learns from the input patterns a mapping from a high-dimensional continuous input space X onto a low-dimensional discrete space L (the lattice) of m neurons which are arranged in fixed topological forms, e.g., as a rectangular 2-dimensional array. The number of neurons m in the map is usually defined after experimentation with the dataset. However, a useful rule-of-thumb is to set it to the highest m fulfilling $\left(2^m < \sqrt{v}\right)$, where v is the number of input vectors available [25].

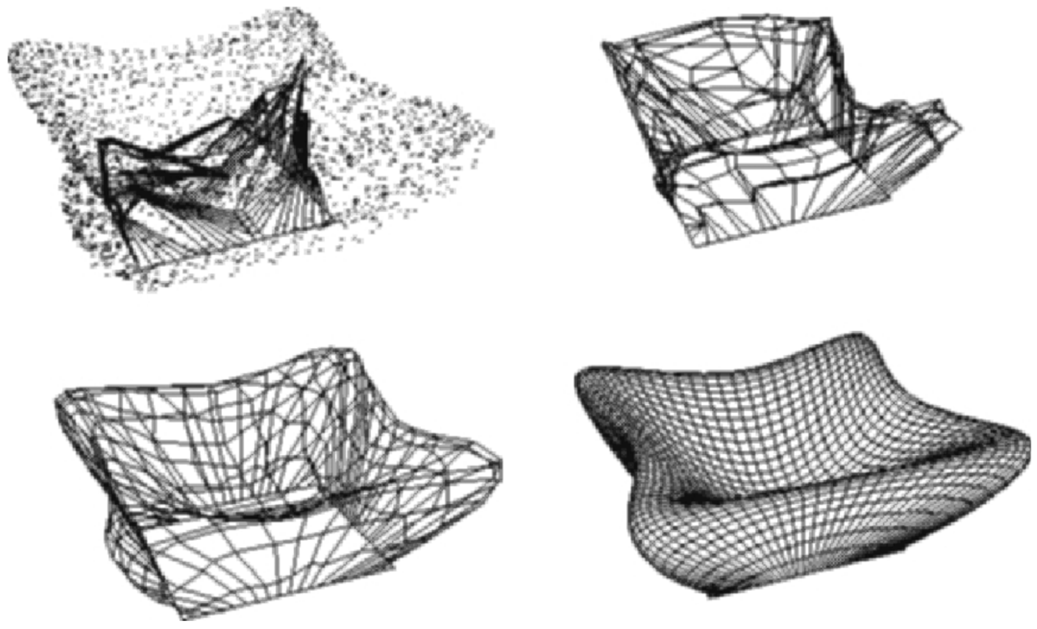
Fig. 6. Several shapes obtained after SOM training.

The function map $b(x) : X \to L$ is defined by assigning to the current input vector $x(t) \in X \subset \mathbb{R}^n$ a neuron index b in the map obtained by:

$$b(x(t)) = \text{argmin}\|x(t) - w_i(t)\|, \quad \forall i \in \{1, \ldots, m\} \tag{2}$$

where $\| \cdot \|$ denotes the Euclidean distance, n is the input vector dimension and t is the discrete time step associated with the iterations of the algorithm. The weight vectors $w_i(t)$ in the map are trained according to a competitive–cooperative learning rule, where the weight vector of a winning neuron with index b (usually denoted as Best Matching Unit, BMU) and its neighbors in the output map array are updated with this formula:

$$w_i(t+1) = w_i(t) + \alpha(t)h(b, i, t)[x(t) - w_i(t)], \quad i = 1, \ldots, m \tag{3}$$

where $0 < \alpha(t) < 1$ is the learning rate and $h(b, i, t)$ is a weighting function which limits the neighborhood of the BMU. This neighborhood function assumes values in [0, 1] and is high for neurons that are close to the BMU, and small (or 0) for neurons far away. The neighborhood radio and $\alpha(t)$ should both decay with t to guarantee convergence of the weight vectors in the map to stable steady states.

5.2. The error function

Generally, there is always some difference between any given input pattern and the neuron it is mapped to. We refer to this difference as the quantization error, and use it as a measure of how well our neurons represent the input patterns. In other words, the quantization error is calculated by summing all the distances between each input pattern and the neuron to which it is mapped, giving a notion of the quality of the representation. In the case of a discrete data, the error function of SOM can be defined as [26]:

$$E = \sum_{j=1}^{v} \sum_{i=1}^{m} h_{bi}\|x_j - w_i\|^2 \tag{4}$$

where v is the number of training samples, and m is the number of map units. The neighborhood function h_{bi} is centered at b, which is the BMU of vector x_j, and evaluated for unit w_i.

5.3. SOM applications

After training, the neuron weights in the map can provide a model of the training patterns mainly through: vector quantization, regression and clustering. Using these techniques, self-organizing maps have been applied in the last decades to multiple applications [27] in areas like automatic speech recognition, monitoring of plants and processes, cloud classification, micro-array data analysis, document organization, image retrieval, etc.

But mainly there are two areas, strongly related with this work, which are optimization and prediction. The first because of the EACO applications scenario, and the last because there is a huge potential in combining these techniques in prediction scenarios, where SOM has been extensively applied [28]. We discuss these topics in the next section.

6. Discussion

In this section we compare the main characteristics of cEAs and SOM to explore the possibilities of applying game theory and coalitions in the latter.

6.1. Comparing cEAs and SOM

On the one hand, cEAs are constructed over cellular automata, but introducing evolutionary operators in order to define the next state (usually a point in the input space). Even they introduce a more complex algorithm to define the next state than elemental cellular automata, they share the main characteristics with basic CAs as a grid defined in a d-dimensional output space (the lattice), a neighborhood function and the next state depending on the state of the center cell and the state of its neighbors.

On the other hand, SOM also shares this same basic characteristics with cEAs, as all of them are basically a type of cellular automata model. SOM stores weights in the neurons state, mapping patterns from the n dimensional input space to the d dimensional lattice of the output space (usually $d \ll n$). The neuron weights evolve depending on their previous value, the new input vector $x(t)$, and the proximity of such neurons to the BMU, i.e., the neighborhood around the cell closest to the new input pattern.

If we add a game theory approach, then every cell has the autonomy to select adaptively its own neighborhood depending on its own needs, and the particular conditions of the cells around it. As introduced before, EA with Coalitions (EACO) is a new class of EA with dynamic population topology that aims at taking profit of the benefits of the two main existing population structures, namely cellular and island populations. This is achieved by introducing the concepts of game theory and coalitions, according to the policies defined in Section 4.

Therefore, the idea is to define new SOM algorithms introducing elements from game theory, as the duality between competition–cooperation needed to define a new model with coalitions able to create dynamic neighborhoods that consider how many times a neuron in the map becomes the BMU, and trying to avoid unused cells. To analyze this, next we explore some of the classical neighborhood functions used either for cEAs or SOM.

6.2. Neighborhood functions

All these CA models base the interaction between cells in neighborhoods, which can be defined using predefined shapes or functions. Fig. 1 presents some of the most typical neighborhoods used in Cellular Automata.

To define the neighborhoods in SOM, it is usual to select a function that monotonically decreases to zero up to a radius r (denoted as neighborhood radius) and zero from there onwards. The two most common neighborhood functions used are the continuous Gaussian hg and the discrete square hs. Considering a 2D lattice, a BMU b and any other neuron i of the grid, we can defined these functions as:

$$hg_{bi}(t) = \exp\left(-\frac{\|w_b(t) - w_i(t)\|}{2r(t)}\right)^2 \tag{5}$$

$$hs_{bi}(t) = \begin{cases} 1 & \text{if } \|w_b(t) - w_i(t)\| \le r(t) \\ 0 & \text{if } \|w_b(t) - w_i(t)\| > r(t). \end{cases} \tag{6}$$

In both cases, the radio $r(t) \to 0$ during training in order to achieve convergence. A theoretical discussion of the effect of neighborhood functions appears in [29].

6.3. Coalitions as dynamic neighborhoods

EACO removes some classical required parameters for distributed EAs, like the fixed neighborhoods, or the islands connectivity topology, the migration frequency, or the policies to exchange and discard individuals. This is achieved thanks to the use of spatial cellular approaches with dynamic neighborhoods, that allows the formation of coalitions among cells as a way to create islands of evolution in order to preserve diversity.

Fig. 7 presents a snapshot of a 2D grid obtained from the EACO algorithm when optimizing the MMDP problem with (20×20) 400 cells. The figure on the left side presents several coalitions present on that particular time of the execution, in different colors; while independent cells (not belonging to any coalition) appear in black color. These dynamic coalitions evolve along generations to match the interest of the players participating in the game, i.e., the cells. This allows to create dynamic neighborhoods that adapt continuously to the particular conditions of the different cells in a certain area.

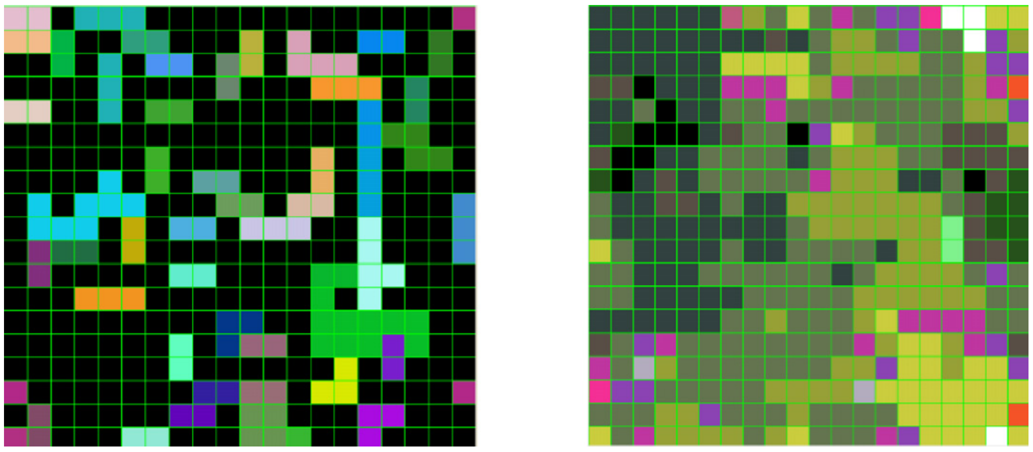

Fig. 7. A snapshot of an EACO run showing on the left side the set of coalitions, where independent cells are in black color. The right side represents the quality of the solutions, where darker cells represent better solutions.

6.4. A coalitional SOM training algorithm

An algorithm for training the SOM network with coalitions could be:
For each input pattern $x(t)$:

1. Find the closest neuron with index b in the map:
   $$(x(t)) = argmin\|x(t) - w_i(t)\|, \quad \forall i \in \{1 \dots m\}.$$
2. Update each neuron in the grid according to the rule:
   $$w_i(t+1) = w_i(t) + \alpha(t) hc_{bi}(t)[x(t) - w_i(t)], \quad i = 1 \dots m.$$
3. Update the coalitions in the grid, for every neuron, based in its local decisions:
   $$c_i(t+1) = \{j \,|\, j = getBestCoalition(i, t), j \in \{0 \dots m\}\}, \quad i = 1 \dots m.$$
4. Repeat the process until a certain stopping criterion is met.

The first difference with the classical algorithm is the coalition assignment introduced in item 3, where a cell i may belong to a coalition depending on index j. If $j = 0$ then cell i is independent, otherwise it belongs to a coalition originally started by j. The second difference is the new neighborhood function $hc_{bi}(t)$ in step 2 that takes into account if the cells b and i belong to the same coalition or not. Even we can define multiple neighborhood functions considering coalitions, a basic proposal could be:

$$hc_{bi}(t) = \begin{cases} 1 & \text{if } ((c_i(t) = 0) \text{ and } \|w_b(t) - w_i(t)\|) \le r(t) \\ 1 & \text{elseif } (c_b(t) = c_i(t)) \\ 0 & \text{else} \end{cases} \tag{7}$$

where cells close to the BMU, when it is an independent cell (1st condition), are in the neighborhood. Otherwise, cells in the same coalition are also in the neighborhood (2nd condition). Finally, cells do not fulfilling any of the previous two conditions are not neighbors (3rd condition). Concerning the general rules for creating, joining or leaving a coalition; we can review the ones considered in [22], and rewrite[1] them as:

- Joining other cells in the neighborhood of a cell can be good for those cells that have never been close to any input vector (pattern).
- Every coalition or independent cell should have a global valuation, depending on the average distance to previous input vectors, that determines its "quality".
- Cells from a given coalition can interact with any other cell in the same coalition, so the coalition behaves like a panmictic island.
- Cells can leave a coalition, joining other neighboring ones or create a new coalition, according to their selfish behavior to maximize the expected benefit. Here the benefit can be defined as their success in mapping input patterns.
- Cells can only belong to one single coalition.

1 Here the term cell is equivalent to solution, neuron or unit.

- Independent cells may have any of the classical neighborhoods: von Neumann, Moore, etc.
- There can be a parameter, denoted as coefficient of independence ($Ind_c \in [0, 1]$), to model the desire that cells have to be independent.

7. Conclusions

This article first introduces the areas of Cellular Automata, Game Theory, Coalitions and Cellular Evolutionary Algorithms, and considers their application to function optimization by means of EACO. The EACO algorithm integrates all these topics in a way to obtain a synergy to develop an Evolutionary Algorithm, with the advantages of different classical distributed models, but avoiding some of their limitations. This can be done thanks to the use of spatial cellular approaches with neighborhoods, allowing the formation of coalitions among cells as a way to create dynamic islands of evolution in order to preserve diversity.

Besides, EACO uses Game Theory to consider every cell as a player of a game arranged in a two dimensional torus. Cells will be able to evolve depending on their payoff with respect to their neighbors, and have also the support provided by their coalition members. This approach allows the payoff of a given solution to be defined in terms of how much such solution has improved in the last generations. The idea is to speed up the evolution of individuals grouped in high-quality coalitions that are quickly converging to promising solutions.

Once EACO has been introduced, we do so with Self-organizing Maps (SOM), a type of neural network with multiple of applications described along the last two decades. Finally, we compare both approaches to explore the similarities, and evaluate the possibilities of merging both models to get a richer, promising and eclectic combination to apply in the areas of optimization and prediction.

The main contribution of this article is to describe the strong similarities between cEAs and SOM models, and how introducing multidisciplinary approaches we can produce new research paths. EACO is an example, and we think that there is strong potential for using similar approaches as a way to define dynamic neighborhoods in SOM models. We consider that starting from here, there is a new interesting path open for future work.

Acknowledgments

The author wants to thank the coauthors from reference [22] for stimulating discussions, and the anonymous reviewers for their helpful comments.

References

[1] S. Wolfram, A New Kind of Science, 2002.
[2] K. Binmore, Game Theory, Mc Graw Hill, 1994.
[3] J. Maynard Smith, Evolution and the Theory of Games, Cambridge University Press, 1982.
[4] T. Bäck, D.B. Fogel, Z. Michalewicz, Handbook of Evolutionary Computation, Oxford University Press, 1997.
[5] E. Alba, B. Dorronsoro, Cellular Genetic Algorithms, in: Operations Research/Computer Science Interfaces Series, Springer-Verlag, 2008.
[6] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Transactions on Evolutionary Computation 6 (5) (2002) 443–462.
[7] T. Kohonen, Self-Organizing Maps, third ed., Springer-Verlag, 2001.
[8] J. von Neumann, The Theory of Self-Reproducing Automata, Univ. of Illinois Press, Urbana, IL, 1966.
[9] M. Gardner, Mathematical games: the fantastic combinations of John Conway's new solitaire game life, Scientific American (1970).
[10] P. Rendell, Turing universality of the game of life, in: Collision-Based Computing, Springer-Verlag, 2002, pp. 513–539.
[11] S. Wolfram, Theory and Application of Cellular Automata, World Scientific, Singapore, 1986.
[12] C.G. Langton, Computation at the edge of chaos, Physica D 42 (1990).
[13] M. Cook, Universality in elementary cellular automata, Complex Sys. 15 (2004) 1–40.
[14] O. Morgenstern, J. von Neumann, The Theory of Games and Economic Behavior, Princeton University Press, 1947.
[15] J. Nash, Equilibrium points in n-person games, Proceedings of the National Academy of Sciences of the United States of America 36 (1) (1950) 48–49.
[16] R. Axelrod, The Evolution of Cooperation, Basic Books, New York, 1984.
[17] M.A. Nowak, R.M. May, Evolutionary games and spatial chaos, Nature 359 (1992) 826–829.
[18] P. Langer, M.A. Nowak, C. Hauert, Spatial invasion of cooperation, J. Theoret. Biol. 250 (2008) 634–641.
[19] J. Nash, Non-cooperative games, The Annals of Mathematics 54 (2) (1951) 286–295.
[20] J. Yang, Z. Luo, Coalition formation mechanism in multi-agent systems based on genetic algorithms, Applied Soft Computing 7 (2) (2007) 561–568.
[21] J.C. Burguillo, A memetic framework for describing and simulating spatial prisoner's dilemma with coalition formation, in: 8th International Conference on Autonomous Agents and Multiagent Systems, 2009.
[22] B. Dorronsoro, J.C. Burguillo, A. Peleteiro, P. Bouvry, Evolutionary algorithms based on game theory and cellular automata with coalitions, in: Handbook of Optimization, Intelligent Systems, Vol. 38, Springer, 2013, pp. 481–503.
[23] D. Whitley, S. Rana, J. Dzubera, K.E. Mathias, Evaluating evolutionary algorithms, Artificial Intelligence 85 (1997) 245–276.
[24] T. Kohonen, Self-Organizing Maps, second ed., Springer-Verlag, 1995.
[25] C.E. Pedreira, R.T. Peres, Preliminary results on noise detection and data selection for vector quantization, in: Proceedings of the IEEE World Congress on Computational Intelligence, WCCI06, 2006, pp. 3617–3621.
[26] T. Kohonen, Self-organizing maps: optimization approaches, in: Artificial Neural Networks, Elsevier, 1991, pp. 981–990.
[27] M.M. Van Hulle, Self-organizing Maps: Theory, Design, and Application, Kaibundo, Tokyo, 2001.
[28] G.A. Barreto, Time series prediction with the self-organizing map: a review, in: Perspectives of Neural-Symbolic Integration, in: Studies in Computational Intelligence, vol. 77, 2007, pp. 135–158.
[29] H. Ritter, T. Martinetz, K. Schulten, Neural Computation and Self-Organizing Maps, Addison-Wesley, 1992.