Self Organizing Classifiers: First Steps in Structured Evolutionary Machine Learning

Danilo Vasconcellos Vargas · Hirotaka Takano · Junichi Murata

Abstract Learning classifier systems are evolutionary machine learning algorithms, flexible enough to be applied to reinforcement, supervised and unsupervised learning problems with good performance. Recently, self organizing classifiers were proposed which are similar to learning classifier systems but have the advantage that in its structured population no balance between niching and fitness pressure is necessary. However, more tests and analysis are required to verify its benefits. Here, a variation of the first algorithm is proposed which uses a parameterless self organizing map (SOM). This algorithm is applied in challenging problems such as big, noisy as well as dynamically changing continuous input-action mazes (growing and compressing mazes are included) with good performance. Moreover, a genetic operator is proposed which utilizes the topological information of the SOM's population structure, improving the results. Thus, the first steps in structured evolutionary machine learning are shown, nonetheless, the problems faced are more difficult than the state-of-art continuous input-action multi-step ones.

1 Introduction

Learning Classifier Systems (LCS) are several algorithms inspired by evolution [29],[20]. They can be applied

Danilo Vasconcellos Vargas
Kyushu University, Fukuoka, Japan
E-mail: vargas@cig.ees.kyushu-u.ac.jp

Hirotaka Takano
Kyushu University, Fukuoka, Japan
E-mail: takano@cig.ees.kyushu-u.ac.jp

Junichi Murata
Kyushu University, Fukuoka, Japan
E-mail: murata@cig.ees.kyushu-u.ac.jp

to reinforcement learning problems (actually, they can solve supervised learning and unsupervised learning [26] as well but we will only focus on reinforcement learning in this article). Different from most reinforcement learning algorithms, however, LCS algorithms do not use state-action look-up tables to predict payoff. To solve RL problems, LCS systems use a set of individuals with condition-action-prediction rules, i.e., solving the problem with piecewise approximations [19]. In this manner, the difficulties that arrive from complex problems, where a large number of states and/or actions are required, can be avoided.

However, the dynamic niches[1] of solutions present in LCS allows over-generalized solutions with higher fitness to compete and win against specialized ones in low-fitness niches (niches where good solutions receive low payoff when compared to other niches), even though the specialized solutions would have a better performance. One way of solving this problem is to separate a fitness defined on a niche from fitnesses defined on other niches (i.e., having a good fitness on other niches would not influence the present niche). This is exactly the niched fitness concept which was introduced and used by Self Organizing Classifiers (SOC) (niched fitness is explained in Section 6). Notice that niched fitness requires well defined niches where the fitness of individuals can be measured and compared locally inside each niche. This concept is difficult to insert into current LCSs.

SOC are a recently proposed family of evolutionary machine learning methods with a structured popula-

---

[1] In this article, the term niche follows the Hutchinsonian niche definition [16]. Hutchinsonian niche is an n-dimensional hyper-volume composed of environmental features. Moreover, niching is basically clustering, i.e., the objective is to create niches (clusters) which are more similar in some sense.

tion (to the knowledge of the authors the first evolutionary machine learning algorithm to use a structured population). The main objective of SOC's construction was to overcome over-generalization problems (over-generalized solutions and related problems) in LCS [31]. In SOC systems, no balance between specialization and generalization are needed, since the niched fitness concept is used. Actually, the niched fitness concept indirectly requires a separation between the niching pressure from the fitness pressure, in other words, by requiring well defined niches, both objectives (i.e. the objective of creating good niches and the objective to reach good solutions) need to be clearly stated. SOC uses evolution only for finding good solutions (fitness pressure), letting the SOM face the clustering objective in parallel (niching pressure). Indeed, it is similar to co-evolutionary approaches. The fitness for each niche can be strength based[2] because the niched fitness already solved the problems with over-generalized classifiers.

Regarding the relationship between SOC and LCS. Although, SOC and LCS possess many similarities, it may not be correct to classify SOC as an LCS. There are many crucial differences that make SOC difficult to match with other LCS algorithms, see Table 1.

This article extends [31] to include a more robust method without modifying its simplicity. A parameterless SOM replaces the SOM algorithm conferring better adaptation properties and less parameters. Previously, SOC were applied to some complex multi-step RL problems with optimum or near optimum results even using small population sizes. This article shows results on four new challenging problems:

- Big mazes - Mazes with as much as four times the area of previous mazes.
- Noisy mazes - Mazes with noise.
- Changing mazes - Mazes which have their structure modified over a series of trials.
- Growing and compressing mazes - Mazes which increase or decrease in size and structure over trials.

Additionally, a new genetic operator is proposed which utilizes the topological information of the SOM population. Its use is shown to improve the results.

2 Learning Classifier Systems in Multi-step and/or Continuous Problems

LCS have been developed for a while, forming a wide and diverse literature. LCS are evolutionary based systems capable of solving problems by evolving a set of agents with condition-action-prediction rules which cooperate or compete with themselves. Here we will briefly review LCS applied to multi-step and/or continuous problems. For a detailed review of the literature, please refer to [29][20].

In problems with continuous actions, LCS was applied to many problems. To begin with, XCSF has been applied to function approximation [35],[10],[28]. Other works in function approximation include the LCS with fuzzy logic [30],[8],[11], neural-based LCS algorithms [7],[8] and genetic programming-based [18]. The success of LCS also span the control of robotic arms [24,9] and navigation problems [6,15].

However, applications to multi-step problems with continuous actions restrict to the mobile robot in a corridor [6] and the empty room with noise [15]. Complex multi-step problems were solved only for **discrete outputs** [19].

3 The Parameterless Self Organizing Map

The commonly used SOM is an algorithm capable of producing a projection of the input usually into two dimensional space. One of the most important advantages of the method is the preservation of the topological relationship of the input in the constructed map. In SOM, for every input x the grid of weights w_i compete for it (the closest weight wins the competition, i.e., the weight having minimum $||x - w_i||$). After the winning cell is decided, all cells in the grid are updated by the following equation:

$$\Delta w_i(t) = \epsilon(t)h_{i,c}\{x(t) - w_i(t)\} \qquad (1)$$

$$w_i(t+1) = w_i(t) + \Delta w_i(t). \qquad (2)$$

where $\Delta w_i(t)$ is the weight update in the current iteration t, ϵ is the learning rate and $h_{i,c}$ is the neighborhood function. x and w_i are respectively the input array and the weight of a given cell i when the winning cell index is c. The learning rate ϵ is a monotonically decreasing function with respect to the number of times the grid update was realized and neighborhood function is usually an exponentially decreasing distance based function. For example:

$$\epsilon = 0.1(0.999999)^t \qquad (3)$$

$$h_{i,c} = e^{-dist(i,c)^2}, \qquad (4)$$

where $dist$ is some distance metric applied on the grid, defining the topological relationship of the grid's cells. Figure 1 illustrates one iteration of the algorithm, showing how the grid adapts to a given two dimensional input.

---

[2] Strength based fitness that is directly proportional to the payoff. They came into disuse in the LCS literature because of consequential over-generalization problems.

Table 1 Difference Between Systems

Algorithm Type	Model	Fitness
LCS	set of condition-action-prediction rules	Accuracy based
SOC	dynamic state-table of prediction rules	Strength based
Standard RL	static state-action look-up tables	Strength based
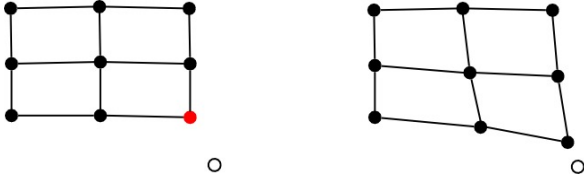
Fig. 1 Illustration of the SOM dynamics. The left part of the figure shows a two dimensional input (white dot) being presented to the SOM's grid (black dots connected by lines). In this scenario, the winning cell (closest to the input) is shown in red. Afterwards, the SOM's grid is updated and the resulting grid is shown on the right.

Fig. 2 Island model structure. Arrows indicate the infrequent immigration procedure, the circles are the individuals and the oval shapes are the subpopulations.
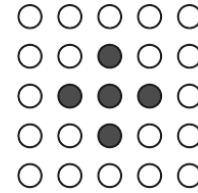
Fig. 3 Cellular algorithm structure. Shaded area indicates an example of neighborhood for the central individual.

A parameterless SOM is a SOM where the learning rate function is not necessary [5]. Instead, it is automatically defined by how good the SOM fits the input (in fact, this value is strongly related to the input's novelty [14],[23]). In the usual SOM, by making learning rate monotonically decreasing with the number of iterations, the more the SOM is used the less it could learn/adapt to new information. The parameterless SOM sets the learning rate according to the error of the input (not a monotonic decreasing function), therefore it is able to increase the learning rate when an rare or unexpected input arrives. Thus, the new parameterless SOM does not only possess fewer parameters, but also the ability to always adapt to changes in the environment.

Let the learning rate ϵ be defined as:

$$r(0) = ||x(0) - w_c(0)|| \tag{5}$$
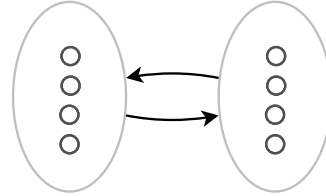
$$r(t) = max(||x(t) - w_c(t)||, r(t-1)) \tag{6}$$

$$\epsilon(t) = \frac{||x(t) - w_c(t)||}{r(t)}, \tag{7}$$

where w_c is the SOM winning cell's weight array. The weight update $\Delta w_i(t)$ is similar to SOM's weight update, changing only in relation to the new learning rate ϵ and the modified neighborhood function $h_{i,c}$. Considering $dist(i,c)$ the distance between cell i and the winner cell c, we have:

$$\Theta(\epsilon(t)) = \epsilon(t)\theta_{max}, \Theta(\epsilon(t)) > \theta_{min} \tag{8}$$

$$h_{i,c} = e^{\frac{-dist(i,c)^2}{\Theta(\epsilon(t))^2}} \tag{9}$$

θ_{max} and θ_{min} are respectively the maximum and minimum of $\Theta(\epsilon(t))$. In this article, θ_{max} equals to the SOM's area (width multiplied by the height of the grid) and $\theta_{min} = 1$ are used.

4 Structured Evolutionary Algorithms

Structured evolutionary algorithms does not possess a panmictic population. Instead they organize the individuals into a structured population [27,2]. As commonly considered in the literature, algorithms which has some sort of implicit structure will not be considered structured. In fact, to avoid this type of confusion the name of parallel evolutionary algorithms are sometimes used in the literature. The need for distinction derives from the fact that algorithms with implicit structure lose many of the benefits of ones with explicit structure.

Two types of structured EAs will be given as examples which are somewhat related to the structure of the proposed method.

The first type is island models (also called distributed genetic algorithms) [4]. Figure 2 shows its structure. Basically, the population is divided into a number of subpopulations ("islands") with few genetic information exchanged between them.

The second type, cellular algorithms are structured evolutionary algorithms where individuals are usually positioned in a vertex of a lattice graph (Figure 3 shows a common cellular structure). They interact solely with adjacent individuals defined by a neighborhood function [21,3].
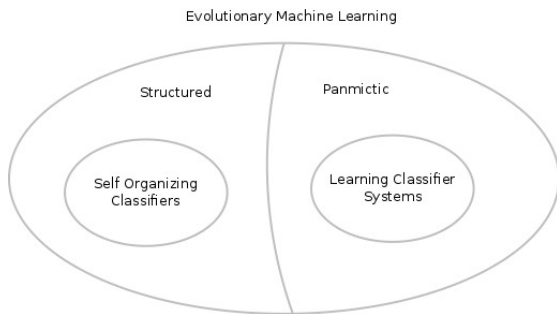
Fig. 4 Structured and unstructured division.

Fig. 5 SOM population structure. A self organizing map grid with subpopulations inside each cell.

5 Evolutionary Machine Learning - Structured and Unstructured

Methods in evolutionary machine learning can also be classified into structured and unstructured[3]. As explained in Section 4, structured algorithms arrange their population in some sort of structure whereas unstructured ones possess a single population set. That is, in the same way that structured evolutionary algorithms differ from unstructured panmictic ones, learning classifier systems can be classified, in this context, as a type of unstructured evolutionary algorithms and self organizing classifiers can be seen as a structured evolutionary algorithm. Figure 4 shows a diagram illustrating this aspect. Note that implicit structured algorithms such the niched genetic algorithm can not be considered structured following the definition in Section 4.

There is a motivation behind structured evolutionary machine learning. Indeed, some advantages of structured against unstructured algorithms should follow the ones from the optimization field. To cite a few:
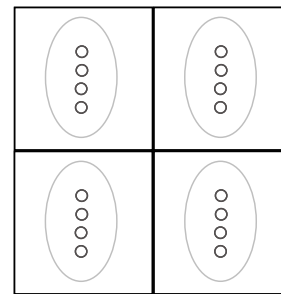
– Diversity - By restricting the interrelation of individuals, structured populations can preserve different subpopulations, i.e., avoid competition;
– Time Performance - Structured populations allow easier parallelization and therefore less running time when running in the appropriate system.

6 Novel Concepts

Two relatively novel concepts are used in the article, both concepts were introduced in [31]. They are respectively:

– Niched fitness - Niched fitness is a fitness defined only in a given niche (place or circumstance). Outside of this niche, the fitness is undefined and therefore nonexistent. The objective here is to evaluate

individuals by their performance in the given niche independent of how they behave in other niches, solving over-generalization problems. In fact, the niched fitness concept expose the similarity between niching and multiobjectivization [22].
– SOM population - SOM population is a 2D cell grid with each cell having a subpopulation. The grid behave as a SOM, self-organizing itself to the input and allowing only the subpopulation inside the winning cell to interact with the input. In other words, SOM population is a mixture of both island models and cellular algorithms with a self-organizing structure (See Figure 5).

7 Self Organizing Classifiers

SOC are a series of algorithms based on the SOM subpopulation, sharing similar model and dynamics. The schematic of SOC is described in Figure 6 (the schematic style is the same as the one previously used to describe ZCS, XCS and many other papers of the LCS literature [34]).

A Q-learning based reinforcement scheme with niched fitness is used by SOC. The fitness update of each individual is done using the Widrow-Hoff rule [33]:

$$F = F + \eta(\hat{F} - F), \tag{10}$$

where η is the learning rate, F is the current fitness and $\hat{F}$ is a new fitness estimate. Given an arbitrary classifier c activated at its SOM's cell $cell$. The fitness estimate of the pair $(cell,c)$ which were activated at time $t-1$ is given by the following equation:

$$\hat{F}(c, cell)_{t-1} = R_{t-1} + \gamma \max_{c' \in cell'} \{F(c', cell')\}, \tag{11}$$

where R is the reward received, γ is the discount-factor, $\max_{c \in cell} \{F(t)\}$ is the maximum fitness inside the activated cell $cell'$ at the current cycle t and c' is a classifier which has the current highest fitness in $cell'$.
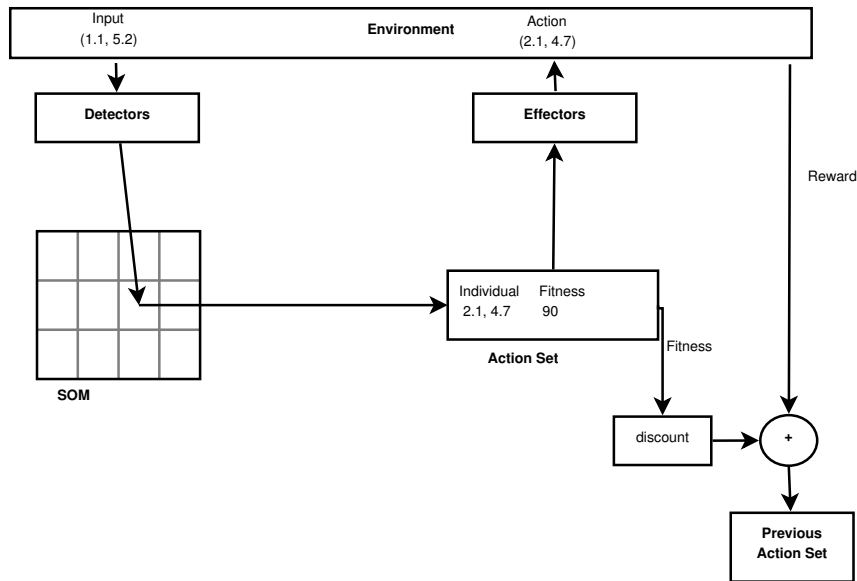
---

[3] Structured and parallel as well as unstructured and panmictic terms when referring to algorithms will be used indistinctly.

Fig. 6 Self Organizing Classifier Schematic

Similar to learning classifier systems, to decrease computation resources, the structure of the SOM population is implemented as a single array of classifiers with a given numerosity indexed by the SOM population structure. In this manner, the numerosity is defined by the number of indexes a given individual possesses.

8 Why SOM?

Before going into the further details of the algorithm, we wish to clarify one question that might surge. Among the vast amount of algorithms that could cluster inputs, why was SOM chosen?

The answer is that we chose SOM not for what it does, but for what it gives as information. In this context, SOM's interesting capabilities are as follows:

- Topological Preserving Projection - high-dimensional inputs are projected in a two-dimensional topological preserving map.
- Novelty Measure - The error of the SOM's cell given the input is an approximation to the uniqueness of the input itself, which is a measure of novelty [14], [23]. In fact, this measure is used to affect the update of the other cell weights in the parameterless SOM.

Evolutionary algorithms can use these pieces of information to better adapt to the problem at hand. Moreover, other procedures can be run on these information to improve the system as a whole. For example, genetic operators may exploit the SOM's structure by reproducing individuals with similar individuals in subpopulations adjacent to them. Observe that adjacent sub-populations are necessarily similar in input and therefore may share similar or even equal solutions.

9 Simplest Self Organizing Classifiers

Simplest Self Organizing Classifiers are implementations of the class of Self Organizing Classifiers (see Section 7). Figure 6 shows how the cycle is executed and Table 2 describes the execution cycle in details.

For the sake of comprehensibility, we give here the name of Simplest Self Organizing Classifier (SSOC) to the first very simple algorithm developed in [31] and the name of "Simplest Self Organizing Classifier 2" (SSOC2) to the one described in this article. The method proposed here replaces the SOM by a parameterless SOM, reducing two of the parameters since the learning rate function is not necessary [5]. There are not any other differences, therefore the definition here applies to both SSOC and SSOC2.

The classifiers used code directly the action by an array of real numbers, i.e., the model is an array of real numbers which is mapped directly to the output.

Moreover, the SOM population has particularly a subpopulation inside each cell which is also divided into two groups: one of best individuals and the other of novel individuals. Best and novel individuals have a fixed size of β and ν respectively. Considering evolutionary algorithm's cycle is an algorithm cycle when the evolutionary algorithm (EA) is called, the following rules take place:

- Best individuals are the best fitted individuals inside the subpopulation in the last EA's cycle.

Table 2 Simplest Self Organizing Classifiers' Cycle

---

1. An input is received by the system
2. The SOM population is activated on the input (a given individual will be returned to act)
   (a) The cell's weight array which is closest to the received input wins the competition
   (b) Inside the winning cell, a random individual is chosen either from the novel group or from the best group (depending if it is an exploration or exploitation cycle)
   (c) The cells' weight array of the SOM population is updated by the SOM algorithm
   (d) The chosen individual is returned
3. The chosen individual acts on the system (in the case of SSOC, the individual's chromosome is the action itself).
4. The previous acted individual on the past activated cell has its fitness updated. The equation used to update is written in Section 7 (notice that the fitness is updated only for the given cell even if the individual is present on other cells, see the niched fitness concept on Section 6).
5. Check if the EA should be called. If positive, execute the EA (see Section 9.1 for the complete description of the EA).

– Novel individuals are renewed every EA's cycle (the detailed process is described in the next Section).

The SOM population begins without any classifiers. Classifiers are created when the respective cell wins the competition inside the SOM. In one hand, novel individuals are created as random classifiers. On the other hand, best individuals, when possible, are set equal to another cell's best individuals from the neighborhood[4] which maximize $\frac{experience}{chebyshevDistance^2}$ (experience is defined on Section 9.1). If not possible, best individuals are initialized in the same way as the novel individuals. Observe that a given cell's experience are set to zero every time the evolution happens in it. This may seem counter-intuitive, but it alleviates a problem with hyper active cells which often possess a high error (see the cell's error for the growth of SOM in [1], [12]). A random selection of a close cell should output the similar results.

Cycles of exploration and exploitation are alternated (a cycle of exploration is followed by an exploitation cycle and so on). Within the SOM's winning cell in a giving exploration or exploitation cycle a random individual from respectively the novel or best individuals are chosen to act.

---

[4] Neighborhood is defined as the cells within a Chebyshev distance of less or equal to four

9.1 Evolution

For every cycle that a cell's individual acts, this cell has its experience counter increased. The evolutionary algorithm is called locally on each cell when the cell's experience is greater than ιS. S is the number of subpopulation individuals (novel plus best individuals) present on each cell and parameter ι defines an experience per individual, above which they should have an accurate fitness evaluation. After the evolution has been applied, the experience of the cell is set to zero.

By applying the evolutionary algorithm locally, SSOC2 (the same is valid for SSOC) respects the niched fitness concept. Its procedure consists of sorting the individuals of the given cell according to their fitness. On one hand, the current best β individuals substitute the previous best individuals and the remaining individuals are discarded (the index is removed and the individual numerosity decrease, if the numerosity reaches 0 it is deleted). On the other hand, novel ν individuals are created using either:

1. Indexing - A copy from (index to) a randomly selected individual of the entire population;
2. Reproduction - Created by a genetic operator.

The two procedures above have equal probabilities of happening.

The differential evolution operator is used in both SSOC and SSOC2. Motivated by its robustness and overall good results [25, 32, 17], even when compared against complex optimization algorithms (e.g., Estimation of Distribution Algorithms) [13]. In this paper, the differential evolution's mutant vector is created by randomly choosing three vectors from the SOM's entire population of individuals (notice that individuals with numerosity bigger than one are counted as one).

10 Experiments

The definitions done in Section 10.1 are important specially when discussing adaptation of the system in question.

10.1 Definitions

Definition 1 *Adaptation - Capability of a system to modify its behavior in order to better fit to changes in the environment.*

Definition 2 *Adaptation's Time - The time necessary for a system to adapt to changes in the environment.*
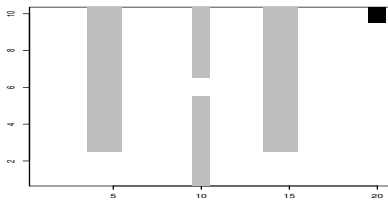
Fig. 7 Maze 1 - Static 10x20 maze problem.

A system with an adaptation's time of zero can be said to possess a generalized model where modifications are unnecessary to fit both past and current environments. In this sense, generality and adaptation are closely related terms. Although, it depends on the details about the system, e.g., the type of model used.

10.2 Environments

The experiments were conducted on both mazes of Figure 8 as well as on the dynamic mazes of Figures 9 and 10. Figures 9 and 10 show each two maze states, in each of these problems the maze is constantly changing every 10000 trials between one state (right side) and the other state (left side). Notice that the problem described in Figure 10 shows a maze which constantly increases and decreases in size. Agents act on all environments with continuous (x,y) translation actions. The variable observed by the agent is the agent's position which is also continuous.

At every trial, the agent starts at a random position on the environment. Naturally, starting inside a wall is not possible. Reaching the goal would give the agent a reward of 1000, hitting an obstacle would return -20 and any other action would return -10. Additionally, agents can not move more than 1.0 in any direction. The collision system is simply implemented (which makes it harder than a real system). If an agent tries to move inside a wall, the system detects the infraction, sets the agent in the previous position and returns the reward. In other words, an agent constantly hitting the wall will not move at all. However, an agent that hits the limits of a maze will have its final position limited by the environment. Therefore, it is possible to move sideways when hitting the limit of the environment.

10.3 Settings and Design of Experiments

The parameters of the algorithm are fixed and described in Table 3. Here, *it* means the SOM iteration number, *chebyshevDistance*() is the Chebyshev distance between the current cell and the cell which won the SOM's competition and *random*(a, b) is a function which
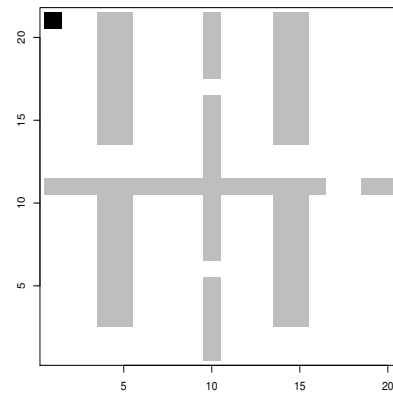
Fig. 8 Maze 2 - Static 21x21 maze problem.
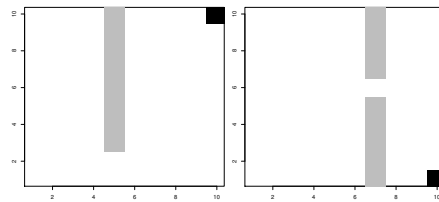
Fig. 9 Maze 3 - Dynamic maze problem with constant dimensions 10x10. The maze change from the left to the right state and vice-versa every 10000 trials.
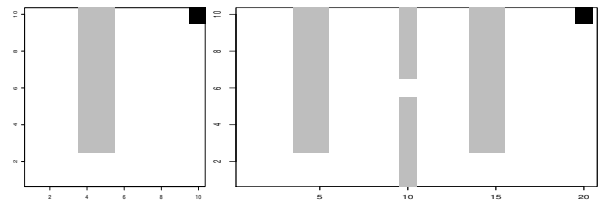
Fig. 10 Maze 4 - Dynamic maze problem where the dimensions also change, growing from 10x10 to 10x20 state and vice-versa.

returns a uniform random value between a and b. The cells of the SOM are only updated if the neighborhood function multiplied by the learning restraint surpasses the cell update threshold.

Following the design of [19] the performance is computed as the average steps to reach the goal during the last 100 trials. Moreover, trials can not last more than 500 steps. Any trial which last more than 500 is terminated and a new trial is started with the agent, as usual, in a random position. Every result, when not stated otherwise, are averaged over 20 experiments.

To give an overall idea of the resulting behavior of the agent (or its fitness) after the algorithm has learned, a sampling strategy was used. First, we divide the maze in blocks of size 1x1, then the action of the agent (or the fitness of the winning cell) is sampled 100 times and averaged in a given 1x1 block. By repeating this process for all blocks inside the maze, we have a general idea of the behavior learned (or fitness distribution).

Table 3 Parameters

Differential Evolution	CR	0.2
	F	$random(0,1)$
Self Organizing Map	Matrix Size	10×10
	Weight's initial value	$random(0,1)$
	Neighborhood function	$\exp(-chebyshevDistance()^2)$
	Cell update threshold	0.005
Self Organizing Classifiers	η	0.2
	β	2
	ν	5
	ι	20
	γ	0.90
	$InitialFitness$	0

10.4 Big Mazes

Big mazes challenge methods to create long chains of actions. In fact, the creation of long chains of actions may be difficult for various reasons. For example, fitness may not spread wisely, learning may take too long (e.g., due to maze bottlenecks), instabilities may arise, etc.

Figure 11 shows the behavior, population and performance of SSOC2 over Maze 1 with both the default 10x10 SOM population and with a smaller 5x5 SOM population. It can be seen that even with an insufficient population the algorithm achieves a solution capable of near optimal results. The same can be said for the results over Maze 2 in Figures 12 and 13. Despite the size of the maze, the quality of the result does not decrease much with smaller population sizes. In fact, the behavior stays more or less the same with the performance being a little worse. Both algorithms use a discount factor of 0.99 to enable a good spread of fitness.

SSOC2's sensitivity to the size of population is existent but it is, surprisingly, not big even with such simple classifiers. The sensitivity happens due to two limitations:

1. The SOM's current incapability of growing to couple with the difficulty of the problem in question.
2. SSOC2's classifiers are as simple as possible. Therefore, niching's granularity needs to increase for the method to cope with different actions. In fact, it is just evolving piecewise constant approximations (neural XCSF from [15] used piecewise nonlinear approximations to solve continuous input-action multi-step problems).

Therefore, with adaptable niching's granularity (growing and shrinking SOM population) as well as more complex classifiers (complex models) the sensitivity should disappear completely.

10.5 Noisy mazes

Real world problems always have some degree of noise involved. In some of them, the noise present can not be disregarded, affecting some algorithms undesirably. To simulate mazes where noisy can not be disregarded, a $\pm 5\%$ noisy variation is added to the observed variables.

The result is shown for both Mazes 1 and 2 on respectively Figures 14 and 15. For Maze 1, Figure 14 shows very good results with no visible issues. In fact, the performance is better than the results without noise (see Figure 11), i.e., it has less oscillations while converging to the same value. This happens because without noise the algorithm may get stuck, repeating for a long time the same input and consequently building a poor SOM model (the weights of SOM's cells will get near the repeated input). With noise the input received by the algorithm will rarely be the same. Actually, the algorithm may even found itself fewer times stuck, since the noise is constantly changing the input giving chance to other SOM cells to activate and therefore the possibility of different actions to arise. Maze 2 shows a slightly worse performance and behavior. This happens because aliasing states with very different fitness appeared. To explain this phenomenon, first take a look at the fitness in part A, B and C. Notice that parts A and C have greater values than part B. This is only possible because part C is getting a bit of fitness from the high fitness northwest portion of the map just above it. Due to the noise effects, part C may receive inputs from the northwest portion of the map, allowing it to receive a greater reward than it should. This influence of a high fitness aliasing state cause the unwise behavior at the frontiers. However, when the initial state is not in this problematic regions, the result is as good as without noise.

To give an idea of SSOC2's performance in relation to the evolutionary machine learning literature, Figure 16 shows the resulting behavior for the empty room with noise of [15]. Both SSOC2 and the neural XCSF
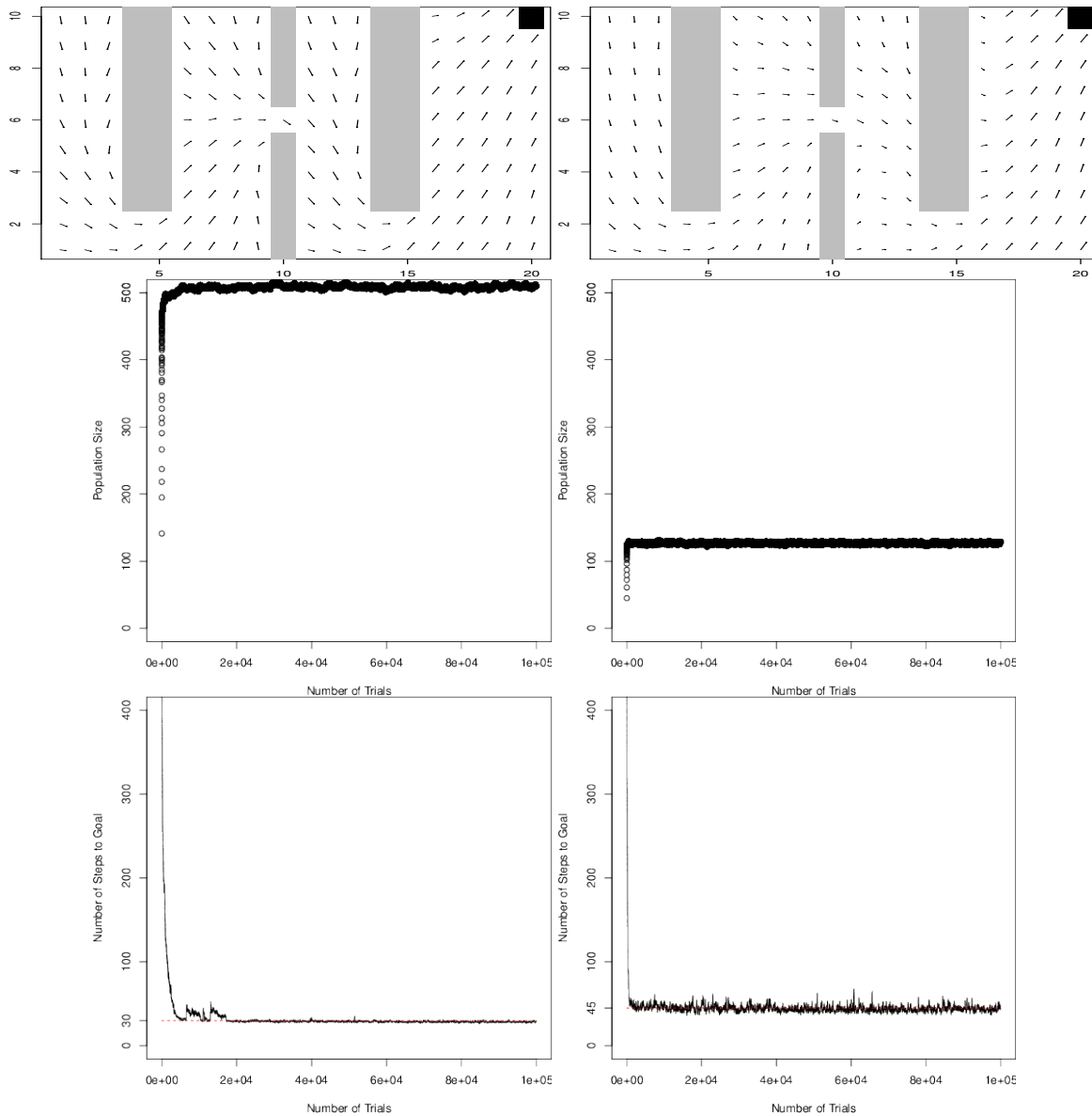
Fig. 11 SSOC2's behavior (upper row), population (middle row) and performance (lower row) throughout the trials. The left column shows the results for the default 10x10 sized population and the right show results for the smaller 5x5 SOM population.

converge to the optimum behavior. However, SSOC2 used a SOM population of size 5x5 which is equivalent to a maximum of 175 individuals in a panmictic population (each cell of the SOM population has 7 individuals, see Table 3), while XCSF took 16000 individuals to solve it, i.e., SSCO2 used 91 times less population. Both SSCO2 and XCSF were not optimized and these populations numbers are not the minimal population required to solve the maze. However it is still a quantitative measure that gives an overall picture of the population requirements for both algorithms on multi-step RL problems.

10.6 Changing Mazes

Problems in the real world are never static. To reflect this challenging characteristic, we consider a maze (see Figure 9) which changes from time to time. In other words, the agent's adaptation ability is put to test.

Figure 17 shows the number of steps required by the agent over time to solve Maze 3. Notice that the first two peaks are bigger than subsequent ones. This fact verifies the ability of the algorithm to reuse the knowledge when possible. Moreover, the repetition of exponentially decreasing learning curves demonstrate the agent's ability to change its model to reflect the
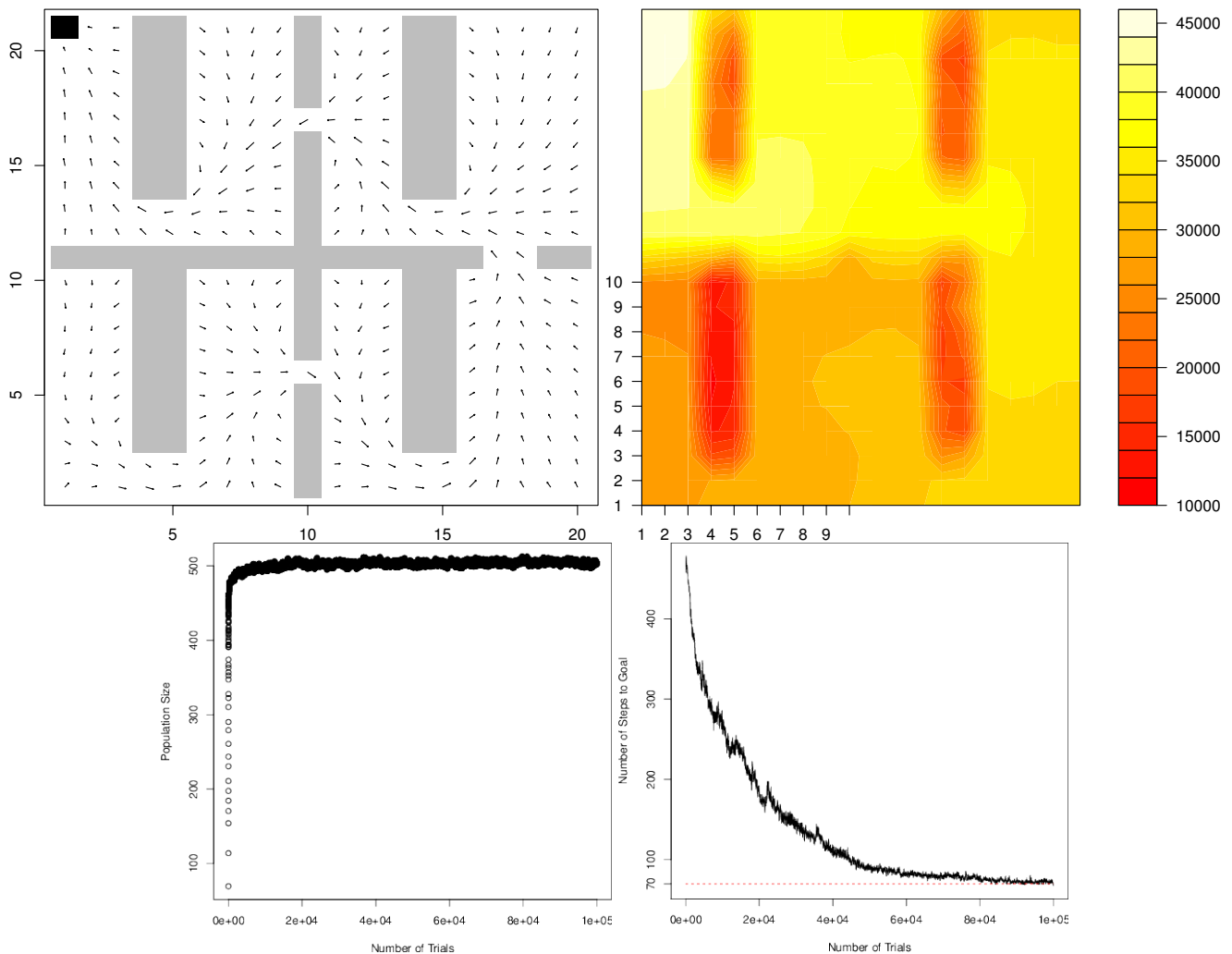
Fig. 12 SSOC2 applied to Maze 2 with 20x20 SOM population and discount factor of 0.99. The figure shows the behavior (upper left), fitness (upper right), population (lower left) and performance (lower right).

environment whenever the environment changes. This verifies the agent's adaptation ability.

10.7 Growing and compressing mazes

A special case of dynamic mazes are mazes which does not only change its internal structure but also size. That is the case for Maze 4's growing maze (see Figure 10).

The results demonstrate that SSOC has a good capability of adaptation which is also unchanged throughout the experiment (see Figure 18). Moreover, the decreasing peaks shows the reuse of knowledge from the previous problem. The system seems to arrive at a stage where minimal steps are required to adapt, i.e., minimal adaptation's time.

11 SSOC versus SSOC2

Here, SSOC2 and SSOC algorithms are compared. In the SSOC algorithm, $0.1(0.999999)^{it}$ is used as a learning rate function (parameter absent in a parameterless SOM).

The main difference between SSOC and SSOC2 are the incapability of changing the model when an early distribution was biased (or when the problem has just changed).Therefore, the results from Figure 19 are expected (SSOC was not able to continuously adapt as the learning rate gets smaller and can not increase with time). Figures 20 and 21 shows respectively a smaller problem (solved by both algorithms) and a bigger problem where SSOC had difficulties. Similar to problems that change with time, big mazes may cause the distribution of input from a series of trials to differ strongly. To make the differences explicit, Figure 22 shows a comparison between the resulting SOM's structure from
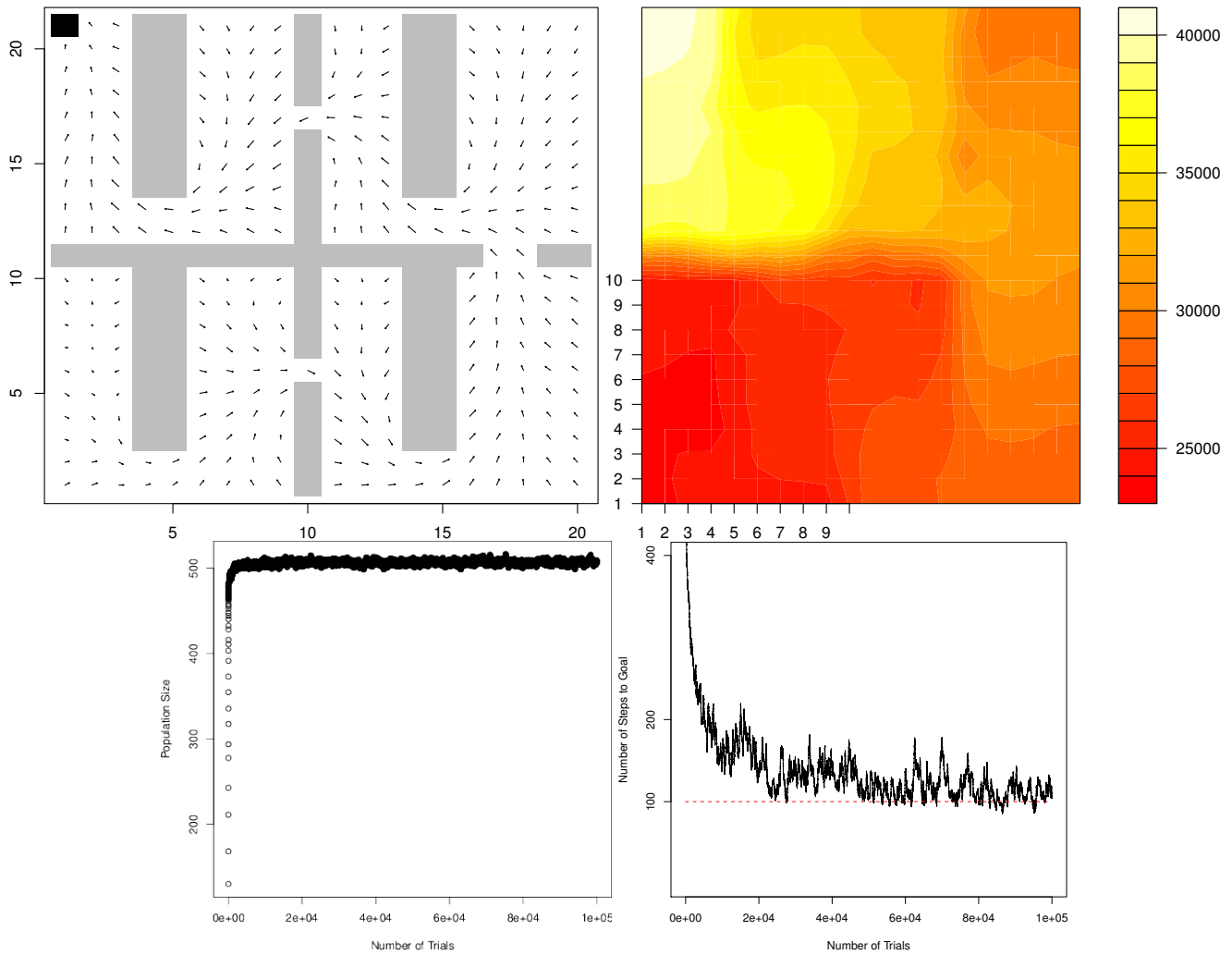
Fig. 13 Smaller population results. SSOC2 applied to Maze 2 with a discount factor of 0.99 (remaining parameters are the default, i.e., 10x10 SOM Population, etc). The figure shows the behavior (upper left), fitness (upper right), population (lower left) and performance (lower right).

both SSOC and SSOC2. It is clear that SSOC focuses on the frequency of input while SSOC2 focuses on the novelty of the input [5]. That explains the poorer coverage of SSOC. Having said that, a poor coverage does not mean a worse result. It depends on the system. For example, if the high frequency of inputs coincides with the most difficult inputs, the poor coverage would be a very useful mapping.
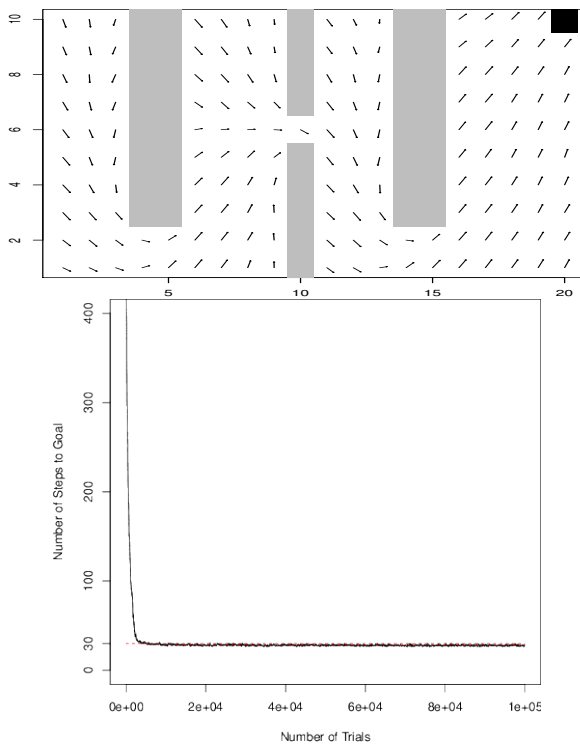
12 Mixed Genetic Operator

SOM's population is naturally a structure. However, until now the evolutionary algorithm did not take any

benefit from this aspect. Here we show that by using the topological preserving properties of SOM it is possible to mix global and local solutions to obtain relevant improvements.

The mixed genetic operator used is, as before, based on the differential evolution operator. But instead of choosing three random global solutions, each of the three solutions chosen come with equal probability from either a random adjacent cell (local) or a random solution (global). Results are shown on Figure 23. The number of steps necessary to reach the goal is on average approximately 60 after the algorithm has converged. Recall that using the previous (only global) genetic operator the average reached approximately 70 steps (see Figures 12). This result shows that evolutionary algorithms can explore the structure of the SOM with genetic operators, ending up also working on the inherent structure of the problem. In other words, it is possible

---

[5] recall that the error between the SOM's cell and the input, which is used to determine the rate of learning, is an approximation to uniqueness, i.e., a measure of novelty [14], [23]

Fig. 14 SSOC2's behavior and performance for Maze 1 with noise.

to efficiently reuse similar partial solutions (adjacent solutions) to solve similar partial problems (adjacent cells).

13 Justification

SSOC2 has a simple model. Nonetheless, it is still possible to achieve good results in complex problems with it. One way to think about it (as described in Section 1) is to see that niched fitness is necessary to avoid over-generalized solutions, deriving the rest of the reasoning from it. However, one might also think that at the heart of any problem, there is the necessity to divide and decouple as much as there is the necessity to solve the parts. In other words, divide and conquer[6] is essential. The hurdle is that it was always a multi-objective problem, treating it as a single-objective problem will consequently provoke either division or conquer strategies to prevail over another, while we may want both to co-exist[7]. By separating both niching and fitness pressures into a cooperating system, SOC managed to overcome this problem. In this sense, niched fitness becomes a consequence.

---

[6] The name of a widely known algorithm but most importantly a line of thought of how to solve problems in general
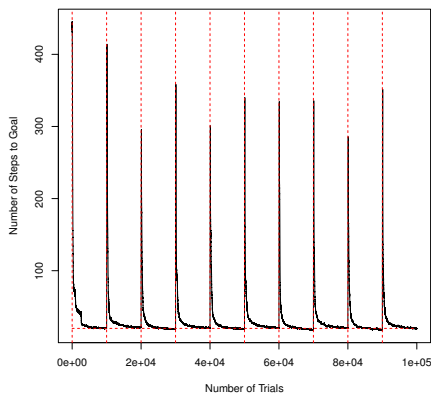
[7] over-generalizing solutions derives exactly from the prevalence of conquer strategies over division strategies
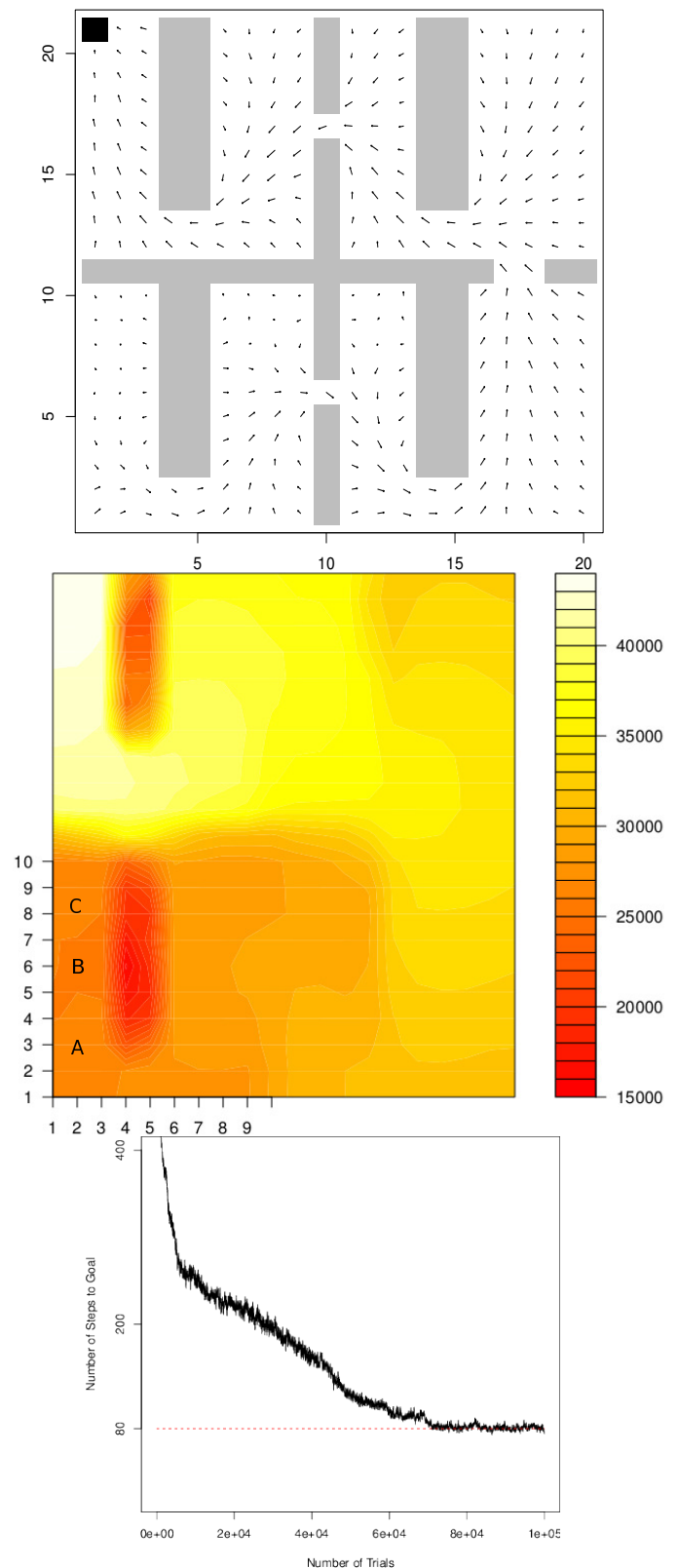

Fig. 15 SSOC2's behavior, fitness and performance for Maze 2 with noise. SOM population's size is 20x20 and the discount factor is set to 0.99.
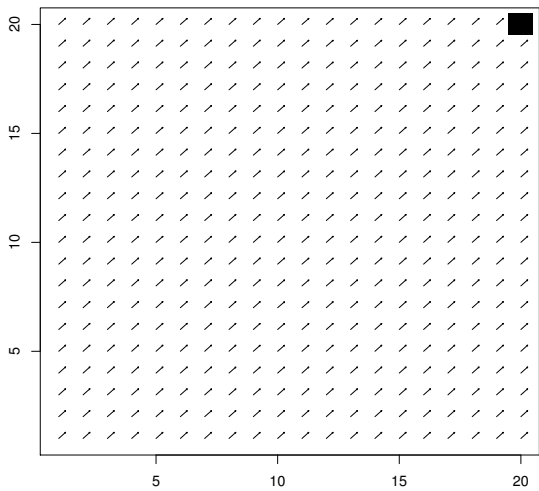
Fig. 16 Empty room maze with noise solved by SSOC2 with a SOM population's size of 5x5.


Fig. 17 Number of steps required by the agent to reach the objective in Maze 3. Vertical red dashed lines shows when the maze changed. A horizontal dashed line at 20 is drawn for orientation purposes.
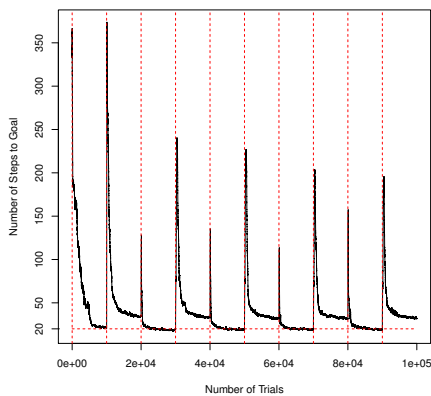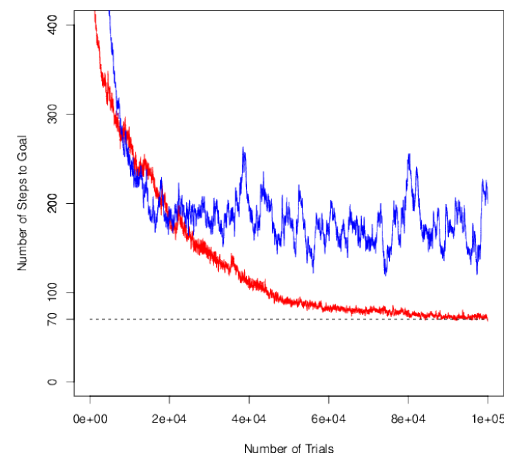

Fig. 18 SSOC2 performance on Maze 4. Vertical red dashed lines show when the maze changed. A horizontal dashed line at 20 is drawn for orientation purposes.
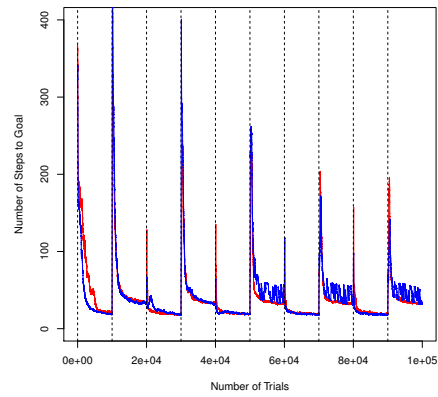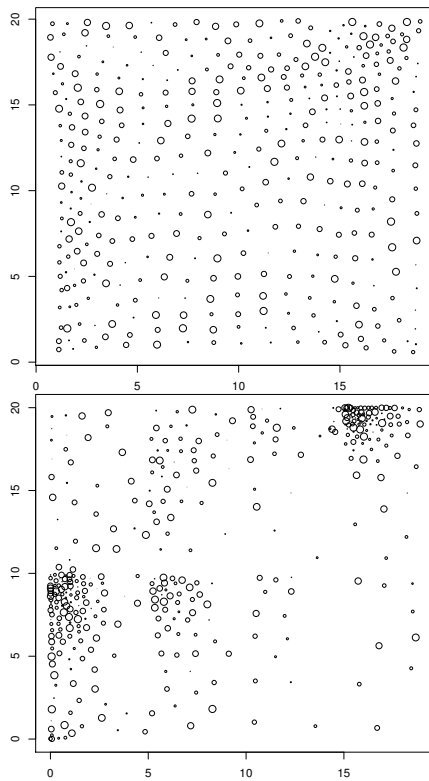

Fig. 19 Number of steps required by SSOC2 (red) and SSOC (blue) agents to reach the objective in Maze 4. Vertical dashed lines show when the maze changed.
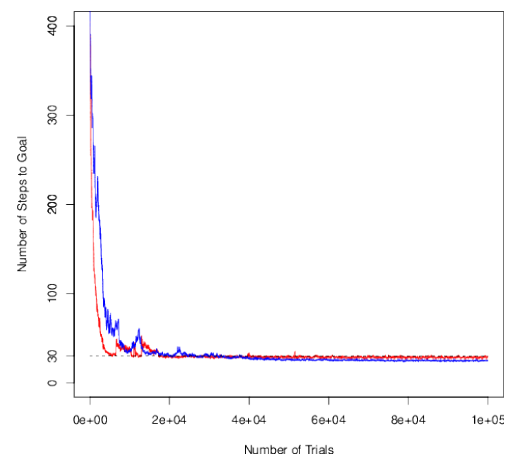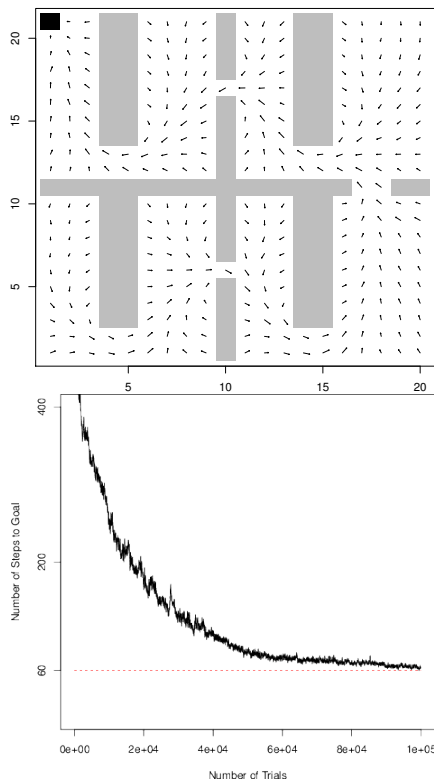

Fig. 20 Results on Maze 1 for SSOC2 (red) and SSOC (blue).

Fig. 21 Results on Maze2 for SSOC2 (red) and SSOC (blue).

Fig. 22 Two samples of the final distribution of SOM's cells for SSOC2 (above) and SSOC (below) for Maze 2. The size of the points is directly proportional to their experience (number of times they were selected to act).

Fig. 23 SSOC2 with mixed genetic operator applied to Maze 2.

13.1 The Mistake in the Metaphor

LCS are based on an amazing metaphor where individuals should behave as well as choose their own niche. There is no mistake in this metaphor. The problem comes when fitness is defined as an exaggeratedly simple single function per individual. In this point, niches and other interesting metaphors are discarded because the system is using an absolute fitness, while the fitness should be relative to the niches not only numerically but in meaning (there is no meaning in combining a fitness from a niche with a fitness of another niche, they are variables from different attributes). Monkeys may be amazingly adaptive in forests and many other environments, however, they are never able to interfere in any sense with the lantern-fish (fish of deep seawater). That is the metaphor mistake that niched fitness attempts to solve.

14 Conclusion

This article evaluated deeply the benefits and challenges of self organized classifiers (or more generally structured evolutionary machine learning). The following are the main points:

- SSOC2 - An improved version of SSOC (named SSOC2) that was created with the substitution of the SOM by the parameterless SOM. Differences in the final SOM population structured was shown as well as better results.
- State-of-art Results On Challenging Problems - Results were shown in a variety of continuous input-action multi-step problems. Problems with noisy and with dynamic environments were also considered. To the knowledge of the authors these problems are the most difficult continuous input-action multi-step problems faced by an evolutionary machine learning algorithm to date. Although SSOC2 is made of very simple classifiers (SSOC2 is capable of only piecewise constant approximations), near optimal results were presented.
- Genetic Operator - Tests have shown the advantages of using the SOM population's topological information inside the evolutionary algorithm. By using global and local solutions inside the genetic operator, the average number of steps required to reach the objective reduced in approximately 10 steps.

Thus, with the good results on very difficult problems it was possible to verify the strength of the approach even with a very simple internal model. There are many possible branches of research that can derive from SOC. In fact, structured evolutionary machine

learning algorithms were barely researched. Numerous widely different algorithms are possible from the combination of parallel (structured) evolutionary algorithms, learning classifier systems and machine learning. This is just the first step.

References

1. D. Alahakoon, S. K. Halgamuge, and B. Srinivasan. Dynamic self-organizing maps with controlled growth for knowledge discovery. *Neural Networks, IEEE Transactions on*, 11(3):601–614, 2000.

2. E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 6(5):443–462, 2002.

3. E. Alba and J. Troya. Cellular evolutionary algorithms: Evaluating the influence of ratio. In *Parallel Problem Solving from Nature PPSN VI*, pages 29–38. Springer, 2000.

4. T. Belding. The distributed genetic algorithm revisited. In *Proceedings of the Sixth International Conference on Genetic Algorithms: University of Pittsburgh, July 15-19, 1995*, page 114. Morgan Kaufmann Pub, 1995.

5. E. Berglund and J. Sitte. The parameterless self-organizing map algorithm. *Neural Networks, IEEE Transactions on*, 17(2):305–316, 2006.

6. A. Bonarini, C. Bonacina, and M. Matteucci. Fuzzy and crisp representations of real-valued input for learning classifier systems. *Learning Classifier Systems*, pages 107–124, 2000.

7. L. Bull. On using constructivism in neural classifier systems. *Parallel problem solving from nature-PPSN VII*, pages 558–567, 2002.

8. L. Bull and T. O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 905–911. Morgan Kaufmann Publishers Inc., 2002.

9. M. Butz and O. Herbort. Context-dependent predictions and cognitive arm control with XCSF. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1357–1364. ACM, 2008.

10. M. Butz, P. Lanzi, and S. Wilson. Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction. *Evolutionary Computation, IEEE Transactions on*, 12(3):355–376, 2008.

11. J. Casillas, B. Carse, and L. Bull. Fuzzy-XCS: A michigan genetic fuzzy system. *Fuzzy Systems, IEEE Transactions on*, 15(4):536–550, 2007.

12. M. Dittenbach, D. Merkl, and A. Rauber. The growing hierarchical self-organizing map. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 6, pages 15–19. IEEE, 2000.

13. S. García, D. Molina, M. Lozano, and F. 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. *Journal of Heuristics*, 15(6):617–644, 2009.

14. T. Hester and P. Stone. Intrinsically motivated model learning for a developing curious agent. In *The Eleventh International Conference on Development and Learning (ICDL)*, Nov 2012.

15. G. Howard, L. Bull, and P. Lanzi. Towards continuous actions in continuous space and time using self-adaptive constructivism in neural XCSF. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1219–1226. ACM, 2009.

16. G. Hutchinson. Concluding remarks. In *Cold Spring Harbor Symposia on Quantitative Biology 22*, pages 415–427, 1957.

17. A. Iorio and X. Li. Solving rotated multi-objective optimization problems using differential evolution. *AI 2004: Advances in Artificial Intelligence*, pages 861–872, 2005.

18. M. Iqbal, W. N. Browne, and M. Zhang. Xcsr with computed continuous action. In *AI 2012: Advances in Artificial Intelligence*, pages 350–361. Springer, 2012.

19. P. Lanzi, D. Loiacono, S. Wilson, and D. Goldberg. XCS with computed prediction in multistep environments. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1859–1866. ACM, 2005.

20. P. Lanzi and R. Riolo. A roadmap to the last decade of learning classifier system research (from 1989 to 1999). *Learning Classifier Systems*, pages 33–61, 2000.

21. B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithms. In *ICGA'89*, pages 428–433, 1989.

22. J. Mouret. Novelty-based multiobjectivization. *New Horizons in Evolutionary Robotics*, pages 139–154, 2011.

23. E. Reehuis, M. Olhofer, M. Emmerich, B. Sendhoff, and T. Bäck. Novelty and interestingness measures for design-space exploration. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 1541–1548. ACM, 2013.

24. P. Stalph and M. Butz. Learning local linear jacobians for flexible and adaptive robot arm control. *Genetic programming and evolvable machines*, 13(2):137–157, 2012.

25. R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

26. K. Tamee, L. Bull, and O. Pinngern. Towards clustering with xcs. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1854–1860. ACM, 2007.

27. M. Tomassini. *Spatially structured evolutionary algorithms*. Springer, 2005.

28. H. Tran, C. Sanza, Y. Duthen, and T. Nguyen. XCSF with computed continuous action. In *Genetic And Evolutionary Computation Conference: Proceedings of the 9 th annual conference on Genetic and evolutionary computation*, volume 7, pages 1861–1869, 2007.

29. R. Urbanowicz and J. Moore. Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009:1, 2009.

30. M. Valenzuela-Rendón. The fuzzy classifier system: A classifier system for continuously varying variables. In *Proceedings of the Fourth International Conference on Genetic Algorithms pp346-353, Morgan Kaufmann I*, volume 991, pages 223–230, 1991.

31. D. V. Vargas, H. Takano, and J. Murata. Self organizing classifiers and niched fitness. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 1109–1116. ACM, 2013.

32. J. Vesterstrom and R. Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1980–1987. IEEE, 2004.

33. B. Widrow and M. E. Hoff. Adaptive Switching Circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104, New York, 1960. IRE.

34. S. Wilson. ZCS: A zeroth level classifier system. *Evolutionary computation*, 2(1):1–18, 1994.
35. S. W. Wilson. Classifiers that approximate functions. 1(2-3):211–234, June 2002.