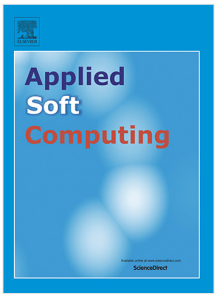


A population-based approach for multi-agent interpretable reinforcement learning

Marco Crespi, Andrea Ferigo, Leonardo Lucio Custode,
Giovanni Iacca

PII: S1568-4946(23)00776-7
DOI: https://doi.org/10.1016/j.asoc.2023.110758
Reference: ASOC 110758

To appear in: *Applied Soft Computing*

Received date : 30 May 2023
Revised date : 19 July 2023
Accepted date : 13 August 2023

Please cite this article as: M. Crespi, A. Ferigo, L.L. Custode et al., A population-based approach for multi-agent interpretable reinforcement learning, *Applied Soft Computing* (2023), doi: https://doi.org/10.1016/j.asoc.2023.110758.

Revised Manuscript (Clean version)

A Population-Based Approach for Multi-Agent Interpretable Reinforcement Learning

Marco Crespi, Andrea Ferigo, Leonardo Lucio Custode, Giovanni Iacca*

Department of Information Engineering and Computer Science
University of Trento
Via Sommarive 9, 38123 Trento, Italy

Abstract

Multi-Agent Reinforcement Learning (MARL) made significant progress in the last decade, mainly thanks to the major developments in the field of Deep Neural Networks (DNNs). However, DNNs suffer from a fundamental issue: their lack of interpretability. While this is true for most applications of DNNs, this is exacerbated in their applications in MARL. In fact, the mutual interactions between agents and environment, as well as across agents, make it particularly difficult to understand learned strategies in these settings. One possible way to achieve explainability in MARL is through the use of interpretable models, such as decision trees, that allow for a direct inspection and understanding of their inner workings. In this work, we make a step forward in this direction, proposing a population-based algorithm that combines evolutionary principles with RL for training interpretable models in multi-agent systems. We evaluate the proposed approach in a highly dynamic task where two teams of agents compete with each other. We test different variants of the proposed method in different settings, namely with/without coevolution and with/without initialization from a handcrafted policy. We find that, in most settings, our method is able to find fairly effective policies. Moreover, we show that the learned policies are easy to inspect and, possibly, interpreted based on domain knowledge.

Keywords: Multi-Agent Reinforcement Learning, Evolutionary Algorithm, Explainable Artificial Intelligence.

*Corresponding author
Email addresses: marco.crespi@studenti.unitn.it (Marco Crespi), andrea.ferigo@unitn.it (Andrea Ferigo), leonardo.custode@unitn.it (Leonardo Lucio Custode), giovanni.iacca@unitn.it (Giovanni Iacca)

1. Introduction

The field of Multi-Agent Reinforcement Learning (MARL) saw quick progress during the last decade [1]. Most of the milestones achieved in this field are due to the many breakthroughs in the field of Deep Learning (DL).

While DL approaches are powerful and widespread, they are weak in one fundamental aspect: they are not interpretable. Namely, it is difficult, for a human, to inspect a DL-based model and understand why an agent controlled by it performs a certain action [2]. Interpretability is especially interesting when the agents have to perform critical operations, which often happens where high interests are at stake: for example, in cases where the model interacts with humans, or in cases where multiple agents compete with each other, which will be the focus of this work. Moreover, if several agents interact while performing the task, not only do we have to account for the behavior of the single agent, but also for the *emergent behavior* of all agents taken together.

Here, we adopt a hybrid system developed for single-agent tasks, where we combine Evolutionary Algorithms (EAs) and RL to optimize models based on decision trees (DTs) [3]. In a gist, Grammatical Evolution (GE) [4] is used to optimize the structure of a DT, while Q-Learning [5] learns the actions performed by the leaves. To adapt this model to the multi-agent setting, we take a cooperative coevolutionary approach, where each agent (or group thereof) that takes part in fitness evaluation is handled by a different optimization process.

In a previous work [6], we employed a fully coevolutionary mapping (i.e., one optimization process per each agent) and compared it with a non-coevolutionary approach. In this work, we extend the work from [6] by: (1) trying to lower the overall complexity of the method, using a group-based mapping where we evolve fewer populations with respect to the total number of agents (i.e., we consider the case of one optimization process per each group of agents); and (2) trying to inject a handcrafted policy into the starting population of the evolutionary process, in order to evaluate the ability of the evolutionary algorithm at improving an existing policy. To summarize, the main contributions of this paper are the following:

1. We propose a novel coevolutionary approach that uses fewer parallel optimization processes (which, in turn, can potentially allow scaling to larger problems).

2. We introduce a bootstrap mechanism, that empowers the approach from [6] by enabling it to use previous knowledge to increase the efficiency of the evolutionary process.

3. We perform an experimental analysis of all the proposed approaches.

4. We interpret the DTs obtained with all the approaches, as a practical demonstration of the benefits of using interpretable models.

The rest of the paper is structured as follows: in Section 2 we overview the literature to highlight the novelty of the present work. In Section 3 we describe the proposed method, while in Section 4 we present the benchmark task used in the experimentation. Then, in Section 5 we discuss the results obtained. Finally, in Section 6 we draw the conclusions and indicate some future works.

2. Related work

In the following, we briefly summarize the most relevant works in the field of MARL. For a more exhaustive review of the field, we refer the interested reader to surveys of the field [7, 1, 2, 8, 9, 10]. Here, we intend to discuss only the most closely related works.

In [8], the authors explore the advantages that a multi-agent approach can offer with respect to a more complex, single-agent approach. Starting from that seminal study, several works have approached MARL settings from different perspectives. Sandholm et al., in [11], make a comparison between table-based and Recurrent Neural Networks (RNNs) in the iterated prisoner's dilemma. Their results show that RNNs tend to be less cooperative than table-based RL approaches. In another work [12] the authors observe Minimax Q, namely an algorithm that merges "minimax" and Q-learning, produces more robust policies in comparison to the ones learned by classical Q-learning.

Finally, Tan [13] proposes a scalable and decentralized version of Q-learning, called Independent Q-learning (IQL). However, this algorithm has a drawback, i.e., the inability of using NNs as function approximators for IQL. This is because NNs require a replay buffer, which is not compatible with IQL. More recent versions try to mitigate this problem, e.g., by using centralized critics [14], management agents [15], hysteretic Q-learning [16] or hysteretic deep recurrent networks [17], and reward shaping [18]. On the other hand, other works circumvent the problems arising with IQL by switching to an actor-critic model. Chu an Ye, in [19], propose using parameter sharing to improve efficiency in MARL. In [20], the authors study the impact of communication on the performance of the multi-agent system. Macua et al, in [21], propose an approach based on duality theory to derive a novel MARL approach. The method proposed in [22] uses a value-decomposition network to "translate" a single, joint reward signal into individual reward signals. Finally, the authors in

[23] propose a multi-stage approach, where each agent learns how to act in both single-agent and multi-agent scenarios.

Another approach is presented in [24], where the authors use Genetic Programming (GP) to evolve, in a cooperative coevolutionary setting, agents for a prey-predator task. The results indicate that the evolved strategies were more effective than the strategies handcrafted by the authors.

Finally, regarding the interpretability of machine learning models, recent works propose two ways to measure it. The first, taken from [25], uses a metric of interpretability based on the elements that compose the mathematical formula of the model. The second way, taken from [26], uses the computational complexity of the model as a metric for interpretability. In this work, as described in Section 5.1, we use the latter method, due to its greater simplicity. In fact, the underlying assumption is that the fewer computations are performed, the less complex is the model.

3. Method

Our goal is to co-evolve agents that are able to cooperate to solve a task (in our experimentation, we consider a specific task based on a competition between two teams, of which one is evolved, as we will show in Section 4; however, the proposed method can be applied also to other MARL tasks, provided a proper grammar). As we will show in this section, our agents are DTs composed of two elements that are optimized separately, as already proven successful in our previous works [3, 27, 28]. The inner structure of the DT is optimized by means of GE [4], while the Q-Learning algorithm [5] learns the optimal action for each leaf. By doing so, the agents can learn also *during* the task execution, without having to wait for the next generation as in traditional purely generation-based evolutionary processes.

As in this case we are working in a multi-agent setting, we need to devise a way to compose the team that will perform the task. To better analyze the effectiveness of our method, we test three strategies to compose a team. The first setting is a baseline non-coevolutionary approach where the team is composed of a *single, replicated, agent* (note that the agents have the same structure, but they act independently). We refer to this non-coevolutionary approach as *single-population mapping*. The second setting is a *fully coevolutionary mapping*, where each member of the team comes from a different optimization process. Finally, we consider a group-based setting, that we refer to as *partially coevolutionary mapping*, where a team is divided into groups, each one being handled by a different optimization process.

In addition to the three evolutionary settings mentioned above, we also consider two different initialization methods: one where the starting populations are *randomly initialized*, and one where they are *bootstrapped with a handcrafted policy* that is injected into the evolutionary process.

By comparing the six combinations of evolutionary settings and initialization methods, we are interested in answering the following research questions:

RQ1 Is the coevolutionary approach better than the non-coevolutionary one? What is the coevolutionary approach that yields better performance?

RQ2 Can we improve the performance of the evolutionary methods by injecting knowledge (in the form of a handcrafted policy) into the initial population?

RQ3 Can we interpret the agents obtained in the different settings?

3.1. Fitness evaluation

During the fitness evaluation, a team of A agents performs a task consisting in a simulation of N_{ep} episodes. As introduced before, at the beginning of the simulation (i.e., at the first episode) the agent is a DT that needs to learn the mapping between the leaves and actions. Each agent learns this function by IQL with a dynamic epsilon greedy exploration approach. Hence, it is independent of the other agents as they are considered part of the environment. However, if enough episodes are simulated, the agents, continuously interacting with the environment, will co-adapt their behavior to solve the task.

After the simulation (i.e., at the end of the last episode) we need to assign to the agents their fitness to measure the quality of the genotype. In particular, we need a method to aggregate the partial reward of each episode in a final, global value. Note that the aggregation method has to consider that we aim to measure the quality of the state-space decomposition function [3]. Hence, it has to be measured when the performance converges.

In a preliminary study (not reported here for the sake of brevity), we considered different aggregation functions, such as mean, median, maximum, and some percentile values. The resulting insights are shown below. Using the max function as an aggregator is prone to give too much importance to outliers, where good performance is due to random actions and not to "good" (i.e., stable, consistent) behaviors. Taking the average of the rewards has also a drawback, as the agents initially use a high epsilon value, hence the initial episodes have a high impact on the mean.

While the median would solve this problem, it would not properly weigh the episodes with a high performance. In fact, while we expect that the partial rewards increase towards the end of the simulation, the median would also consider the results when the model has not fully converged yet.

Therefore, after testing different percentiles to aggregate the partial rewards, we found that the 70^{th} percentile represents a good trade-off between the median and the maximum. The overall fitness evaluation process is illustrated in Figure 1.

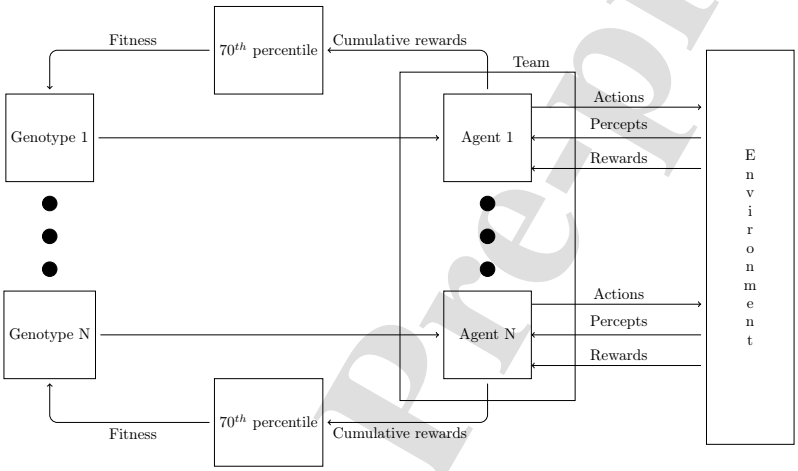


Figure 1: Graphical representation of the proposed method.

3.2. Team creation (mappings)

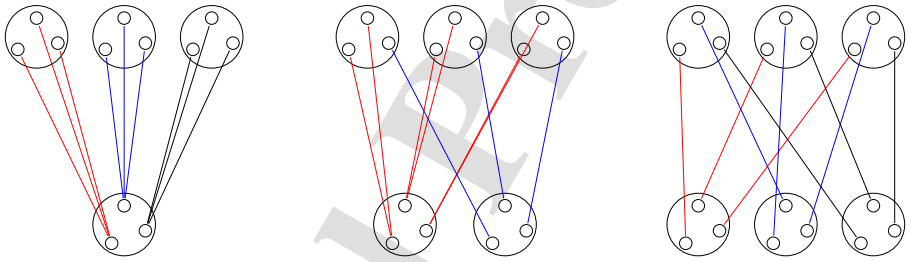
As we described in the previous section, during the fitness evaluation a team of A agents performs a task. These agents come from one or more evolutionary processes. As mentioned before, we map the agents of the team according to three different strategies, referred to respectively as single-population mapping, partially coevolutionary mapping, and fully coevolutionary mapping. Note that all the mappings create in all cases N teams, where N is the number of individuals in the population.

The *single-population mapping* uses a single evolutionary process to compose the N teams. Here we compose a team from a single individual, replicating it many times. Namely, the agents in the team share the same genotype (and its corresponding phenotype, i.e., a DT), but their behaviors

can be different in terms of actions performed in their leaves, as they are independently learned during the task by means of IRL. Hence, they can diverge based on the experience of the agents during the simulation.

In the *partially coevolutionary mapping*, we divide the team into m groups, with $m < n$, where n is the number of agents that are needed to compose a team. Hence, each group is composed of multiple agents sharing the same genotype (and its corresponding phenotype). Also in this case agents sharing the same genotype can however show different behaviors. Note however that, differently from the previous case, here, we utilize m optimization processes rather than one.

The last case is the *fully coevolutionary mapping*: this is the extreme case of the partially coevolutionary mapping when $m = n$, which leads to groups composed by a single agent and, hence, to n evolutionary processes (one per agent). Note that, in this case, all the agents have different genotypes, unless the migration mechanism (see below) moves a genotype to another evolutionary process. Figure 2 visually represents the three different mappings.



(a) Single-population mapping (b) Partially coevolutionary mapping (c) Fully coevolutionary mapping

Figure 2: Graphical representation of the proposed mapping schemes. The outer circles on the top represent teams that have to be evaluated, with the inner circles being the agents. Instead, the outer circles on the bottom are evolutionary processes, with the inner circles being individuals.

3.3. Grammatical evolution

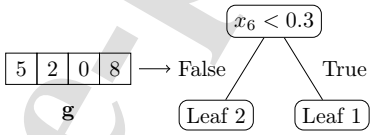
To optimize the agents, we use GE [4], where each genotype is a vector of integers (more on this below), and a population of genotypes is optimized through the application of mutation, crossover, selection, and replacement operators, repeated for a certain number of generations until a total number of individuals is evaluated. This algorithm has been already successfully applied to a number of RL and robotic tasks [29, 30, 31], hence it represents a valid solution for this kind of tasks. In this work, in addition to the traditional elements of GE, we add an migration and adoption operator

to move a solution from one optimization process to another, in order to improve the efficiency of the coevolutionary process.

Individual encoding. An individual is a list of integers in which each integer represents the index of the production to choose for the current rule. We choose a production by calculating the modulo of the integer currently selected and the number of possible productions. Unlike the original version of GE, in this work we use fixed-length genotypes. The overall procedure that translate the genotype into the phenotype is presented in Figure 3

Rule	Production
Root	`if`
If	`if Condition` then `action` else `action`
Condition	$\texttt{input}_{\texttt{index}} < \texttt{const}$
Action	`leaf` or `if`
const	$[1, 10]$, with step of 1
index	$[1, 10]$, with step of 1

(a)



(b)

Figure 3: Illustration of the individual encoding: (a) a simplified grammar; (b) example of translation of a genotype $\mathbf{g}$ into a DT, using the grammar shown in (a), with $M = 10$.

Mutation. We utilize uniform mutation, where each mutation is applied with probability p_{mut}, and each gene has a probability p_{gene} to mutate its value. The mutation changes the value of the gene sampling a random number in the range $[0, M]$, where M is a hyperparameter, whose value must be much larger than the number of production rules to ensure that they are distributed, approximately, uniform.

Crossover. We employ a simple one-point crossover that, given two genotypes g_1 and g_2, randomly split them in two parts, hence generating $g_{1,1}$, $g_{1,2}$, and $g_{2,1}$ $g_{2,2}$. Then, we compose the new genotypes by merging, respectively, $g_{1,1}$ with $g_{2,2}$, and $g_{2,1}$ with $g_{1,2}$.

Selection. As selection operator, we use the tournament selection: we create T tournaments, each randomly populated with N_T individuals. The best (in terms of fitness) from each tournament is then selected to become part of the population for the next generation.

Replacement. During the replacement phase, the new individuals, obtained by mutation and crossover, will replace their parent if they achieve better fitness. Note that, for individuals obtained

by mutation, the process is straightforward as they have only one parent. In the case of crossover, instead, the offspring has instead two parents: in this case, we simply replace only the worst parent. There is a third case, where an offspring is "adopted" (see the next paragraph): in this case, the replacement operator is applied on the adopted individual and its "foster parents", maintaining the one with better fitness.

Migration and adoption. Migration and adoption occur after applying the mutation and crossover operators. When an offspring is generated, a population is randomly chosen and the individual within it with the highest fitness value is selected and copied into the other populations. At this point, the migrating individual replaces a randomly chosen offspring from the receiving population, and the parents of that replaced individual are assigned as "foster parents" to the incoming individual. This last step is when the "adoption" operator takes place. The rationale of this operator is to propagate good individuals through populations but also to allow to replace migrating individuals with one of the parents, through the replacement operator, in case the incoming individuals do not perform well in the new population. Note that, when migration occurs, no trace of the individual is left in its original population.

While this mechanism may reduce the diversity in the populations, at the same time it improves convergence speed, which is crucial given the computational complexity and the large search space that characterizes the typical MARL tasks that our proposed approach is designed for.

3.4. Handcrafted policy injection

For each of the three mappings described in Section 3.2, we run two sets of experiments: one starting from random population(s), and one in which a handcrafted policy is injected into the initial population(s). Given an initial policy, we manually translate it into the corresponding decision tree. Then, we perform a "reverse-translation", which translates a DT into the corresponding genotype. This policy is then injected as an individual into each population at the beginning of the evolutionary process(es). When the individual is injected, it is copied into the population reaching, with a total number of copies set to 0.05% of the population size. This value has been set empirically so to avoid too many copies of the same individual (which may cause premature convergence). Note that this process does not increase the size of the population.

The goal of injecting a handcrafted policy into the starting population(s) is to provide the evolutionary process with additional knowledge. Note that, purposely, we do not provide as

handcrafted policy a complete "solution" for the environment, but only a generic structure for handling the four cardinal directions with respect to the agent's position. A partial representation of the resulting DT can be seen in Figure 4. Note that, as for the randomly generated individuals, the leaves do not contain actions, as these will be defined during the training episodes. The injected strategy is quite straightforward (see Table 3 for the definition of symbols): for each direction, we check if there are enemies, if there is an obstacle, or if the agent can attack a nearby enemy. The four directions with respect to the agent's position are evaluated in the following order: above, left, right and below.

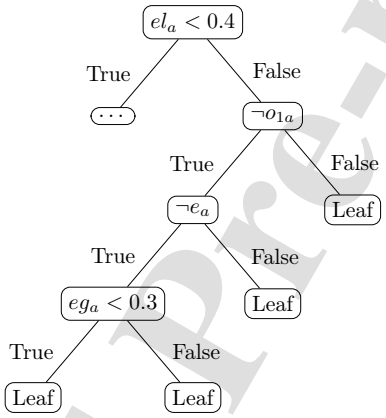


Figure 4: Partial representation of the DT obtained from the injected individual. The same structure is repeated in the empty branch $(\dots)$ by switching the directions, i.e., up/down and left/right. The meaning of the abbreviations contained in the nodes is shown in Table 3.

4. Environment

The environment used for our experiments is the Battlefield environment from the MAgent [32] suite contained in the PettingZoo library [33].

The Battlefield environment is a multi-agent game where there are two teams: the red team and the blue team. Each team is made up of 12 agents. The goal of each team is to eliminate all the agents in the opposing team. In order to achieve this result, the agents that belong to the same team need to learn how to coordinate their movements in an 80×80 grid surrounded by outer walls. In the middle of the environment, there are also three inner walls, see Figure 5.

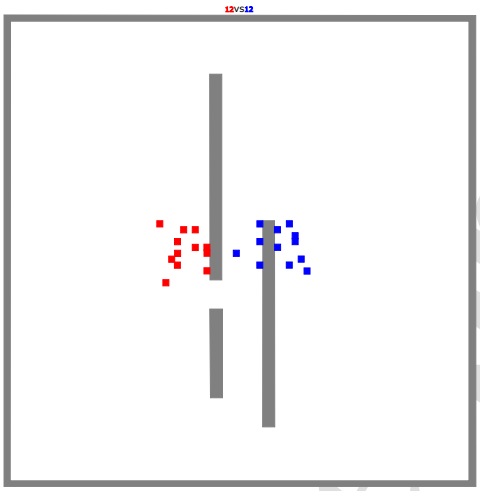


Figure 5: A screenshot from the Battlefield environment.

Each agent in both teams starts with 10 health points (hp) and when it takes a hit from an opponent (i.e., the opponent is close to the agent and carries out an attack in its direction) the amount of hp is reduced by 2. An agent receives 0.1 hp at every timestep, unless it already has 10 hp. An agent dies when it reaches 0 hp. Each episode lasts in total 500 timesteps.

The field of view of each agent is a 13×13 grid. An agent has both local and global perception. The local perception area corresponds to a (discretized) circle with a radius of six cells around that agent and provides to the agent information about presence / absence of an obstacle in a cell, presence / absence of an enemy in a cell, hp of the enemy in a specific cell, presence / absence of a teammate in a cell, and hp of the teammate in a specific cell. In the global perception, each cell of the perception grid represents an area of 7×7 cells in the environment and provides information about the density of enemies and teammates in that area. In order to simplify the learning phase (and the interpretability of the policies evolved), we perform a pre-processing of the inputs to extract high-level features that are used in the GE to obtain the DTs. An overview of these features used are reported in Table 3. Note that the "Abbreviation" column shows how we refer to a specific feature in the rest of the text.

An agent can choose between 21 discrete actions at each timestep. The possible actions are: no action; move to any of the 8 adjacent cells; move two cells on either left, right, up, down; attack any

of the 8 adjacent cells. Table 4 shown all the possible actions with the abbreviation used to refer to them.

At each timestep, an agent receives a reward from the environment. The rewards and their values are: 5 points if the agent kills an opponent; 0.9 points if the agent hits an opponent (to encourage the agent to hit opponents without waiting to kill an enemy to get the reward); -0.1 points if the agent dies; -0.1 points for any attack (to make the agent attack only when necessary); -0.005 points for each timestep (as a form of time penalty). Note that the environment does not provide an explicit reward for agent collaboration.

Table 1: Parameters used for the two algorithms (GE and Q-learning) used in the experimentation.

Algorithm	**Parameter**	**Value**
Grammatical Evolution	N_{ind}	60
	N_{gen}	40
	p_{xover}	0.4
	p_{mut}	0.8
	p_{gene}	0.05
	Genotype length	500
	Selection	Tournament
	s_t	3
Q-learning	α	$1/v$
	ε	1
	N_{ep}	400
	$decay_{\varepsilon}$	0.99

Table 2: Grammar used to evolve the DTs. "|" denotes the possibility to choose between different productions; "dt" indicates the starting symbol.

Rule	**Production**
dt	$\langle root \rangle$
root	$\langle condition \rangle$ | leaf
condition	if $\langle input\_index \rangle < \langle float \rangle$ then $\langle root \rangle$ else $\langle root \rangle$
input_index	$[0, 33]$, step 1
float	$[0.1, 0.9]$, step 0.1

Table 3: Extracted features, their abbreviation and their domain.

Feature	Abbr.	Domain
Obstacle 2 cells above	o_{2a}	{0, 1}
Obstacle 2 cells left	o_{2l}	{0, 1}
Obstacle 2 cells right	o_{2r}	{0, 1}
Obstacle 2 cells below	o_{2b}	{0, 1}
Obstacle 1 cell above-left	o_{1al}	{0, 1}
Obstacle 1 cell above	o_{1a}	{0, 1}
Obstacle 1 cell above-right	o_{1ar}	{0, 1}
Obstacle 1 cell left	o_{1l}	{0, 1}
Obstacle 1 cells right	o_{1r}	{0, 1}
Obstacle 1 cell below-left	o_{1bl}	{0, 1}
Obstacle 1 cells below	o_{1b}	{0, 1}
Obstacle 1 cells below-right	o_{1br}	{0, 1}
Allied global density above	ag_a	[0, 1]
Allied global density left	ag_l	[0, 1]
Allied global density same quadrant	ag_s	[0, 1]
Allied global density right	ag_r	[0, 1]
Allied global density below	ag_b	[0, 1]
Enemies global density above	eg_a	[0, 1]
Enemies global density left	eg_l	[0, 1]
Enemies global density same quadrant	eg_s	[0, 1]
Enemies global density right	eg_r	[0, 1]
Enemies global density below	eg_b	[0, 1]
Enemies local density above	el_a	[0, 1]
Enemies local density left	el_l	[0, 1]
Enemies local density right	el_r	[0, 1]
Enemies local density below	el_b	[0, 1]
Enemy presence above-left	e_{al}	{0, 1}
Enemy presence above	e_a	{0, 1}
Enemy presence above-right	e_{ar}	{0, 1}
Enemy presence left	e_l	{0, 1}
Enemy presence right	e_r	{0, 1}
Enemy presence below-left	e_{bl}	{0, 1}
Enemy presence below	e_b	{0, 1}
Enemy presence below-right	e_{br}	{0, 1}

Table 4: Available actions and their abbreviation.

Action	Abbr.
Move 2 cells above	m_{2a}
Move 1 cell above-left	m_{1al}
Move 1 cell above	m_{1a}
Move 1 cell above-right	m_{1ar}
Move 2 cells left	m_{2l}
Move 1 cell left	m_{1l}
No action	m_n
Move 1 cells right	m_{1r}
Move 2 cells right	m_{2r}
Move 1 cell below-left	m_{1bl}
Move 1 cells below	m_{1b}
Move 1 cells below-right	m_{1br}
Move 2 cells below	m_{2b}
Attack above-left	a_{al}
Attack above	a_a
Attack above-right	a_{ar}
Attack left	a_l
Attack right	a_r
Attack below-left	a_{bl}
Attack below	a_b
Attack below-right	a_{br}

5. Results

For each combination of mapping and initialization strategy, we perform 10 independent evolutionary runs. Note that we performed 10 independent runs as they were enough to produce statistically-significant results. Moreover, we had to trade-off between the number of runs to achieve statistical significance and the computational cost required for the experimentation. Another important thing to notice is that we evolve only the policy of the agents in the blue team, while the other team, the red one, performs random actions. This configuration prevents in fact the blue team from overfitting to specific strategies of the red one.

We show the parameters used by GE and Q-learning in Table 1, while we present the production

grammar in Table 2. Summarizing, each evolutionary process has a population of 60 individuals that evolve for 40 generations. For the RL algorithm, we use a dynamic learning rate to ensure that the Q function converges to the optimal one. In particular, the learning rate of each leaf is $\alpha = \frac{1}{v}$, where v is the number of times the leaf is visited.

Since the goal of the task is to eliminate all the opponents, we use three metrics to measure, a posteriori, the performance of the evolved policies over 100 unseen episodes. The first one is "No. kills", which indicates the average number of enemies killed in each episode. Note that a team is formed by 12 agents, therefore in a single episode the number of enemies killed is limited between 0 and 12. The second one is "Agent reward", which measures the average reward of the agents in the team. Finally, the last metric is "Completed", which measures the percentage of completed episodes, where an episode is considered completed when the team has completely eliminated the opposing team.

5.1. RQ1: Performance analysis

In this section, we answer RQ1. Hence, we compare the performance obtained by the three different mappings. We will observe the fitness trend during the evolution and compare the fitness of the best individuals obtained.

Firstly, Figure 6 shows the trend of the best fitness obtained by each method during the evolutionary process. The solid lines indicate the average between the 10 independent runs, while the shaded area represents the standard deviation.

Here, we can observe that the coevolutionary methods perform better than the non-coevolutionary baseline. Moreover, the fully coevolutionary approach outperforms the partially coevolutionary one. This happens in both cases, without (left) and with (right) the injection method. This may be due to the migration mechanism that allows good agents to spread between different evolutionary processes.

Figure 7 partially confirm these results. Here, we compare the best teams found at the end of the evolutionary processes. We compare them using 4 metrics: the final fitness (first row), the average number of kills obtained by the agents in the team, the average score obtained by the agents, and the percentage of completed episodes. As mentioned earlier, we calculate the last three metrics after the evolutionary process, by simulating the best individuals found at the end of the evolution in 100 new episodes. We further confirm the results by means of the Mann-Whitney-U statistical test (with confidence level $\alpha = 0.05$), applying the Bonferroni correction. We observe

that, when there is no policy injection (left column), the coevolutionary methods reach statistically better fitness and better number of kills than the non-coevolutionary approach. However, policy injection (right column) improves the outcome of the non-coevolutionary baseline, as discussed in the next subsection.

Finally, the metrics computed for each of the 10 independent runs are shown in Table 5.

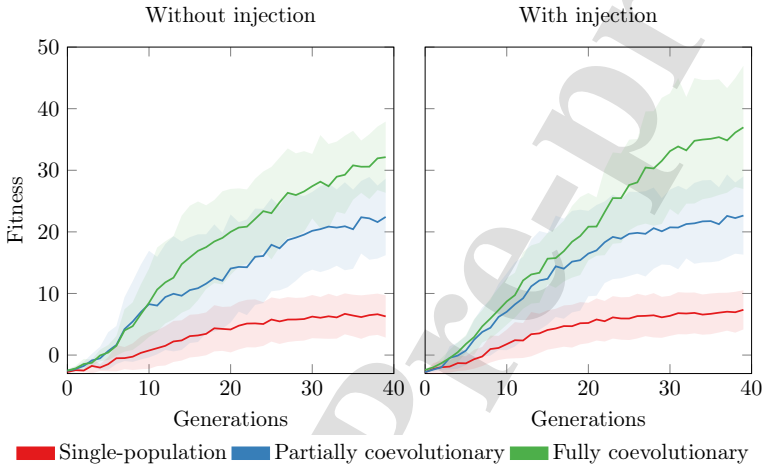


Figure 6: Evolutionary trends (average ± std. dev. across 10 independent runs) of the best fitness found at each generation of the proposed algorithm in the six tested experimental settings.

5.2. RQ2: Policy injection

Now that we have observed how the coevolutionary methods perform better than the non-coevolutionary baseline, our next goal is to assess the impact of the policy injection.

First of all, in Figure 8 we can observe a comparison between a team composed of injected handcrafted agents and one composed of randomly sampled agents. Here, we compare only the agent reward, as both teams during independent evaluations do not score any kills. However, we can see that the agent reward for the handcrafted policy statistically outperforms the random agents. Hence it is plausible to assume that injecting such handcrafted policy into the evolution process can provide a benefit.

In Figure 6, we can observe that the fitness trends are very similar between the two configurations, with the trends with injection converging slightly faster than the ones observed without injection.

Figure 7 confirms this trend, highlighting the difference in performance between the two method in the first row. However, in the a posteriori comparison (second, third and fourth rows), we observe that the non-coevolutionary approach is only slightly worst, and the difference is not statistically relevant.

In Figure 9, we can observe how the injection models overall seem to improve their performance with respect to the models without injection, although we cannot reject the null hypothesis in the various pairwise tests. This could mean that, the injection of a well-performing individual in a random population may strongly bias the optimization process towards a local optima that is not significantly better than the ones reachable by a random population. This is more evident in the coevolutionary approaches, where the distribution of the results significantly overlap.

Finally, it is worth noticing that the most important improvement appears when applying policy injection to the single-population approach, indicating that the coevolutionary approaches are able to learn policies that match the performance of the injected policy without need for human intervention.

Table 5: Results obtained for each of the methods in 10 independent runs.

Method	**Metrics**	**Runs**									
Single-population Without injection	No. kills	10.74	11.98	11.46	0.51	11.98	10.87	9.07	10.03	11.43	10.20
	Agent reward	5.89	8.20	7.37	−2.71	8.17	4.26	4.86	5.09	6.65	5.74
	Completed	44.00%	99.00%	79.00%	0.00%	98.00%	48.00%	18.00%	17.00%	74.00%	37.00%
Partially coevolutionary Without injection	No. kills	11.63	12.00	10.27	11.18	11.66	11.95	12.00	11.66	9.96	12.00
	Agent reward	7.04	8.40	5.73	6.14	7.57	7.90	8.39	7.14	6.24	8.44
	Completed	72.00%	100.00%	39.00%	50.00%	80.00%	95.00%	100.00%	79.00%	38.00%	100.00%
Fully coevolutionary Without injection	No. kills	11.96	11.85	11.98	11.97	11.81	11.91	8.98	11.65	11.15	11.90
	Agent reward	7.91	8.05	8.19	7.84	8.08	8.17	4.43	6.83	6.99	8.22
	Completed	96.00%	94.00%	98.00%	98.00%	91.00%	98.00%	1.00%	79.00%	63.00%	93.00%
Single-population With injection	No. kills	10.74	11.98	11.46	0.51	11.98	10.87	9.07	10.03	11.43	10.20
	Agent reward	5.89	8.19	7.36	−2.71	8.17	4.26	4.85	5.09	6.65	5.73
	Completed	44.00%	99.00%	79.00%	0.00%	98.00%	48.00%	18.00%	17.00%	74.00%	37.00%
Partially coevolutionary With injection	No. kills	10.89	10.86	10.80	3.58	11.74	11.60	11.96	11.97	11.98	11.98
	Agent reward	5.72	6.29	6.22	0.89	7.22	7.03	7.80	8.05	8.09	8.00
	Completed	50.00%	48.00%	41.00%	0.00%	81.00%	66.00%	97.00%	98.00%	99.00%	98.00%
Fully coevolutionary With injection	No. kills	11.83	11.99	11.63	11.95	11.38	12.00	11.93	11.34	11.11	12.00
	Agent reward	7.51	8.13	7.34	8.04	6.92	8.44	7.80	7.25	6.51	8.34
	Completed	90.00%	99.00%	82.00%	98.00%	56.00%	100.00%	97.00%	65.00%	54.00%	100.00%

5.3. RQ3: Agent interpretability

In this section, we describe the behavior of two selected agents, one evolved with the fully coevolutionary mapping, and one with the partially coevolutionary mapping, interpreting such behavior from their phenotype (i.e., their corresponding DTs). However, it should be noted that the considerations we will draw here are valid also for agents evolved in the other configurations. To

better understand the evolved policies, is worth remembering that we evolve only the blue agents' behavior. Moreover, we assume that the blue agents always start on the right side of the environment (see Figure 5). To facilitate the description of the evolved policy, we added an id to each node in the DTs.

We show the first agent in Figure 10. Analyzing the structure of the tree, we can observe how the agent moves in the environment. This agent tries to encircle the enemy, moving up to the left (id 24) until it perceives that enough enemies are below or to the right (ids 6 and 18 respectively). After having overrun the enemy, the agent turns back, then it moves towards the enemy, and eventually it starts to move to the right or down (ids 11, 21 25, and 22). Namely, the agent tries to flank the enemy moving to the top left of the map, to then turn and attack the enemy from behind.

Another interesting behavior of this agent relies on the fact that it tries not to be on the front lines: in fact, if there is a high density of allies in the same quadrant (id 14), it tends to move to the right (id 17), therefore from the direction from which its team started.

The attack actions are easy to understand: if an enemy is located in a certain cell, the agent simply attacks that cell. There are two particular cases. One is caused by the few visits of a certain leaf (id 9). The other one happens when there is an enemy above the agent (id 16): in this case, the agent tries to escape to the right (id 26), unless there is an obstacle in the above right cell (id 19), in which case it attacks the enemy (id 27). Since an obstacle can be either a wall or an ally, this particular condition leads to two different behaviors. If the obstacle is an ally, the agent helps to kill the opponent, otherwise it tries to escape to the right. If the obstacle is a wall, this means that the agent is located on the left side of that wall, since there is no possibility to have an opponent above while the agent is located next to a wall. This means that if there is a wall on the right, the agent cannot escape and has to fight. According to the behavior that this agent appears to have, this tells us that the agent tries to support other allies in the area, while it retreats if enemies are trying to surround it. Summarizing, this agent performs like a "cavalry wing", trying to overrun the enemy team to encircle them and attack from behind.

The second selected agent, presented in Figure 11, has a different behavior: while it tries to avoid being near friendly units, it always moves towards the enemy. Hence, we can describe this as an "interceptor", that actively moves in less populated areas to search for the enemy.

Other interesting behaviors emerge from the observation of the teams in the environment. A common behavior of the agents that, by chance, start closer to the opponent team is to go through

the gap in the inner walls to reach the enemies. There are also more complex behaviors. In some runs, it is possible to see some agents moving to the top of the environment, passing the inner walls from above, and then descending to hit the enemies they encounter. Much rarer is the reverse behavior, where agents pass the inner walls from below and then move up (this is possibly due to the fact that the passage on the bottom of the environment is narrower).

6. Conclusions and future works

In the past few years, the field of MARL has been gaining a lot of attention from the research community, due to its wide applicability to real-world scenarios. However, for such purposes, interpretability is a crucial requirement.

In this work, we studied different approaches for interpretable MARL, starting from our previous works on single-agent Interpretable RL. We studied different schemes, including non-coevolutionary, partially coevolutionary, and fully coevolutionary mappings. Overall, our results show that the coevolutionary approaches outperform the non-coevolutionary one. Moreover, the partially coevolutionary approach has the additional advantage of significantly reducing the number of parallel evolutionary processes, thus providing an opportunity for scaling more efficiently. However, it is worth noting that the number of parallel evolutionary processes can be used as a hyperparameter to trade-off between performance and computational cost.

Furthermore, we studied the impact of injecting human knowledge, in the form of a handcrafted policy, into the optimization process (which is one of the advantages of training interpretable models with respect to non-interpretable ones). The results show that introducing human knowledge in the evolutionary processes reduces the gap between the non-coevolutionary approach and the coevolutionary ones.

Future work includes testing the proposed method on other multi-agent tasks, self-optimizing the number of groups, using Quality-Diversity evolutionary algorithms, e.g. based on MAP-Elites, as done in [34, 28, 35, 36, 37, 38], Enki [39], or DOMiNO [40], and, finally, studying more sophisticated migration mechanisms.

Acknowledgments

Funded by the European Union (project no. 101071179). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or EISMEA. Neither the European Union nor the granting authority can be held responsible for them.

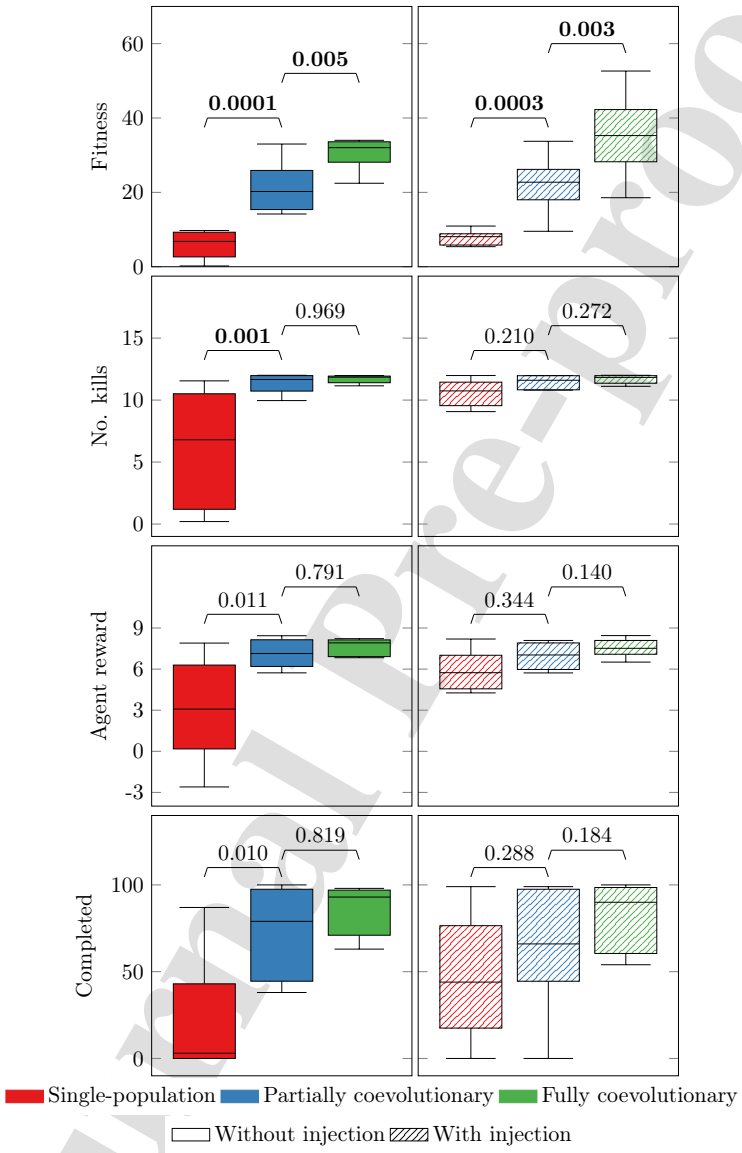


Figure 7: Comparison between all the methods on the metrics of interest. Bold numbers indicate statistically significant p-values.

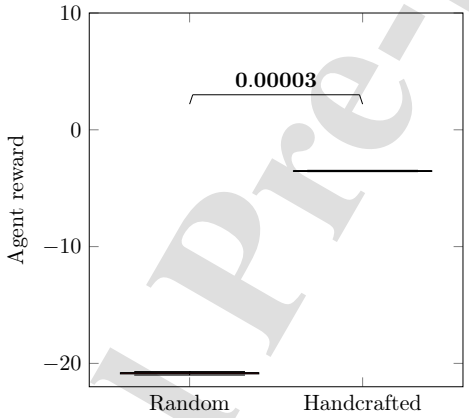


Figure 8: Comparison between random and handcrafted agents.

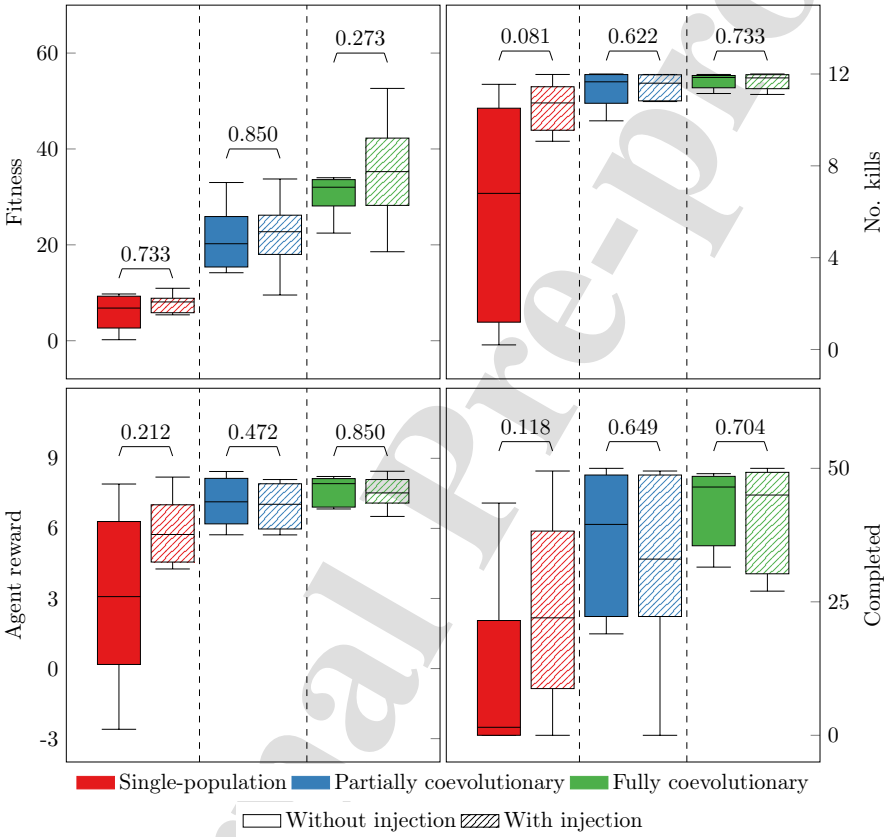


Figure 9: Comparison between all the methods on the metrics of interest.

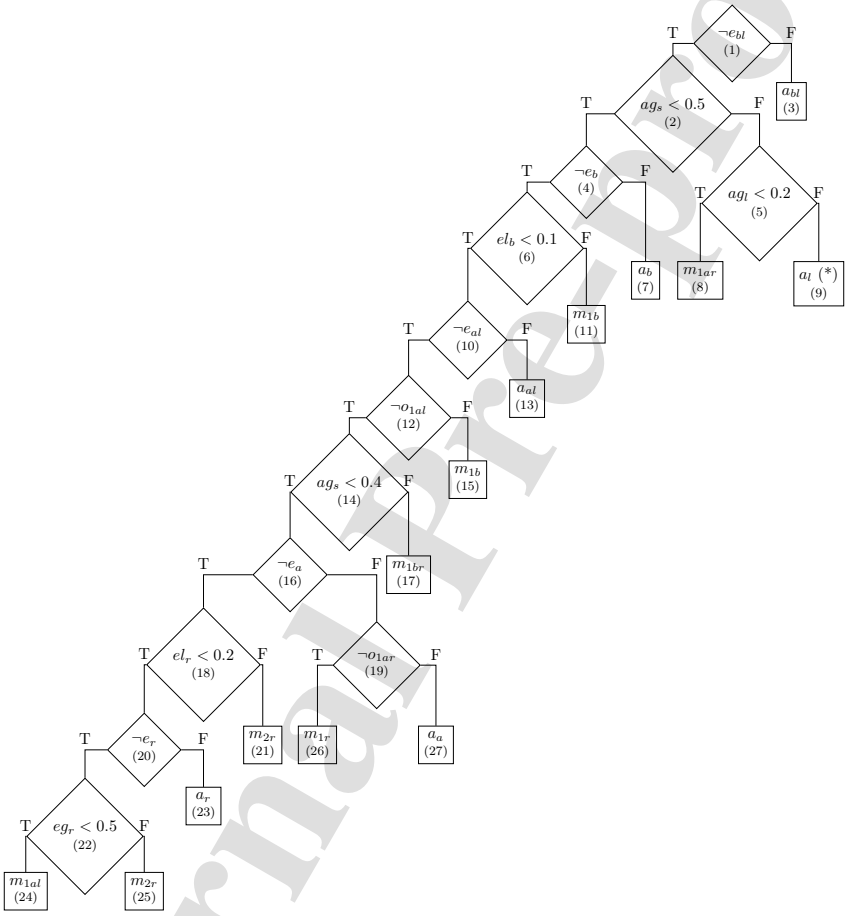


Figure 10: Example DT found in the fully coevolutionary setting. The "(*)" notation indicates that the leaf has been visited a number of times that is not sufficient to train it, thus it can be seen as a random action. The numbers in parentheses are the identifiers of the nodes (added for the sake of explanation).

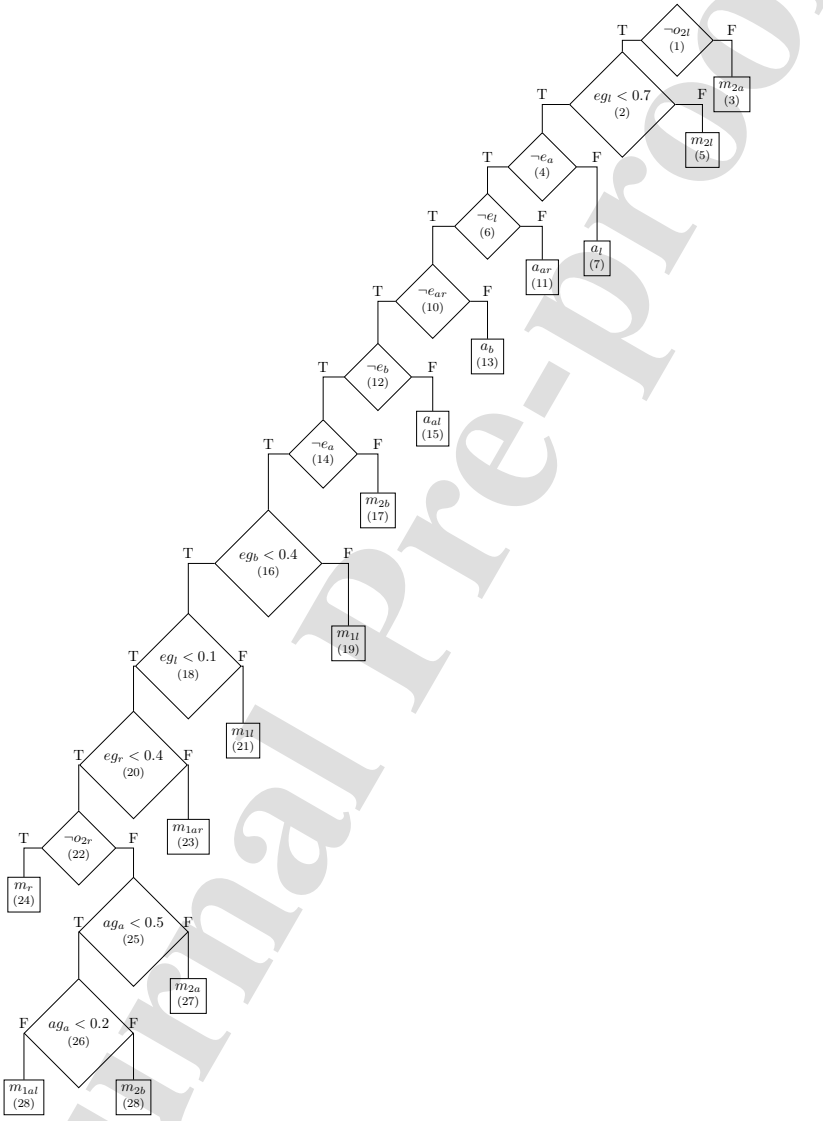


Figure 11: Example DT found in the partially coevolutionary setting. The "(*)" notation indicates that the leaf has been visited a number of times that is not sufficient to train it, thus it can be seen as a random action. The numbers in parentheses are the identifiers of the nodes (added for the sake of explanation).

References

[1] A. OroojlooyJadid, D. Hajinezhad, A review of cooperative multi-agent deep reinforcement learning, 2020. ArXiv:1908.03963.

[2] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, F. Herrera, Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI, Information Fusion 58 (2020) 82–115.

[3] L. L. Custode, G. Iacca, Evolutionary learning of interpretable decision trees, IEEE Access 11 (2023).

[4] C. Ryan, J. Collins, M. O. Neill, Grammatical evolution: Evolving programs for an arbitrary language, in: European Conference on Genetic Programming, Springer, Berlin, Heidelberg, 1998, pp. 83–96.

[5] C. J. C. H. Watkins, Learning from delayed rewards, Ph.D. thesis, King's College, Cambridge, United Kingdom, 1989.

[6] M. Crespi, L. L. Custode, G. Iacca, Towards interpretable policies in multi-agent reinforcement learning tasks, in: Bioinspired Optimization Methods and Their Applications: 10th International Conference, BIOMA 2022, Maribor, Slovenia, November 17–18, 2022, Proceedings, Springer, 2022, pp. 262–276.

[7] L. Busoniu, R. Babuska, B. De Schutter, A Comprehensive Survey of Multiagent Reinforcement Learning, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 38 (2008) 156–172.

[8] P. Stone, M. Veloso, Multiagent Systems: A Survey from a Machine Learning Perspective, Technical Report, Defense Technical Information Center, Fort Belvoir, VA, 1997.

[9] C. Yu, J. Liu, S. Nemati, Reinforcement Learning in Healthcare: A Survey, 2020. ArXiv:1908.08796.

[10] J. Bacardit, A. E. Brownlee, S. Cagnoni, G. Iacca, J. McCall, D. Walker, The intersection of evolutionary computation and explainable ai, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2022, pp. 1757–1762.

[11] T. W. Sandholm, R. H. Crites, On multiagent Q-learning in a semi-competitive domain, in: Adaption and Learning in Multi-Agent Systems, volume 1042, Springer, Berlin, Heidelberg, 1996, pp. 191–205.

[12] M. L. Littman, Markov games as a framework for multi-agent reinforcement learning, in: Machine Learning Proceedings 1994, Morgan Kaufmann, San Francisco (CA), 1994, pp. 157 – 163.

[13] M. Tan, Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, pp. 330–337.

[14] M. Lauer, M. A. Riedmiller, An algorithm for distributed reinforcement learning in cooperative multi-agent systems, in: International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000, p. 535–542.

[15] T. Fuji, K. Ito, K. Matsumoto, K. Yano, Deep multi-agent reinforcement learning using DNN-weight evolution to optimize supply chain performance, in: Hawaii International Conference on System Sciences, HICSS, Honolulu, HI, USA, 2018, pp. 1278–1287.

[16] L. Matignon, G. J. Laurent, N. Le Fort-Piat, Hysteretic Q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams, in: International Conference on Intelligent Robots and Systems, IEEE/RSJ, New York, NY, USA, 2007, pp. 64–69.

[17] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, J. Vian, Deep decentralized multi-task multi-agent reinforcement learning under partial observability, in: International Conference on Machine Learning, JMLR, Sydney, NSW, Australia, 2017, pp. 2681–2690.

[18] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, R. Vicente, Multiagent Cooperation and Competition with Deep Reinforcement Learning, 2015. ArXiv:1511.08779.

[19] X. Chu, H. Ye, Parameter Sharing Deep Deterministic Policy Gradient for Cooperative Multi-agent Reinforcement Learning, 2017. ArXiv:1710.00336.

[20] A. Singh, T. Jain, S. Sukhbaatar, Learning when to communicate at scale in multiagent cooperative and competitive tasks, 2018. ArXiv:1812.09755.

[21] S. V. Macua, A. Tukiainen, D. G.-O. Hernández, D. Baldazo, E. M. de Cote, S. Zazo, Diff-DAC: Distributed actor-critic for average multitask deep reinforcement learning, 2019. ArXiv:1710.10363.

[22] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, T. Graepel, Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward, in: International Conference on Autonomous Agents and MultiAgent Systems, Stockholm, Sweden, 2018, pp. 2085–2087.

[23] J. Yang, A. Nakhaei, D. Isele, K. Fujimura, H. Zha, CM3: Cooperative Multi-goal Multi-stage Multi-agent Reinforcement Learning, 2020. ArXiv:1809.05188.

[24] T. Haynes, R. L. Wainwright, S. Sen, D. A. Schoenefeld, Strongly Typed Genetic Programming in Evolving Cooperation Strategies, in: International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995, pp. 271–278.

[25] M. Virgolin, A. De Lorenzo, E. Medvet, F. Randone, Learning a formula of interpretability to learn interpretable formulas, in: Parallel Problem Solving from Nature, Springer International Publishing, Cham, 2020, pp. 79–93.

[26] P. Barceló, M. Monet, J. Pérez, B. Subercaseaux, Model interpretability through the lens of computational complexity, Advances in Neural Information Processing Systems 33 (2020).

[27] L. L. Custode, G. Iacca, A co-evolutionary approach to interpretable reinforcement learning in environments with continuous action spaces, in: 2021 IEEE Symposium Series on Computational Intelligence (SSCI), 2021, pp. 1–8.

[28] A. Ferigo, L. L. Custode, G. Iacca, Quality diversity evolutionary learning of decision trees, 2022. ArXiv:2208.12758.

[29] A. Hallawa, T. Born, A. Schmeink, G. Dartmann, A. Peine, L. Martin, G. Iacca, A. E. Eiben, G. Ascheid, EVO-RL: Evolutionary-Driven Reinforcement Learning, 2020. ArXiv:2007.04725.

[30] A. Hallawa, G. Iacca, C. Sariman, T. Rahman, M. Cochez, G. Ascheid, Morphological evolution for pipe inspection using robot operating system (ROS), Materials and Manufacturing Processes 35 (2020) 714–724.

[31] A. Hallawa, S. Schug, G. Iacca, G. Ascheid, Evolving instinctive behaviour in resource-constrained autonomous agents using grammatical evolution, in: Applications of Evolutionary Computation: 23rd European Conference, EvoApplications 2020, Held as Part of EvoStar 2020, Seville, Spain, April 15–17, 2020, Proceedings 23, Springer International Publishing, 2020, pp. 369–383.

[32] L. Zheng, J. Yang, H. Cai, M. Zhou, W. Zhang, J. Wang, Y. Yu, MAgent: A many-agent reinforcement learning platform for artificial collective intelligence, Proceedings of the AAAI Conference on Artificial Intelligence 32 (2018) 8222–8223.

[33] J. K. Terry, B. Black, M. Jayakumar, A. Hari, R. Sullivan, L. Santos, C. Dieffendahl, N. L. Williams, Y. Lokesh, C. Horsch, et al., Pettingzoo: Gym for multi-agent reinforcement learning, 2020. ArXiv:2009.14471.

[34] J. K. Pugh, L. B. Soros, K. O. Stanley, Quality diversity: A new frontier for evolutionary computation, Frontiers in Robotics and AI 3 (2016) 40.

[35] E. Zardini, D. Zappetti, D. Zambrano, G. Iacca, D. Floreano, Seeking quality diversity in evolutionary co-design of morphology and control of soft tensegrity modular robots, in: Genetic and Evolutionary Computation Conference, 2021, pp. 189–197.

[36] J. Nordmoen, F. Veenstra, K. O. Ellefsen, K. Glette, MAP-Elites enables powerful stepping stones and diversity for modular robotics, Frontiers in Robotics and AI 8 (2021) 1–17.

[37] B. Lim, L. Grillotti, L. Bernasconi, A. Cully, Dynamics-aware quality-diversity for efficient learning of skill repertoires, in: 2022 International Conference on Robotics and Automation (ICRA), 2022, pp. 5360–5366.

[38] B. Tjanaka, M. C. Fontaine, J. Togelius, S. Nikolaidis, Approximating gradients for differentiable quality diversity in reinforcement learning, 2022. ArXiv:2202.03666.

[39] M. A. Langford, B. H. Cheng, Enki: A diversity-driven approach to test and train robust learning-enabled systems, ACM Transactions on Autonomous and Adaptive Systems (TAAS) 15 (2021) 1–32.

[40] T. Zahavy, Y. Schroecker, F. Behbahani, K. Baumli, S. Flennerhag, S. Hou, S. Singh, Discovering policies with DOMiNO: Diversity optimization maintaining near optimality, 2022. ArXiv:2205.13521.

Highlights

- We propose an evolutionary algorithm for multi-agent reinforcement learning.
- We consider interpretable models based on decision trees.
- We demonstrate the method on a highly dynamic competitive multi-agent task.
- Our results show the effective applicability of the proposed method.

Marco Crespi: Conceptualization; Methodology; Software; Data curation; Investigation; Formal analysis; Writing - original draft.
Andrea Ferigo: Conceptualization; Methodology; Software; Visualization; Validation; Writing - original draft.
Leonardo Lucio Custode: Conceptualization; Methodology; Software; Writing - original draft.
Giovanni Iacca: Conceptualization; Methodology; Resources; Writing - review & editing.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: