Towards interpretable policies in multi-agent reinforcement learning tasks

Marco Crespi, Leonardo Lucio Custode[0000−0002−1652−1690], and Giovanni Iacca[0000−0001−9723−1830]

University of Trento
Department of Information Engineering and Computer Science
Via Sommarive 9, 38123 Povo (TN), Italy
marco.crespi@studenti.unitn.it
{leonardo.custode,giovanni.iacca}@unitn.it

Abstract. Deep Learning (DL) allowed the field of Multi-Agent Reinforcement Learning (MARL) to make significant advances, speeding-up the progress in the field. However, agents trained by means of DL in MARL settings have an important drawback: their policies are extremely hard to interpret, not only at the individual agent level, but also (and especially) considering the fact that one has to take into account the interactions across the whole set of agents. In this work, we make a step towards achieving interpretability in MARL tasks. To do that, we present an approach that combines evolutionary computation (i.e., grammatical evolution) and reinforcement learning (Q-learning), which allows us to produce agents that are, at least to some extent, understandable. Moreover, differently from the typically centralized DL-based approaches (and because of the possibility to use a replay buffer), in our method we can easily employ Independent Q-learning to train a team of agents, which facilitates robustness and scalability. By evaluating our approach on the Battlefield task from the MAgent implementation in the PettingZoo library, we observe that the evolved team of agents is able to coordinate its actions in a distributed fashion, solving the task in an effective way.

Keywords: Reinforcement learning · Multi-agent systems · Grammatical evolution · Interpretability

1 Introduction

In recent years, the application of Deep Learning (DL) to the field of Multi-Agent Reinforcement Learning (MARL) led to the achievement of significant results in the field. While DL allows to train powerful multi-agent systems (MASs), it has some drawbacks. First of all, to exploit state-of-the-art deep reinforcement learning (RL) algorithms, one often has to employ centralized approaches for training [1], which limits the scalability of the system, i.e., no agents can be added after the MAS has been trained. Moreover, deep RL methods suffer from

an even worse drawback: the lack of *interpretability*[1]. In fact, in safety-critical or high-stakes contexts, DL approaches cannot be employed as they are not fully predictable [3–5] and, thus, they may exhibit unexpected behaviors in edge cases. While interpretability in RL is an important concern, in MARL it is even more important. In fact, in contrast to traditional RL setups where safety can be assessed by inspecting the trained agent, in MARL not only do we need to analyze each agent, but we also need to understand their *collective behavior*.

In this paper, we employ a recently proposed methodology [6] (originally designed for single-agent tasks) for training an interpretable MAS. More specifically, we extend the setup proposed in [6] by creating a *cooperative co-evolutionary algorithm* [7] in which each evolutionary process addresses the evolution of an agent of the MAS. As a baseline, we also provide the results obtained when a single policy is trained for all the agents in the MAS. We evaluate our approach on the Battlefield task from MAgent [8] (implemented in the PettingZoo library [9]). The teams evolved with our approach are able to obtain promising performances, eliminating the whole opponent team in up to 98% of the cases.

So, the main contribution of this paper are: 1) the extension of the approach presented in [6] to multi-agent reinforcement learning settings; 2) the introduction of two approaches, a co-evolutionary one and single-policy one; 3) a validation of the proposed methods on the Battlefield task.

The rest of the paper is structured as follows. In the next section, we briefly overview the related work. In Section 3, we describe the proposed method. Then, in Section 4 and 5 we present the experimental setup and the numerical results, respectively. Finally, we draw the conclusions in Section 6.

2 Related work

In the following, we make a summary of the state-of-the-art in the field of MARL. For a more complete review, we refer the reader to [1, 2, 10–12].

In a preliminary work [11], the authors explained the advantages of adopting a multi-agent approach instead of a single, complex agent approach. Several approaches have then been proposed for MARL. In [13] the authors compared two function approximators in the iterated prisoner's dilemma: a table-based approach and a recurrent neural network (RNN). The experiments showed that the agents based on the tabular approach were more prone to cooperate than the ones trained using the RNN, indicating that the agents trained by using the tabular approach had learned a better approximation of the Q function. Littman [14] presented a novel algorithm based on Q-learning and minimax, named "minimax Q". This algorithm, in the experimental results, proved to be able to learn policies that were more robust than the policies learned by Q-learning. In [15] the authors made use of cooperative co-evolution with strongly-typed genetic programming (GP) to evolve agents for a predator-prey game. The evolved strategies were more effective than handcrafted policies.

[1] In the rest of this paper, we will define as an interpretable system one that can be *understood* and inspected by humans [2].

Independent Q-learning (IQL) [16] is another convenient approach to MARL, as it is scalable and decentralized. However, when using neural networks as function approximators for reinforcement learning, this method cannot be applied. In fact, the need for a replay buffer does not make this method suitable in settings with neural networks. To mitigate this issue, several approaches have been proposed [17–21]. Other approaches circumvent this problem by using instead the actor-critic model [22–26].

Recently, some approaches have been proposed to measure the interpretability of a machine learning model. For instance, Virgolin et al., [27], propose a metric of interpretability based on the elements contained in the mathematical formula described by the model. In [28], the authors suggest that the computational complexity of the model can be used as a measure of interpretability. In this paper we follow this approach, assuming that less complex models are easier to interpret, see Section 5.1.

3 Method

The goal of our work is to produce interpretable agents that are capable of cooperate to solve a given task. To do that, we evolve populations of interpretable agents in the form of decision trees. To evolve these decision trees, we use the same approach that was recently proposed in [6, 29]. In particular, we use the Grammatical Evolution (GE) algorithm [30] to evolve a genotype made of integers that, by using a grammar translator, is converted into a decision tree. However, we do not build the full decision tree. Instead, we only build the inner structure (i.e., the tree without leaves). The reason behind this choice relies on the fact that we want to *exploit* at their best the rewards given by the environment, using them to train the state-action function embedded in the leaves. Moreover, using Q-learning allows the agents to refine (and modify) their behavior in real-time, without having to wait for the next generation to improve the performance, which is particularly useful in multi-agent settings.

Finally, it is important to note that our method employs a cooperative coevolutionary process [7], where each population optimizes the structure of the tree for a particular agent of the environment.

3.1 Creation of the teams

To evaluate a genotype, we have to assess the quality of the corresponding phenotype when placed inside a team. Each agent (i.e., a member of the team) has its own evolutionary process (i.e., there is a separate population for each agent in the task). Thus, we assemble teams composed of one phenotype (i.e., a genotype transformed into a decision tree) taken from each agent-local population.

Each agent-local population has N_{ind} individuals, such that N_{ind} different teams are created. Each i-th team is formed by the corresponding i-th individuals (one per each agent-local population), where i is an index $\in [0, N_{ind} - 1]$. This approach guarantees that each individual from each agent-local population is

evaluated exactly once. Note that the selection operator, when applied, shuffles the array of the individuals. This means that an individual from an agent-local population is generally not always evaluated with the same individuals taken from the other agent-local populations.

At the end of the evolutionary process, we form the final team by combining the best individuals from all the agent-local populations. Moreover, by using an adoption mechanism (described in Section 3.4), the structure of the best agents may be shared between different agent-local populations.

3.2 Fitness evaluation

Once a team is created, it undergoes N_{ep} episodes of simulation of the task. In the simulation phase, the agents perform IQL (with a dynamic ε-greedy exploration approach) to learn the function that maps the leaves to actions. By using IQL, each agent does not have to take into account the choices made by the other agents, as these are modeled as part of the environment. Moreover, given a sufficient number of episodes for the evaluation, the continuous learning of all the agents results in a co-adaptation. After the simulation phase, the seventh decile of the returns (i.e., the cumulative reward for each episode) received by an agent is used as fitness. The choice of the seventh decile lies on the fact that our fitness function is meant to describe the quality of a genotype as the quality of the state-space decomposition function [6], which can only be measured when the performance of the agent converges. While also the mean, the maximum, and the median have been considered as aggregation functions to compute the fitness, they have been discarded for the following reasons. Since the agents initially use a high ε for the exploration, the initial returns have a significant impact on the mean, thus they do not reflect the true quality of the genotype. Using the median would also present problems: on the one hand, the median would discard all the episodes in which the co-operation between the agents was fruitful enough to receive high returns; on the other hand, since we expect the returns to grow towards the end of the simulation phase, using the median would mean that we take into account the performance of a not-fully-trained agent. Finally, if we used the maximum to aggregate the returns, we would give too much importance to spurious good performance that may occur in the simulation (e.g., returns obtained just by randomly effective behaviors), without taking into account the performance of the trained version of the agent. In a preliminary experimental phase, the seventh decile represented a good trade-off between the median and the maximum, reflecting more closely the performance of the agents. The fitness evaluation process is described in Figure 1.

3.3 Individual encoding

Each individual is represented as a list of integers, where each integer indicates the production to choose for the current rule (modulo the number of productions). Unlike the original version of GE, we do not use variable-length genotypes. Instead, the genotype is a list that has fixed length. The process used to create

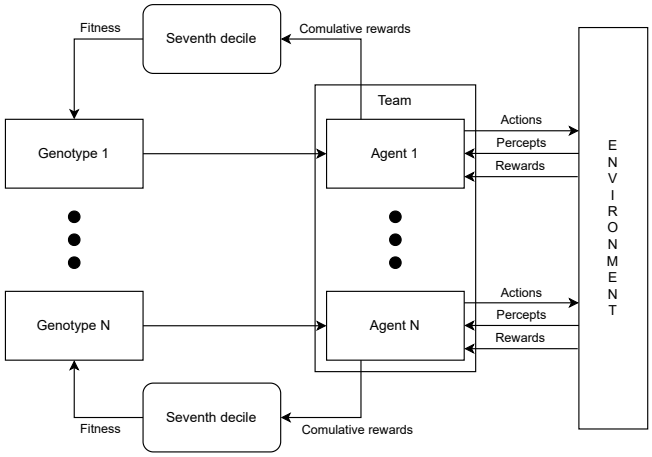

Fig. 1. Block diagram of the fitness evaluation process.

decision trees from genotypes (i.e., lists of integers) is the following. Starting from the first integer of the list, we apply the first (i.e., leftmost) non-expanded rule from the current phenotype by using the production rule indicated by the current integer (modulo the number of productions). The start symbol for the grammar is called "dt". The process can terminate in two different ways, depending on the case: (a) The phenotype does not contain any non-expanded rule: in this case, the phenotype is simply returned; or (b) all the parameters from the genotype have been converted into productions, but there are still non-expanded rules: in this case, the missing branches of the trees are linked to novel leaves.

3.4 Operators

Mutation The mutation operator used in this work is the uniform mutation. This operator mutates each gene of the genotype with a probability p_{gene}. When a gene of the genotype is selected for mutation, its next value is selected uniformly $\in [0, M]$, where M is a number significantly bigger than the maximum number of choices in the grammar, to ensure that the productions are approximately uniformly distributed.

Crossover The crossover operator used in this work is the one-point crossover operator. This operator simply chooses a random splitting point for the two fixed-length genotypes. Then, it produces two offspring by mixing the two substrings of the genotypes.

Selection The individuals are selected by means of a tournament selection. This operator creates N_{ind} "tournaments" (i.e., random groups of s_t individuals taken from an agent-local population). Then, for each tournament, the best individual is selected to create the population for the next generation.

Replacement Individuals in each population are replaced by their offspring, (obtained through mutation or crossover) when the new individuals perform better than their parents. If an individual is obtained through mutation, it will replace its parent only if it reaches a better fitness. In case of crossover (which involves two individuals and two parents) the individual of the offspring with the best fitness replaces the parent with the worst fitness. This mechanism also allows to systematically discard "adopted" individuals (see the next paragraph) that perform worse than their parents in the new population.

Adoption The adoption of an individual happens at the end of each generation. An agent-local population is randomly chosen and its individual with the highest fitness is selected. At this point, the selected individual is copied into the other agent-local populations, replacing a randomly selected individual from the offspring. The adopted individual's parents are then assigned to the replaced individual's parents. The reason why we use this adoption mechanism lies in the reward system of the specific BattleField environment (see Section 4.1). As mentioned by the authors of PettingZoo: "Agents are rewarded for their individual performance, and not for the performance of their neighbors, so coordination is difficult"[2]. This means that only agents capable of hitting or killing enemies (this will become clearer in the next section) obtain a high fitness and the adoption mechanism allows sharing "knowledge" across agent-local populations.

4 Experimental setup

4.1 Environment

We simulate a multi-agent environment by using the PettingZoo library [9]. More specifically, we use the Battlefield environment from the MAgent [8] suite. A screenshot of the environment is shown in Figure 2.

In this task, there are two teams: the red team and the blue team. As the name of the environment suggests, the goal of each team is to defeat the other team by killing all of its members. The environment is an 80×80 grid. To win the battle, the agents have to learn to collaborate with their team in order to eliminate the enemies, and to move through the map to overcome walls and obstacles.

Each agent has a perceptive field of 13×13 squares and can either move or attack at each turn. The agents' perception is composed of: local presence/absence of an obstacle in a square; local presence/absence of a teammate/enemy in a square; health points (hp) of the teammate/enemy in a square; global density of teammates/enemies. A square represents a 7×7 quadrant of the environment. Note that each agent's local perception area corresponds to a circle with a radius of six squares around that agent. Moreover, to simplify the learning phase (and the interpretability of the agents evolved), we perform a pre-processing of these

[2] https://www.pettingzoo.ml/magent/battlefield (accessed on 02/02/2022).

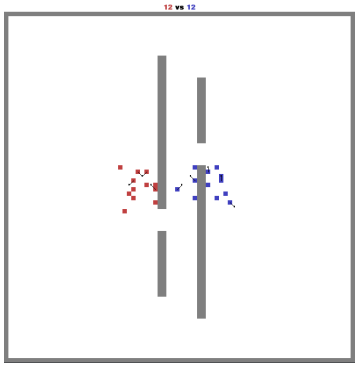

Fig. 2. A screenshot from the Battlefield environment.

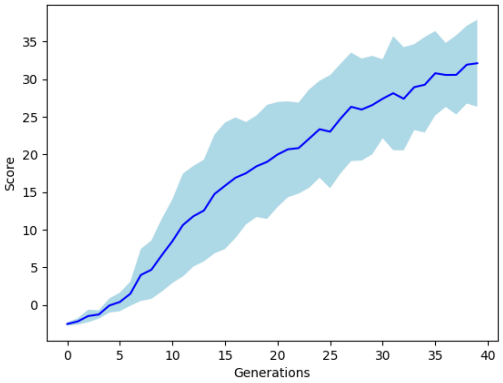

Fig. 3. Maximum return (average ± std. dev. across 10 runs of the proposed co-evolutionary approach) at each generation.

features, based on domain knowledge, in order to obtain higher-level features that are then fed as inputs to the decision tree. The selected features, extracted from the raw observations, are reported in Table 1. The "Abbreviation" column shows the abbreviation that we will use throughout the text to refer to a specific feature.

Both local and global density are calculated based on the active agents in the environment, i.e., killed agents are not taken into account.

Each agent initially has 10 hp. When an agent attacks another agent (called target), the target's hp are decreased by 2 hp. Moreover, each turn increases the agents' health points by 0.1 hp (unless the agent already has already 10 hp).

An agent, at each step, can perform 21 discrete actions: no action; move to any of the 8 adjacent squares; move to two squares on either left, right, up, down; attack any of the 8 adjacent squares.

Table 2 shows the action that can be performed by the agent. As in Table 1, the "Abbreviation" column shows how we refer to the actions in the remainder of the text. The rewards obtained by the environment are the following: 5 points if the agent kills an opponent; -0.005 points for each timestep (a time penalty, thus the quicker the team wins, the higher the reward); -0.1 for attacking (to make the agent attack only when necessary); 0.9 when the agent hits an opponent (to give a quicker feedback to the agent, without having to wait for killing an agent to obtain a positive reward that encourages hitting enemies); -0.1 if the agent dies. At each timestep, the agent receives a combination of these rewards based on the events that happened in the last timestep. For instance, if an agent attacks and hits an enemy, it obtains a total reward of $r = 0.9 - 0.1 - 0.005$.

While there is no reward for collaboration, we decided to not alter the reward function to encourage it, to preserve the original configuration of the environment. Note that we evolve only one of the two teams (the blue one), while the other team (the red one) uses a random behavior for all the agents. This choice

Table 1. Extracted features, their abbreviation and their domain.

Feature	Abbreviation	Domain
Obstacle 2 squares above	o_{2a}	{0, 1}
Obstacle 2 squares left	o_{2l}	{0, 1}
Obstacle 2 squares right	o_{2r}	{0, 1}
Obstacle 2 squares below	o_{2b}	{0, 1}
Obstacle 1 square above-left	o_{1al}	{0, 1}
Obstacle 1 square above	o_{1a}	{0, 1}
Obstacle 1 square above-right	o_{1ar}	{0, 1}
Obstacle 1 square left	o_{1l}	{0, 1}
Obstacle 1 squares right	o_{1r}	{0, 1}
Obstacle 1 square below-left	o_{1bl}	{0, 1}
Obstacle 1 squares below	o_{1b}	{0, 1}
Obstacle 1 squares 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 2. Actions that the agent can perform.

Action	Abbreviation
Move 2 squares above	m_{2a}
Move 1 square above-left	m_{1al}
Move 1 square above	m_{1a}
Move 1 square above-right	m_{1ar}
Move 2 squares left	m_{2l}
Move 1 square left	m_{1l}
No action	m_n
Move 1 squares right	m_{1r}
Move 2 squares right	m_{2r}
Move 1 square below-left	m_{1bl}
Move 1 squares below	m_{1b}
Move 1 squares below-right	m_{1br}
Move 2 squares 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}

has been made in order to provide a non-biased baseline policy, i.e., to prevent the evolved policies from overfitting to a specific handmade policy for the red team. Furthermore, we decided not to competitively co-evolve the policies for both teams (blue and red) to reduce the complexity of the evolutionary process, and focus on the interpretability of the evolved policy for the blue team. We reserve this kind of investigations for future works. For each fitness evaluation, N_{ep} episodes are simulated, each of 500 timesteps.

4.2 Parameters

The parameters used for GE and Q-learning are shown in Table 3. To ensure that the Q function tends to the optimal one, we employ a learning rate of $\alpha = \frac{1}{v}$, where v is the number of visits made to the state-action pair [31]. The grammar for the GE algorithm is shown in Table 4. Note that we constrain the grammar to evolve orthogonal decision trees, i.e., decision trees whose conditions are in the form $x < c$, where x is a variable and c is a constant.

Table 3. Parameters used for the two algorithms (Grammatical Evolution 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 4. Grammar used to evolve the decision trees. "|" denotes the possibility to choose between different productions; "dt" indicates the start 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

5 Experimental results

We perform 10 independent evolutionary runs to evolve the policy of each agent in the blue team. Figure 3 shows the average maximum return (across the 10 runs) during the evolutionary process generation. The shaded area indicates the standard deviation across runs. We should note that while the average trend did not reach yet a plateau after the considered number of generations, we had to limit the total duration of our runs due to constraints on the available computational resources. On average, one full run of our approach takes approximately 30 hours on a 16-core machine with parallelization at the level of the individual evaluation.

Since the goal of the task is the elimination of the opponent team, we use two metrics to analyze the results in a post-hoc test phase (i.e., after the evolutionary process): the number of opponents killed, and the agents' returns over 100 unseen episodes. Table 5 shows the results of this test phase. For each of the 10 evolutionary runs, we report the statistics obtained with a team composed of the best agents (one for each population) evolved in that run over unseen episodes. The "Team kills" row shows the descriptive statistics of the 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 "Agents' returns" row shows the descriptive statistics of the average returns of all the agents in the team. The "Completed" column shows the percentage of episodes in which the team was able to eliminate the entire opponent team.

We observe that, for most runs, the obtained teams are able to complete the task (i.e., kill all the enemies) in most cases. In fact, the average number of kills is very close to the maximum achievable value and the standard deviation confirms that the behaviour of the teams is quite consistent.

Table 5. Summary of the test results (co-evolutionary approach).

Run	Type	Mean	Std	Best	Worst	Completed
1	Team kills	11.96	0.20	12	11	96.0%
	Agents' returns	7.92	0.93	9.08	4.01	
2	Team kills	11.85	0.62	12	8	94.0%
	Agents' returns	8.06	0.93	8.96	3.33	
3	Team kills	11.98	0.14	12	11	98.0%
	Agents' returns	8.20	0.56	9.09	5.10	
4	Team kills	11.97	0.22	12	10	98.0%
	Agents' returns	7.85	0.94	9.00	2.16	
5	Team kills	11.81	0.73	12	7	91.0%
	Agents' returns	8.09	1.09	9.21	3.10	
6	Team kills	11.91	0.71	12	5	97.0%
	Agents' returns	8.17	0.82	8.94	1.92	
7	Team kills	8.98	1.60	12	4	1.0%
	Agents' returns	4.43	1.19	8.22	1.02	
8	Team kills	11.65	0.77	12	9	79.0%
	Agents' returns	6.83	2.13	9.20	0.07	
9	Team kills	11.15	1.46	12	5	63.0%
	Agents' returns	7.00	1.60	9.38	2.29	
10	Team kills	11.9	0.46	12	8	93.0%
	Agents' returns	8.22	0.95	8.95	3.14	

5.1 Interpretation

In this section we practically demonstrate the interpretability of the obtained agents.

Figure 4 shows the decision tree of one of the agents evolved in one of the evolutionary runs presented before. For space reasons, we cannot present all the evolved agents from each run. However, similar considerations apply also to the other evolved agents in the various runs.

By reading the decision tree in the figure, we can describe how the agent moves in the environment. In the following, please remember that we evolve only the blue agents' behavior, and that these agents always start on the right side of the environment (see Figure 2). To facilitate the description of the evolved policy, we added an id to each node in the decision tree.

The selected agent moves up to the left (id 24) until the local density of enemies below (id 6) or to the right (id 18) reaches a certain threshold. In both cases the agent changes the direction and moves towards the enemies (ids 11 and 21). It also moves to the right (id 25) if there is a high global density of enemies on its right (id 22). This means that this agent moves to the top left of the map and intercepts the enemies it finds in that area. Another interesting behaviour of this agent relies in 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. This

agent appears to behave like a "wing": it moves above the enemies and tries to eliminate the ones that try to move in the space between it and the allies below.

The attack actions are easy to understand: if an enemy is located in a certain square, the agent simply attacks that square. There are two particular cases. One is caused by the few visits of the 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 square (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 on the right. If the obstacle is present and 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 role that this agent appears to have, this behavior tells us that the agent tries to support other allies in the area, while it retreats if enemies are trying to surround them.

This behavior is quite common, in fact in every run agents can be seen that move to the top left of the environment and capture the enemies in that area.

Other interesting behaviours emerge from the observation of the teams in the environment. A common behavior of the agents starting closer to the opponent team is to go through the gap in the 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 walls from above and then descending to hit the enemies they encounter. Much rarer is the reverse behavior, where agents pass the walls from below and then move up.

5.2 Comparison with a non co-evolutionary approach

To provide a baseline for the proposed co-evolutionary approach, we also performed experiments in the same environment using a single phenotype for the entire team, i.e., by cloning the phenotype and assigning it to each member of the team. The parameters used in these experiments are the same shown in Table 3. Note that in this case each agent in the team shares the same decision tree structure, but each one develops its own leaf by using IQL.

In this case, the fitness evaluation is realized using the average of the seventh decile of the returns obtained by each agent over the training episodes. This choice is motivated by the following rationale. Since the structure of the agents is shared, we must favor the phenotype that, besides guaranteeing a high number of kills, also gives high importance to agents that do not kill any enemy.

Table 6 shows the test results obtained by the agents evolved in each of 10 runs over 100 unseen episodes. We can compare these results with the ones obtained in the co-evolutionary setup (shown in Table 5) by using the number of kills at test time. In this regard, we observe that there is a large difference in performance between the two setups, being the co-evolutionary clearly superior. This indicates that, even though the agents have similar goals in both setups, the co-evolutionary setup can indeed find much better solutions. This may be due

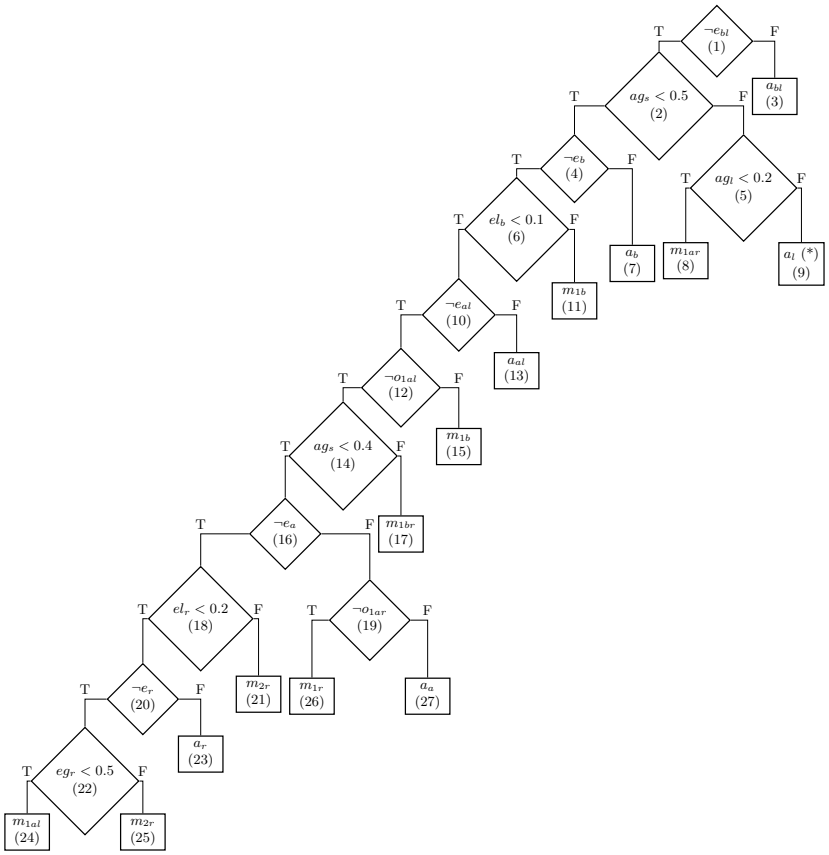

Fig. 4. Decision tree of the selected agent. The "(*)" notation indicates that the leaf has been visited a number of times that is not sufficient to train it, thus it can be seen as a random action. The numbers in parentheses are the identifiers of the nodes.

to the fact that the adoption mechanism used in the co-evolutionary approach allows for a quicker spreading of high-performing genotypes in the populations.

Another observation concerns the completion percentage: by looking at it, it appears that the performance of the non co-evolutionary is much less robust across runs. This suggests that the performance of this setup is heavily impacted by the initialization, with only a few occasional runs achieving a satisfactory completion percentage. A possible improvement of the non co-evolutionary setup would be to include an ad hoc method, e.g. based on domain knowledge, to provide a smarter initialization. We will consider this possibility in future works.

6 Conclusions and future works

Interpretability in AI is becoming a matter of concern for its applications in safety-critical and high-stakes scenarios. In MAS, this need is even stronger, and

Table 6. Summary of the test results (non co-evolutionary approach).

Run	Type	Mean	Std	Best	Worst	Completed
1	Team kills	0.51	0.74	3	0	0.0%
	Agents' returns	-2.60	0.72	0.13	-3.47	
2	Team kills	1.87	2.23	10	0	0.0%
	Agents' returns	-1.15	1.66	4.27	-3.42	
3	Team kills	11.03	1.20	12	6	45.0%
	Agents' returns	6.59	1.45	9.45	2.60	
4	Team kills	11.55	1.33	12	5	87.0%
	Agents' returns	7.90	1.48	9.88	1.96	
5	Team kills	8.08	2.81	12	1	14%
	Agents' returns	3.71	2.27	8.13	-2.23	
6	Team kills	9.99	2.27	12	4	41.0%
	Agents' returns	6.00	1.95	9.22	0.81	
7	Team kills	4.24	1.93	10	0	0.0%
	Agents' returns	1.50	1.44	5.46	-1.99	
8	Team kills	11.16	2.01	12	2	81.0%
	Agents' returns	7.30	190	10.02	-0.25	
9	Team kills	0.20	0.57	2	0	0.0%
	Agents' returns	-3.08	0.48	-1.32	-3.56	
10	Team kills	6.8	2.33	12	3	3.0%
	Agents' returns	3.00	1.74	7.29	-0.50	

achieving it is even more challenging. In fact, in MAS, besides the need for the interpretability of the agents, also the interpretability of their interactions is important. In this paper, we proposed a co-evolutionary approach to interpretable RL in MARL settings. We evaluated our approach on the Battlefield environment from MAgent, obtaining promising results in most of the runs. In contrast, a non co-evolutionary approach obtained poorer performance.

Future work includes: 1) evaluating the proposed approach on different tasks; 2) introducing the possibility of communication between agents (both symbolic [32] and sub-symbolic [33]); 3) designing more efficient methodologies for training interpretable MARL systems, including for instance using other RL algorithms (different from Q-learning), and comparing them with existing methods, as well as handmade problem-specific policies; and 4) performing a sensitivity analysis for the proposed method.

References

1. OroojlooyJadid, A., Hajinezhad, D.: A review of cooperative multi-agent deep reinforcement learning (2020) arXiv:1908.03963.
2. Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., Herrera, F.: Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information Fusion **58** (June 2020) 82–115

3. Rudin, C.: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature Machine Intelligence **1**(5) (May 2019) 206–215
4. Rudin, C., Radin, J.: Why Are We Using Black Box Models in AI When We Don't Need To? A Lesson From An Explainable AI Competition. Harvard Data Science Review **1**(2) (November 2019)
5. Rudin, C., Chen, C., Chen, Z., Huang, H., Semenova, L., Zhong, C.: Interpretable Machine Learning: Fundamental Principles and 10 Grand Challenges (July 2021) arXiv:2103.11251.
6. Custode, L.L., Iacca, G.: Evolutionary learning of interpretable decision trees (2020)
7. Potter, M.A., De Jong, K.A.: A cooperative coevolutionary approach to function optimization. In: Parallel Problem Solving from Nature, Berlin, Heidelberg, Springer (1994) 249–257
8. Zheng, L., Yang, J., Cai, H., Zhou, M., Zhang, W., Wang, J., Yu, Y.: Magent: A many-agent reinforcement learning platform for artificial collective intelligence. Proceedings of the AAAI Conference on Artificial Intelligence **32**(1) (April 2018) 8222–8223
9. Terry, J.K., Black, B., Jayakumar, M., Hari, A., Sullivan, R., Santos, L., Dieffendahl, C., Williams, N.L., Lokesh, Y., Horsch, C., et al.: Pettingzoo: Gym for multi-agent reinforcement learning (2020) arXiv:2009.14471.
10. Busoniu, L., Babuska, R., De Schutter, B.: A Comprehensive Survey of Multiagent Reinforcement Learning. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **38**(2) (March 2008) 156–172
11. Stone, P., Veloso, M.: Multiagent Systems: A Survey from a Machine Learning Perspective:. Technical report, Defense Technical Information Center, Fort Belvoir, VA (December 1997)
12. Yu, C., Liu, J., Nemati, S.: Reinforcement Learning in Healthcare: A Survey (April 2020) arXiv:1908.08796.
13. Sandholm, T.W., Crites, R.H.: On multiagent Q-learning in a semi-competitive domain. In: Adaption and Learning in Multi-Agent Systems. Volume 1042. Springer, Berlin, Heidelberg (1996) 191–205
14. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: Machine Learning Proceedings 1994. Morgan Kaufmann, San Francisco (CA) (1994) 157 – 163
15. Haynes, T., Wainwright, R.L., Sen, S., Schoenefeld, D.A.: Strongly Typed Genetic Programming in Evolving Cooperation Strategies. In: International Conference on Genetic Algorithms, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (July 1995) 271–278
16. Tan, M. In: Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1997) 487–494
17. Lauer, M., Riedmiller, M.A.: An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In: International Conference on Machine Learning, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2000) 535–542
18. Fuji, T., Ito, K., Matsumoto, K., Yano, K.: Deep multi-agent reinforcement learning using dnn-weight evolution to optimize supply chain performance. In: Hawaii International Conference on System Sciences, Honolulu, HI, USA, HICSS (2018) 1278–1287

19. Omidshafiei, S., Pazis, J., Amato, C., How, J.P., Vian, J.: Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In: International Conference on Machine Learning, Sydney, NSW, Australia, JMLR.org (August 2017) 2681–2690
20. Matignon, L., Laurent, G.J., Le Fort-Piat, N.: Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In: International Conference on Intelligent Robots and Systems, New York, NY, USA, IEEE/RSJ (2007) 64–69
21. Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., Vicente, R.: Multiagent Cooperation and Competition with Deep Reinforcement Learning (November 2015) arXiv:1511.08779.
22. Chu, X., Ye, H.: Parameter Sharing Deep Deterministic Policy Gradient for Cooperative Multi-agent Reinforcement Learning (October 2017) arXiv:1710.00336.
23. Singh, A., Jain, T., Sukhbaatar, S.: Learning when to communicate at scale in multiagent cooperative and competitive tasks (2018) arXiv:1812.09755.
24. Macua, S.V., Tukiainen, A., Hernández, D.G.O., Baldazo, D., de Cote, E.M., Zazo, S.: Diff-dac: Distributed actor-critic for average multitask deep reinforcement learning (2019) arXiv:1710.10363.
25. Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W.M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J.Z., Tuyls, K., Graepel, T.: Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In: International Conference on Autonomous Agents and MultiAgent Systems, Stockholm, Sweden, International Foundation for Autonomous Agents and Multiagent Systems (July 2018) 2085–2087
26. Yang, J., Nakhaei, A., Isele, D., Fujimura, K., Zha, H.: CM3: Cooperative Multi-goal Multi-stage Multi-agent Reinforcement Learning (January 2020) arXiv:1809.05188.
27. Virgolin, M., De Lorenzo, A., Medvet, E., Randone, F.: Learning a formula of interpretability to learn interpretable formulas. In Bäck, T., Preuss, M., Deutz, A., Wang, H., Doerr, C., Emmerich, M., Trautmann, H., eds.: Parallel Problem Solving from Nature, Cham, Springer International Publishing (2020) 79–93
28. Barceló, P., Monet, M., Pérez, J., Subercaseaux, B.: Model interpretability through the lens of computational complexity. Advances in Neural Information Processing Systems **33** (2020)
29. Custode, L.L., Iacca, G.: A co-evolutionary approach to interpretable reinforcement learning in environments with continuous action spaces. In: 2021 IEEE Symposium Series on Computational Intelligence (SSCI). (December 2021) 1–8
30. Ryan, C., Collins, J., Neill, M.O.: Grammatical evolution: Evolving programs for an arbitrary language. In: European Conference on Genetic Programming. Springer, Berlin, Heidelberg (1998) 83–96
31. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA (2018)
32. Foerster, J., Assael, I.A., de Freitas, N., Whiteson, S.: Learning to communicate with deep multi-agent reinforcement learning. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R., eds.: Advances in Neural Information Processing Systems. Volume 29., Curran Associates, Inc. (2016)
33. Lotito, Q.F., Custode, L.L., Iacca, G.: A signal-centric perspective on the evolution of symbolic communication. In: Proceedings of the Genetic and Evolutionary Computation Conference. Association for Computing Machinery, New York, NY, USA (June 2021) 120–128