IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Evolutionary learning of interpretable decision trees

LEONARDO L. CUSTODE, GIOVANNI IACCA (MEMBER, IEEE) [2]
Department of Information Engineering and Computer Science, University of Trento, Via Sommarive 9, 38123 Povo (Trento), Italy
(e-mail: {leonardo.custode,giovanni.iacca}@unitn.it)

Corresponding author: Leonardo L. Custode (e-mail: leonardo.custode@unitn.it).

ABSTRACT In the last decade, reinforcement learning (RL) has been used to solve several tasks with human-level performance. However, there is a growing demand for interpretable RL, i.e., there is the need to understand how a RL agent works and the rationale of its decisions. Not only do we need interpretability to assess the safety of such agents, but also we may need it to gain insights into unknown problems. In this work, we propose a novel optimization approach to interpretable RL that builds decision trees. While techniques that optimize decision trees for RL do exist, they usually employ greedy algorithms or do not exploit the rewards given by the environment. For these reasons, these techniques may either get stuck in local optima or be inefficient. On the contrary, our approach is based on a two-level optimization scheme that combines the advantages of evolutionary algorithms with the benefits of Q-learning. This method allows decomposing the problem into two sub-problems: the problem of finding a meaningful decomposition of the state space, and the problem of associating an action to each subspace. We test the proposed method on three well-known RL benchmarks, as well as on a pandemic control task, on which it results competitive with the state-of-the-art in both performance and interpretability.

INDEX TERMS Decision Tree, Evolutionary algorithm, Interpretability, Reinforcement Learning

I. INTRODUCTION

WHILE machine learning (ML) is achieving promising results in various fields of application, there is an emergent need to understand what happens in the learned model, for testing, security, and safety purposes. In fact, as pointed out by the UNESCO[1] and EU[2], the lack of understandability of mainstream ML approaches poses a real threat to their applicability in real-world scenarios.

There are mainly two approaches that try to address this problem, namely eXplainable AI (XAI) and interpretable AI (IAI) [1].

The field of XAI, in particular, gained significant attention in recent years. It is important to note, however, that the currently available XAI techniques are not applicable to all tasks. As stated by [2], it is not safe to apply most of the modern XAI techniques to safety-critical or high-stakes environments. This is because the explanations provided by modern XAI are usually *approximations* of the other models and, as a consequence, they do not represent *exactly* those original models.

IAI techniques, instead, are based on the use of interpretable models, i.e., *models that can be understood and inspected by a human operator* [1] (throughout the paper, this will be our definition of "interpretability"). These techniques allow us to assess the security and safety of the produced models, but they can also serve to better *understand* a problem. In fact, by looking at an interpretable model, a human operator can *extract* knowledge about the problem at hand. However, IAI techniques are still less popular than their black-box counterparts, mainly because of their alleged lower performance. The idea that there is a trade-off between interpretability and performance is quite established, although such a trade-off has not been proved [2].

Recent works have addressed the problem of building interpretable models for reinforcement learning (RL) tasks. For instance, in [3] the authors implement a differentiable version of decision trees (DTs) and optimize them by using backpropagation. Dhebar et al. [4] propose non-linear DTs (NLDTs) for approximating and refining an oracle policy. However, while the results of these approaches seem very

[1] https://unesdoc.unesco.org/ark:/48223/pf0000380455/
[2] https://eur-lex.europa.eu/resource.html?uri=cellar:e0649735-a372-11eb-9585-01aa75ed71a1.0001.02/DOC_1&format=PDF

promising, the structure of the tree must be defined a priori. This issue requires us to either perform a trial-and-error search or include prior knowledge.

In this paper, we address this issue by proposing a two-level optimization scheme, based on a combination of evolutionary computation and RL, for training interpretable RL agents in the form of DTs. This two-level optimization algorithm allows us to decrease the amount of prior knowledge given to the algorithm (in the sense that there is no need to pre-define the agent structure a priori). Differently from the methods from the literature, our approach can learn automatically both a decomposition of the state space and the optimal state-action mappings. The main contributions of this paper can be summarized as follows:

- We propose a two-level optimization approach that optimizes both the topology of the tree and the decisions made for each state.
- We perform experimental tests on three classic RL problems from OpenAI Gym [5]: CartPole-v1, MountainCar-v0, and LunarLander-v2. On these tasks, we perform a comparison—both in terms of performance and a quantitative measure of interpretability—of the agents produced by our method against alternative interpretable and non-interpretable state-of-the-art models. Furthermore, we interpret the solutions produced by our approach, to understand how the agents work.
- We showcase a possible application of the proposed method in a high-stakes, safety-critical domain, namely pandemic control [6], demonstrating that our model produces policies that are more interpretable and, most importantly, more effective than state-of-the-art policies based on deep reinforcement learning and handmade control strategies. This is not an easy task: keeping a balance between the number of infections and economic losses can be indeed associated to the immunization problem [7] and the problem of optimizing social distancing [8], which are both known to be NP-hard. Moreover, this application clearly shows that, in high-stakes scenarios, interpretability is essential. In fact, in the case of pandemic control, a government cannot use a black-box model to take decision about how the pandemic should be controlled. Instead, there is the need for a clear, fairly small set of laws that can be read and understood by both the lawmaker and the people. In the long-term, demonstrating comparable or superior performance in this kind of tasks may lead to the adoption of AI even in high-stakes contexts, where there lack of interpretability currently represents a barrier.

The rest of the paper is structured as follows. In Section II we briefly summarize the most relevant related works, while Section III describes the method used in our approach. Section IV presents the experimental setup for the three control tasks from OpenAI Gym, as well as the metric of interpretability adopted in our analysis, and the mechanism used to simplify the produced DTs by pruning the unvisited nodes and reducing the action nodes. Then, in Section V we present the results of our experiments on the OpenAI Gym tasks. In Section V-B, we interpret the solutions obtained on these tasks, to understand how they work. In Section VI, we present the results we obtained on the pandemic control task, in comparison with those obtained by the existing methods from the literature. Finally, in Section VII we discuss our results and in Section VIII we draw the conclusions of this work.

II. RELATED WORKS

The use of DTs to solve RL tasks has been explored in several previous works. McCallum, in [9], proposes U-Trees: a kind of tree able to perform RL that handles the following sub-problems: choice of memories, selective perception, and value function approximation. In [10], the authors extend U-Trees to make them able to cope with continuous environments. They propose two novel tests that are used to create new conditions that split the state space. They test the proposed approach in two environments, a continuous one and an ordered-discrete one, and their results show that their approach is competitive with other approaches.

Pyeatt and Howe, in [11], propose a novel splitting criterion to build trees that can perform value function approximation. In their experiments, they compare the performance obtained by using their approach to the ones obtained using other splitting criteria, a table-lookup approach, and a neural network. The results show that the proposed approach usually achieves better performance than all the other approaches.

In [12], the authors propose a method that predicts the gain obtained by adding a split to the tree and selecting the best split to grow the tree. The experimental results show that this method is more effective than the method proposed in [11] on the tested environment.

Silva et al. [3] propose an approach to interpretable RL that uses Proximal Policy Optimization (PPO) on differentiable DTs. Moreover, they provide an analysis of the learning process while using either Q-learning or PPO. The experimental results show that this approach can produce competitive solutions in some of the tested tasks. However, it is also shown that when the differentiable DTs are discretized into canonical DTs, their performance may heavily decrease. A continuous version of this approach has been proposed in [13]

In [4], the authors use Evolutionary Algorithms (EAs) to evolve non-linear DTs. By non-linear, the authors mean that each split does not define a linear hyperplane in the feature space. The experimental results show that this approach can obtain competitive performance w.r.t. a neural-network-based approach. A case study for this approach is presented in [14].

In [15], the authors use the Grammatical Evolution (GE) algorithm [16] to evolve behavior trees (i.e., tree structures that allow more complex operations than a DT) for the Mario AI competition. The proposed agent can perform basic actions or pre-determined combinations of basic actions. Their solution achieved fourth place in the Mario AI competition.

However, the authors only evolve a controller, not exploiting the rewards given by the environment to increase the performance of the agent.

Hallawa et al., in [17], use behavior trees as evolved "instinctive" behaviors that are then combined with a learned behavior. While behavior trees are usually interpretable, the authors do not take explicitly into account the interpretability of the whole model, which comprises both a behavior tree and either a neural network or a $\mathcal{Q}$-learning table.

Several works applied evolutionary computation to evolve tree structures outside the RL domain. Krętowski, in [18], proposes a memetic algorithm based on genetic programming [19] and local search to optimize DTs. The results show that this approach can obtain performance that is comparable to the state-of-the-art while keeping the size of the tree significantly lower.

In [20], the authors propose a multi-objective EA to evolve regression trees and model trees. They find a Pareto front for RMSE, the number of nodes, and the number of attributes. The experimental results show that this approach can obtain performance that is comparable to or better than the state-of-the-art while using fewer nodes and attributes.

In [21], [22], the authors use the Genetic and Artificial Life Environment (GALE) to evolve DTs. Their results show that GALE can produce DTs that are competitive with the state-of-the-art.

III. METHOD

Decision trees are non-linear structures where each non-terminal node represents a "split" (i.e., a test on a condition), and each leaf node contains a decision. When creating DTs for RL tasks, one has to address two problems:

1) How do we choose the splits?
2) Given a leaf, which action do we need to assign to that leaf?

Clearly, there is a strong relationship between splits and decisions made in the leaves, so changing one of these without modifying the other may lead to significant changes in performance.

Several works [9]–[12] use greedy heuristics to induce DTs. However, these approaches have the following drawbacks:

- They use greedy choices to expand the trees. However, since inducing DTs is an NP-complete problem [23], this may cause the induction of sub-optimal trees [9], [24].
- They use tests to expand the trees. However, this causes these algorithms to suffer from the curse of dimensionality because, for each expansion of the tree, all the input variables need to be tested [9], [10].

Other works [15] (and [18], [20]–[22], even if they are not applied to RL tasks) induce trees using evolutionary approaches. However, these approaches only rely on an EA. In RL tasks, not exploiting the reinforcement signals obtained from the environment may slow down the evolution, thus resulting in a less efficient process. Our approach, instead, aims to combine evolutionary computation and RL methods to take the best of both worlds.

In particular, we propose a Baldwinian evolutionary approach to simultaneously optimize the structure of the tree and the state-action function. Baldwinian evolution is an evolutionary theory that states that what an individual learns during his life is not passed to their offspring, however, the knowledge acquired by the individual may introduce an evolutionary advantage that modifies the fitness landscape [25].

To evolve the structure of the DT, we use an EA, while we use $\mathcal{Q}$-learning to learn the state-action function. The evolutionary part optimizes, by searching for tree structures, a state space decomposition function. The $\mathcal{Q}$-learning part, instead, learns a mapping between each aggregated state (i.e., all the states that end in a given leaf of the tree) and the corresponding action. We refer to this approach as "two-level optimization approach", as there is an outer optimization loop where the structure of the DT is created, and an inner optimization loop where $\mathcal{Q}$-learning is performed, as shown in Figure 1.

The EA we use is the GE [16]. This EA evolves (context-free) grammars in the Backus-Naur Form.

Figure 1 shows a block diagram that clarifies the inner working of the proposed algorithm. The blue-colored parts are the processes inherent to the evolutionary component of our algorithm, while the red-colored ones are the processes concerning the RL part.

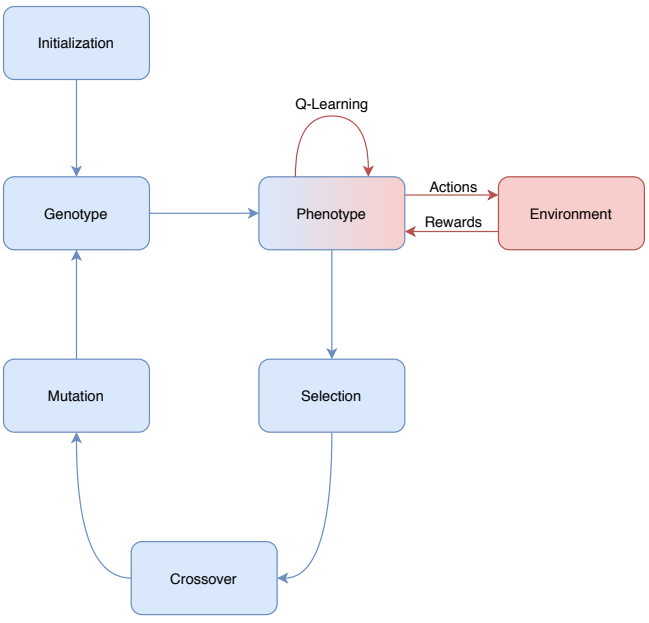


FIGURE 1: A scheme of the inner working of the proposed algorithm. The blue blocks are the ones that derive from the evolutionary part of our algorithm, while the red blocks are the ones that derive from the $\mathcal{Q}$-learning part.

A. EVOLUTIONARY ALGORITHM

To evolve DTs, we define a grammar, similarly to the approach described in [16]. In the following, we describe our algorithm design, highlighting the differences with the original GE introduced in [16]

1) Individual encoding

The genotype of an individual is encoded as a list of codons (represented as integers). However, differently from [16], our genotypes have fixed lengths.

2) Mutation operator

Instead of the mutation operator described in [16], we use uniform mutation. This operator mutates each gene according to a predefined probability. The new value of the gene is drawn uniformly from 0 and M (a number much higher than the maximum number of productions in the grammar, to ensure that all the productions are chosen according to a uniform probability distribution). Then, by using the modulo operator, we choose the production from the production rule.

3) Crossover operator

As for the crossover operator, we use standard one-point crossover. This operator sets a random cutting point and creates two individuals by mixing the two sub-lists of the parent genotypes. Differently from the original GE [16], we do not prune the individuals that have unexpressed codons (i.e., the parts of the genotype that are not used to create the phenotype, that is the DT). The reason underlying our choice not to prune the unexpressed codons lies in the fact that these may be a source of diversity that, in future generations, may lead to an evolutionary advantage for some individuals.

4) Replacement of the individuals

We use the same steady-state selection used in [16].

This mechanism allows us to preserve diversity between the individuals and, at the same time, guarantee elitism (i.e., make sure that good individuals are not lost during the evolutionary process).

5) Fitness evaluation

The fitness evaluation process consists of the following steps. First of all, the genotype is translated to the corresponding phenotype. I.e., given a genotype (in the form of a list of integers), we translate it into a phenotype (i.e., Python code). To do so, the first non-expanded symbol (the starting symbol is "dt") is expanded using the value of the current gene. To do so, for each production rule (i.e., a symbol enclosed in angular brackets) we retrieve the list (of size $|L|$) of possible productions (i.e, their outputs). Then, given the value of the current codon c, we replace the production rule with the $(c \ mod \ |L|)$-th string. Then, the next production rule is expanded using the following codon, and so on.

After translating the genotype into a phenotype, we have a DT without leaves. Then, we assign "empty" leaves to the DT, where each leaf contains a list of values, one for each action.

Then, the DT is used to simulate e training episodes where, at each step, the DT takes in input a state and returns one of the actions, in accordance with the corresponding leaf of the tree (i.e., with probability $1-\varepsilon$, it returns the argmax of the values of the leaf, and with probability ε it returns a random action). Then, using the feedback coming from the environment, the corresponding leaf is updated using the Bellman's equation for Q-learning [26]:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \underset{a'}{max}(Q(s',a')), \quad (1)$$

where $Q(s,a)$ is the a-th value in the list contained by the leaf, α is the learning rate, r is the reward, s and a refer to the current state and the current action, respectively; and s' and a' refer to the next state and the next action, respectively.

Finally, the fitness assigned to the agent is the mean of the returns obtained during the e episodes:

$$f(\pi) = \mathbb{E}_{i \in [1,e]}[R_i(\pi)] = \mathbb{E}_{i \in [1,e]}[\sum_{k=0}^{T} r(s_k^i, \pi(s_k^i))] \quad (2)$$

where f is the fitness of the agent that encodes the policy π, R_i is the return obtained in the i-th episode (i.e., the sum of the rewards obtained during the i-th episode), T is the number of steps performed for each episode, r is the reward function, s_k^i is the k-th state of the i-th episode, and $\pi(s_k^i)$ is the action taken by the policy given the state s_k^i.

Equation 2 is used by the Grammatical Evolution to evaluate the quality of each of the individuals in the current population.

IV. EXPERIMENTAL SETUP

We now present the details of our experimental setup, namely the interpretability metric used for our comparative analysis, the three OpenAI control tasks considered in the experimentation, and the mechanism devised to simplify the produced DTs. All the other details of our setup (i.e., grammar and parameters) can be found in the Supplementary Material. All the experiments have been conducted on a Linux workstation powered by an Intel®CoreTM i9-7940x CPU with 36 cores @3.10GHz and 64GB RAM, by using the DEAP Python library [27] for implementing the evolutionary algorithm[3].

For all the experiments, we perform 10 runs to assess the statistical repeatability of our results. Given the computational cost of the search, we found that 10 runs was an acceptable trade-off between the time required for the experiments and the statistical significance of the results. In fact, as shown in Table 2, the standard deviations are quite small, indicating that the process is statistically repeatable.

A. INTERPRETABILITY METRIC

In the next sections, we will show the results obtained by our method and compare them to the state-of-the-art using

[3]Our code is publicly available on a GitLab repository: https://gitlab.com/leocus/ge_q_dts

two metrics, namely the performance score given by the environment, and a metric of complexity used as a proxy for interpretability. For the latter, we adopt the interpretability metric proposed in [28], which is defined as follows:

$$\mathcal{M}_{orig} = 79.1 - 0.2l - 0.5n_o - 3.4n_{nao} - 4.5n_{naoc}$$

where:

- l is the size of the formula (i.e., the sum of constants, variables, and operations);
- n_o is the number of operations;
- n_{nao} is the number of non-arithmetical operations;
- n_{naoc} is the number of consecutive compositions of non-arithmetical operations.

However, this metric is meant to lie in $[0, 100]$ but, with "big" models (in terms of the number of operations), one can easily exceed these bounds, making the metric meaningless. For this reason, we modify the metric to make it work as a measure of *complexity*. For this purpose, the metric used is the following:

$$\mathcal{M} = -0.2 + 0.2l + 0.5n_o + 3.4n_{nao} + 4.5n_{naoc}$$

The changes we made yield the following properties:

- By changing the sign of all the terms, we obtain that a model with a higher complexity is harder to interpret.
- We replace the positive constant (79.1) with a negative one (-0.2) so that, when we have a constant formula, its complexity becomes 0 (this is the best case from the point of view of interpretability).

Furthermore, it is worth noting that this metric is in line with what stated in [29]. In fact, deep DTs can be as interpretable as black-box methods, because the terms l, n_o, n_{nao} and n_{naoc} have a high magnitude. Also, $\mathcal{M}$ seems to be (loosely) in line with what stated in [30]. In fact, by using a metric proportionate to the number of operations, we approximate the computational complexity of the model that we are executing.

Finally, we should note that while metrics for computing the complexity of specific machine learning models (e.g., DTs) do exist, the $\mathcal{M}$ metric gives us the advantage to be model-agnostic, i.e., it is in principle applicable to every machine learning model that can be expressed as a set of equations. This property is extremely important, as it allows us to compare the interpretability of fundamentally different ML models, such as DTs and neural networks.

B. OPENAI CONTROL TASKS

To test our approach, in the first part of our experimentation we consider the following OpenAI Gym [5] environments: CartPole-v1, MountainCar-v0, and LunarLander-v2.

To assess the statistical repeatability of our experiments, we perform 10 independent runs for each environment and experimental setting. As required by [5], we test (a posteriori, i.e. after the evolutionary process) the best model obtained in each run over 100 independent episodes, to assess its performance. By "testing", we mean in this case the execution of the best policy in 100 testing episodes (i.e., episodes whose starting points are different from those used for training).

In the following, we describe the three environments and their properties, i.e., the observation and action spaces, the reward function, the termination, and the resolution criteria.

1) CartPole-v1

In this task, the agent must balance a pole (placed on top of a cart) by moving the cart.

a: Observation space

The state of the environment is composed of the following features:

- Cart position: $x \in [-4.8, 4.8]\ m$
- Cart velocity: $v \in (-\infty, \infty)\ m/s$
- Pole angle: $\theta \in [-0.418, 0.418]\ rad$
- Pole angular velocity: $\omega \in (-\infty, \infty)\ rad/s$

b: Action space

The actions that the agent can perform are:

- Push the cart to the left by applying a force of 10N: $move\_left$
- Push the cart to the right by applying a force of 10N: $move\_right$

c: Reward

The agent receives a reward of +1 point for each timestep.

d: Termination criterion

The task episode terminates if:

- The cart position lies outside the bounds for the x variable.
- The angle of the pole lies outside the bounds for the θ variable.

e: Goal of the task

This task is solved if the agent receives a mean total reward of $R \geq 475$ points on 100 episodes.

2) MountainCar-v0

In this environment, the agent has to drive a car, starting from a random position in a valley (surrounded by two hills, one on the left and one on the right of the valley), with the goal of reaching a goal position on the right hill. However, the car's engine is not powerful enough, so the agent must learn how to build momentum by exploiting the two hills until the car reaches the goal position.

a: Observation space

The state of the environment consists of the following variables:

- Horizontal position of the car: $x \in [-1.2, 0.6]\ m$
- Horizontal velocity of the car: $v \in [-0.07, 0.07]\ m/s$

b: Action space

The agent can perform 3 actions:

1) Accelerate to the left by applying a force of 0.001N: acc_right
2) Do not accelerate: nop
3) Accelerate to the right by applying a force of 0.001N: acc_left

c: Reward

The agent receives a reward of -1 point for each timestep.

d: Termination criterion

The task episode terminates after 200 timesteps.

e: Goal of the task

This task is solved if the agent receives a mean total reward of $R \geq -110$ points on 100 episodes.

3) LunarLander-v2

In this task, the agent has to land a lander on a landing pad.

a: Observation space

The state of the environment consists of 8 variables:

- Horizontal position: $p_x \in [-1, 1]\ m$
- Vertical position: $p_y \in [-0.5, 2]\ m$
- Horizontal velocity: $v_x \in [-2, 2]\ m/s$
- Vertical velocity: $v_y \in [-2, 0.5]\ m/s$
- Angle w.r.t. the vertical axis: $\theta \in [-2, 2]\ rad$
- Angular velocity: $\omega \in [-2.5, 2.5]\ rad/s$
- Left leg contact: $c_l \in \{0, 1\}$
- Right leg contact: $c_r \in \{0, 1\}$

b: Action space

The agent can perform 4 actions:

1) All engines disabled: nop
2) Enable left engine: $left$
3) Enable main engine: $main$
4) Enable right engine: $right$

c: Reward

The rewards (defined by the OpenAI Gym environment) for moving from the initial point to the landing pad with a final velocity of zero varies between 100 and 140 points. If the lander crashes, it receives a reward of -100 points. If the lander lands correctly, it receives a reward of +100 points. For each leg contact, the agent receives a reward of +10 points. Firing the main engine gives a reward of -0.3 points; firing a side engine rewards the agent with -0.03 points.

d: Termination criterion

The task episode ends if at least one of these conditions is met: the duration exceeds 1000 timesteps, the lander crashes, or it passes the bounds of the environment.

e: Goal of the task

This task is solved if a mean total reward $R \geq 200$ points are obtained on 100 episodes.

C. SIMPLIFICATION MECHANISM

To make our solutions even more interpretable, we use a simplification mechanism for the best solution found across the various runs. The simplification mechanism is the following. First of all, we execute the given policy for 100 validation episodes (i.e., different from both training and testing episodes). Here, we keep a counter for each node of the tree that is increased each time the node is visited. Once this phase finishes, we remove all the nodes that have not been visited. Finally, we iteratively search for nodes in the tree whose leaves correspond to the same action. Each time we find such a node, we replace it with a leaf that contains such action. The iteration stops when the tree does not contain nodes of this type.

V. RESULTS

A. OPENAI CONTROL TASKS: NUMERICAL RESULTS

In this section we present the numerical results obtained on the three considered OpenAI control tasks, and compare them with the state-of-the-art. Please note that in the Supplementary Material we provide further details on this comparison, and perform an ablation study that confirms that our two-level approach boosts the performance w.r.t. a one-level method.

1) CartPole-v1

In this environment, we tested two different grammars: one to evolve *orthogonal* DTs, and one to evolve *oblique* DTs. Orthogonal DTs are DTs in which each condition tests a single variable. Thus, each split defines a hyperplane that is orthogonal to the axis of the tested variable. In contrast, oblique DTs handle multiple variables for each condition, resulting in oblique hyperplanes.

We initialize the leaves randomly in $[-1, 1]$. The number of episodes used for $\mathcal{Q}$-learning is quite low (10 episodes). This choice was made because, since this is a "simple" environment (i.e., the agent does not need to see a great number of episodes in order to solve it), we want to lower the computational cost of the search by *exploiting* the randomness used to initialize the state-action function. So, in this case, $\mathcal{Q}$-learning is used to "fine-tune" the state-action function instead of learning it from scratch. In fact, the whole evolutionary process has a computational complexity of $O(pge)$, where p is the size of the population, g is the number of generations, and e is the number of episodes for each individual. Thus, reducing the number of episodes can significantly reduce the computational cost of the evolutionary process, although this increases the risk that the agents will not learn the optimal state-action function.

The results are shown in Table 1 (second and third rows). The detailed results for each run are shown in the Supplementary Material. Here we observe that, while the orthogonal

grammar can solve the task in 100% of the cases (a case is defined simply as a set of episodes), the testing score is optimal (500 ± 0) only in 40% of the testing episodes. The oblique grammar, instead, can solve the task in 90% of the cases but achieves the optimal score in 80% of the runs. This result suggests that, while the oblique grammar makes the search space more complex, it usually leads to more stable (as in Lyapunov's concept of stability) solutions.

TABLE 1: Descriptive statistics (mean and std. dev. over the best solutions found in 10 independent runs) on the three OpenAI environments considered in the experimentation. The results obtained with the orthogonal grammar are not shown for the LunarLander-v2 environment since we were not able to find a set of hyperparameters that led to solutions for this task.

Environment	Type	Mean training score	Std. dev. training score	Mean testing score	Std. dev. testing score	No. runs solved
CartPole-v1	Orthogonal	499.75	0.55	497.34	5.19	10
	Oblique	500.00	0.00	495.66	12.27	9
MountainCar-v0	Orthogonal	-110.62	4.57	-108.16	6.20	7
	Oblique	-107.90	3.09	-110.31	4.39	5
LunarLander-v2	Oblique	255.58	14.58	213.09	18.73	10

TABLE 2: Testing scores (mean and std. dev. over 100 episodes) for the solutions obtained in each run tested on a 10^4 steps long version of CartPole-v1.

Tree type	Run	Testing mean	Testing std. dev.
Orthogonal	R1	878.08	346.85
	R2	767.94	202.24
	R3	3271.99	2718.79
	R4	5845.54	2898.37
	R5	1237.18	775.24
	R6	2589.85	2715.03
	R7	4561.71	3670.16
	R8	5738.87	3227.96
	R9	1179.21	543.78
	R10	688.75	183.64
Oblique	R1	10000.00	0.00
	R2	9900.68	988.22
	R3	10000.00	0.00
	R4	10000.00	0.00
	R5	10000.00	0.00
	R6	10000.00	0.00
	R7	10000.00	0.00
	R8	10000.00	0.00
	R9	10000.00	0.00
	R10	9200.95	2709.71

To better assess this hypothesis, i.e., that oblique trees are more stable than orthogonal ones, in Table 2 we compare all the trees produced by using the two grammars on a modified environment that has a maximum duration of 10^4 timesteps instead of 500. These results confirm our hypothesis, showing that all the oblique trees are able to obtain significantly better scores, often reaching a perfect score (i.e., $10^4 \pm 0$) also in this setting. Figure 2c shows how the testing mean score varies by varying the number of maximum timesteps for the best agents.

In Figures 2a and 2b, we show instead the mean distance from the point of equilibrium ($p_{eq} = [0, 0, 0, 0]^T$) averaged over 100 episodes (of length 500 timesteps). In these figures, we can easily observe that the oblique policy shows stable behavior (according to Lyapunov's concept of stability), while the orthogonal policy does not.

Moreover, we tested the robustness of the produced agents w.r.t. noise on the inputs received by the sensors. In Figure 2d, we show how the performance of the two best agents varies with additive input noise (distributed as $\mathcal{N}(0, \sigma^2)$). The orthogonal tree is robust only up to noise with σ as big as twice the sampling step used for the constants. In contrast, the oblique tree proves to be significantly more robust, being able to cope with noises that have a σ about 50 times bigger than the sampling step used for the constants.

Finally, in Table 3 we compare our best solutions with the best solutions found in the literature. The complexities computed for the neural-network-based approaches are approximations, i.e., we did not take into account all the details of the methods but only those related to the network architectures (which however account for the major part of the complexity). For our comparison purposes, this omission is acceptable. Our best solutions (in terms of the best score, used for the comparison) are shown in Figures 3a and 3b. The other solutions obtained (in terms of best $\mathcal{M}$) can be found in our public repository.

Please note that, for all the tables comparing our approach to the state-of-the-art (i.e. Table 3 and, similarly for the other two OpenAI tasks, Table 4 and Table 5), we compare our solutions that achieve either the best score or the best interpretability. In these tables, "Best score" refers to the solution(s) that achieve the maximum score. Thus, the $\mathcal{M}$ value in the corresponding row indicates the mean $\mathcal{M}$ value of all our solution(s) that achieve the maximum score. Similarly, in the row indicated by "Best $\mathcal{M}$", we show the statistics of the solution(s) that achieve minimum $\mathcal{M}$, and their mean score.

2) MountainCar-v0

As for the previous environment, we test both the orthogonal and the oblique grammar. Note that, in this environment, we normalize the variables in the oblique case whereas in the other environments we do not. Normalization is necessary because the ranges of variation of the two variables are quite different. Moreover, a preliminary experimental phase confirmed that it was hard to obtain good results by not normalizing the inputs.

Since this can be considered a "simple" task (i.e., the agent does not need to see a great number of episodes in order to solve it), also in this case we train the agent only for 10 episodes, exploiting the randomness of the initialization.

The results obtained by the best solution for each run are shown in Table 1 (fourth and fifth rows). As we can see from the table, the solutions obtained by using the orthogonal grammar solve the task in 70% of the cases. Conversely, we observe that oblique trees perform poorly on this problem. This result suggests that this problem is more difficult to solve by using oblique trees than orthogonal ones. While this may seem counter-intuitive, since oblique trees are a generalization of orthogonal trees, it may be because our

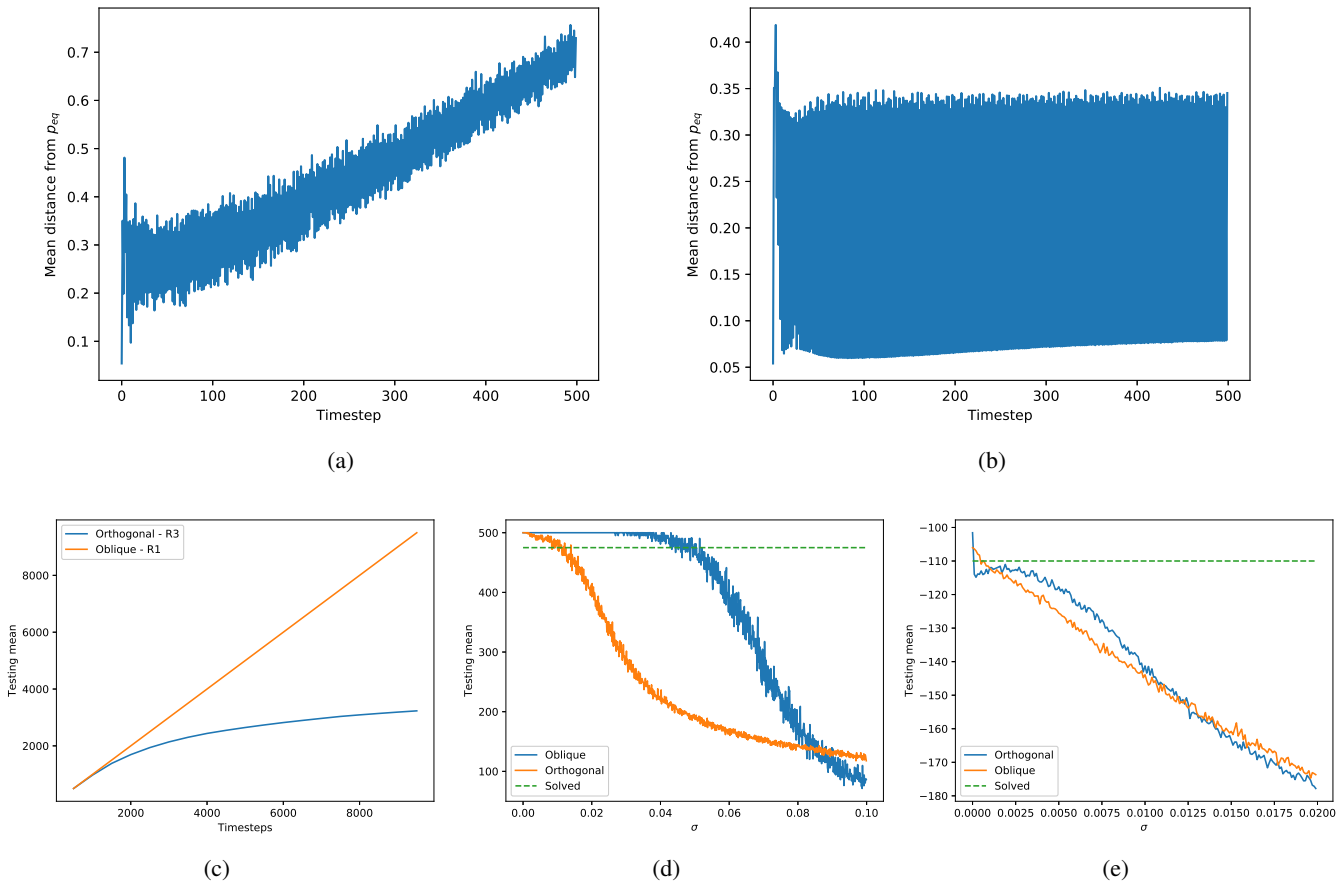


FIGURE 2: (a) Mean distance of the CartPole-v1 system from the point of equilibrium when using the best orthogonal tree as policy; (b) Mean distance of the CartPole-v1 system from the point of equilibrium when using the best oblique tree as policy; (c) Comparison between the best orthogonal tree with the best oblique at different maximum timesteps for the CartPole-v1 environment; (d) Performance of the two best agents on CartPole-v1 as the input noise changes; (e) Mean testing score with different input noises for the best orthogonal and oblique trees on MountainCar-v0.

grammar (the one used to produce oblique trees) hinders obtaining an orthogonal DT.

To compare the two kinds of trees, we analyze the robustness to input noise for both versions. The result is shown in Figure 2e. In this case, both approaches proved to be not robust to noise. Surprisingly, we can observe that the orthogonal tree is not even robust to input noise with $\sigma < \min_i(step_i)$, where $step_i$ is the sampling step for the constants of the i-th variable.

Finally, we perform a comparison of our solutions w.r.t. the state-of-the-art. In Table 4, we show the results of the comparison. The best trees (in terms of mean testing score) used for the comparison are shown in Figures 3d and 3c. We can see that our models achieve the best results both in terms of score as well as $\mathcal{M}$ metric.

3) LunarLander-v2

For this environment, we did not find a configuration that gave satisfying results with orthogonal trees. For this reason, we show only the results obtained by using the oblique grammar.

In this case, we significantly increased the number of episodes used for training (i.e., we set it to 1000). The rationale for this choice is the following:

- The LunarLander-v2 environment is not as easy to solve as the previous environments.
- In this case, we do not use a random initialization of the leaves, but we leverage only $\mathcal{Q}$-learning to learn the state-action function and remove the bias due to the randomness.

[4]https://github.com/ZhiqingXiao/OpenAIGymSolution
[5]https://github.com/StepNeverStop/RLs
[6]https://github.com/harshitandro/Deep-Q-Network
[7]https://github.com/CM-Data/Noisy-Dueling-Double-DQN-MountainCar
[8]https://github.com/amitkvikram/rl-agent

TABLE 3: Comparison (testing score and $\mathcal{M}$ metric of the best agents trained on 10 independent runs) of the solutions obtained by using the proposed approach w.r.t. the state-of-the-art in the CartPole-v1 environment. The boldface indicates the best value of the score and $\mathcal{M}$ metric. Note that the results from [31] are averaged over 10 runs, while the results from [3] regard the discretized tree shown in Figure 3 (right) reported in [3], tested on the same episodes used for the evaluation of our solutions.
(*): Result confirmed by personal communication with the first author of the study.
(**): The tree has been simplified by using the technique used in our work.

Source	Method	Score	$\mathcal{M}$
Meng et al. [31]	Deep Q Network	327.30	1157.20
Meng et al. [31]	Tree-Backup(λ)	494.70	1157.20
Meng et al. [31]	Importance-Sampling	498.70	1157.20
Meng et al. [31]	Qπ	489.90	1157.20
Meng et al. [31]	Retrace(λ)	461.10	1157.20
Meng et al. [31]	Policy discrepancy	499.90	1157.20
Meng et al. [31]	Policy discrepancy	493.20	1157.20
Meng et al. [31]	Watkins's Q(λ)	484.30	1157.20
Meng et al. [31]	Policy discrepancy	494.90	1157.20
Meng et al. [31]	Policy discrepancy	493.30	1157.20
Meng et al. [31]	P&W Q(λ)	496.70	1157.20
Meng et al. [31]	Policy discrepancy	**500.00**	1157.20
Meng et al. [31]	Policy discrepancy	499.40	1157.20
Meng et al. [31]	General Q(λ)	499.90	1157.20
Meng et al. [31]	Policy discrepancy	**500.00**	1157.20
Meng et al. [31]	Policy discrepancy	**500.00**	1157.20
Xuan et al. [32]	Deep Q Network	98.33	5170174.80
Xuan et al. [32]	Bayesian Deep RL	113.52	8090.40
Xuan et al. [32]	Bayesian Deep RL	136.75	8090.40
Beltiukov [33]	K-FAC	321.00	70786.20
Silva et al. [3]	Differentiable DTs	388.76	89.20
Silva et al. [3]	Differentiable DTs (*)	**500.00**	106.80
Silva et al. [3]	Differentiable DTs (**)	**500.00**	53.40
Ours – Best score	Orthogonal DT	**500.00**	35.60
Ours – Best score	Oblique DT	**500.00**	**24.10**
Ours – Best $\mathcal{M}$	Orthogonal DT	**500.00**	35.60
Ours – Best $\mathcal{M}$	Oblique DT	**500.00**	**24.10**

TABLE 4: Comparison (testing score and $\mathcal{M}$ metric of the best agents trained on 10 independent runs) of the solutions obtained by using the proposed approach to the state-of-the-art in the MountainCar-v0 environment. The boldface indicates the best value of the score and $\mathcal{M}$ metric.

Source	Method	Score	$\mathcal{M}$
Zhiqing Xiao[4]	Closed-form policy	-102.61	54.70
Keavnn[5]	Soft Q Networks [34]	-104.58	31079.20
Harshit Singh[6]	Deep Q Network	-108.85	984160.30
Colin M[7]	Double Deep Q Network	-107.83	46681.60
Amit[8]	Tabular SARSA	-105.99	381.50
Dhebar et al. [4]	NLDT (Open-loop)	-128.87	66.80
Ours – Best score	Orthogonal DT	**-101.72**	106.80
Ours – Best score	Oblique DT	-106.02	46.80
Ours – Best $\mathcal{M}$	Orthogonal DT	-116.68	35.60
Ours – Best $\mathcal{M}$	Oblique DT	-106.50	**23.40**

Moreover, we use a slightly different $\mathcal{Q}$-learning configuration for this environment. In fact, in this case, we use a decay for ε, to better explore the search space. The decay works as follows: in the k-th visit to the leaf, an $\varepsilon = \varepsilon_0 \cdot decay^k$ is used. We set the learning rate to $\frac{1}{k}$, where k is the number of visits of the state-action pair. This choice guarantees that the state-action function converges to the optimum with $k \to \infty$. Finally, to save computation time, we implement an early stopping criterion that works as follows: if the mean score over the current period (i.e., a predefined number of episodes) is smaller than the one obtained in the previous period, then training is stopped. This mechanism relies on the following assumption: if the current score worsens, but the state-action function is converging, the worsening is due to small oscillations caused by randomness.

The results obtained in this environment are summarized in Table 1 (last row). In this case, our approach solves the task in 100% of the testing episodes.

A comparison of our two best solutions (w.r.t. score and interpretability) with the state-of-the-art is shown in Table 5. As we can observe, even though we do not achieve (in absolute terms) the best score and the best $\mathcal{M}$, our solutions represent the best compromise between the two metrics. Moreover, we can observe that a Pareto front that explains the trade-off between interpretability and performance seems to exist in this task. However, our best solution achieves a performance comparable to the best score reported in the state-of-the-art, while having a substantially smaller complexity. Our best solution is shown in Figure 3e.

TABLE 5: Comparison (testing score and $\mathcal{M}$ metric of the best agents trained on 10 independent runs) of the solutions obtained by using the proposed approach and the state-of-the-art in the LunarLander-v2 environment. The boldface indicates the best value of the score and $\mathcal{M}$ metric. Note that the results from [35] are averaged over 5 runs.

Source	Method	Score	$\mathcal{M}$
Keavnn[9]	Soft Actor Critic	217.92	210733.2
liu[10]	Soft Q Network	217.09	647691.1
Ash Bellet[11]	Deep Q Network	225.79	1295307.1
Sanket Thakur[12]	Deep Q Network	200.65	259285.8
Mahmood[13]	Deep Q Network	200.30	237079.7
Daniel Barbosa[14]	Proximal Policy Optimization	201.47	1673.0
XinlyYu[15]	Deep Q Network	**278.23**	518153.0
Ruslan[16]	Dueling Deep Q Network	200.22	30878.1
Ollie Graham[17]	Deep Q Network	201.46	30878.1
Nikhil Barhate[18]	Actor Critic	254.58	4337.3
Udacity[19]	Deep Q Network	201.46	30878.1
Sigve Rokenes[20]	Deep Q Network	266.00	$1.21 \cdot 10^8$
Peng et al. [35]	Advantage-weighting	229.00 ± 2.00	518153.0
Xu et al. [36]	Value-difference	248.20 ± 21.00	632620.2
Malagon et al. [37]	Shallow neural network	258.80	**77.6**
Silva et al. [3]	Rule list	-78.40	89.0
Dhebar et al. [4]	NLDT* – Depth 3	132.83	136.7
Ours – Best score	Oblique DT	272.14	118.9
Ours – Best $\mathcal{M}$	Oblique DT	262.18	86.9

B. OPENAI CONTROL TASKS: INTERPRETATION OF THE OBTAINED POLICIES

In the following, we interpret the best solutions found on the three OpenAI control tasks, to understand how they work and highlight the potentialities of our proposed approach in terms of interpretability.

1) CartPole-v1

a: Orthogonal tree

The tree shown in Figure 3a is extremely easy to interpret. This agent moves the cart to the left if:

$$\omega < 0.074 \wedge \theta < 0.022. \tag{3}$$

Otherwise, it moves the cart to the right. Note that there is a case in which the pole is falling to the right but the agent moves the cart to the left: $\theta \in [0, 0.022) rad \wedge \omega \in [0, 0.074) rad/s$. This is not a problem, because when the agent moves the cart to the right, it increases the velocity of the pole, resulting in a "move_right" action in the subsequent steps.

b: Oblique tree

In this case (Figure 3b), the interpretation of the policy is a bit harder. The condition used by the agent to discriminate between the two states is:

$$-0.274x_k - 0.543v_k - 0.904\theta_k - 0.559\omega_k < -0.169 \tag{4}$$

where k refers to the current timestep. To simplify the process, we write Eq. (3) as the following:

$$-ax_k - bv_k - c\theta_k - d\omega_k < t. \tag{5}$$

First of all, we want to analyze the role of the constant t in the policy. By testing it with different values (i.e., $t = -0.169$, $t = 0.169$, $t = -0.1$, $t = 0.1$, $t = 0$) we observed that it holds that the final point in which the pole is balanced can be obtained as follows:

$$x_n \approx -\frac{t}{a} \tag{6}$$

where n is the index of the last timestep. For simplicity, let us assume that $x_n = -\frac{t}{a}$. This means that we can rewrite Eq. (5) as follows:

$$-x_k - \frac{b}{a}v_k - \frac{c}{a}\theta_k - \frac{d}{a}\omega_k < \frac{t}{a} = -x_n. \tag{7}$$

We can then perform other algebraic steps and obtain:

$$-x_k - b'v_k - c'\theta_k - d'\omega_k < -x_n \Rightarrow \tag{8}$$

$$-b'v_k - c'\theta_k - d'\omega_k < -x_n + x_k \Rightarrow \tag{9}$$

$$-b'v_k - c'\theta_k - d'\omega_k < -x_n + x_{n-1} - \cdots + x_k = \\ = \sum_{j=n}^{k+1} -x_j + x_{j-1}. \tag{10}$$

Then, by noting that:

$$\frac{x_k - x_{k-1}}{\tau} = v_k \tag{11}$$

we can rewrite Eq. (10) as:

$$-b'v_k - c'\theta_k - d'\omega_k < -\sum_{j=k+1}^{n} v_j\tau \tag{12}$$

$$-c'\theta_k - d'\omega_k < -\sum_{j=k}^{n} g_j v_j \tau \tag{13}$$

where:

$$g_j = \begin{cases} \frac{-b'}{\tau} & \text{if } j = k \\ 1 & \text{otherwise} \end{cases}$$

Now, by observing that:

$$\frac{\theta_k - \theta_{k-1}}{\tau} = \omega_k \tag{14}$$

we obtain:

$$-c'\theta_k - d'\frac{\theta_k - \theta_{k-1}}{\tau} < -\sum_{j=k+1}^{n} g_j v_j \tau \tag{15}$$

$$-(d' + \tau c')\theta_k + d'\theta_{k-1} < -\tau^2 \sum_{j=k+1}^{n} g_j v_j. \tag{16}$$

Finally, noting that, usually, in the first 50 timesteps of the episodes the velocities are high ($\max_k |v_k| < 1.5$) and then they become small ($\max_k |v_k| < 0.55$) because the pole is balanced, we can write that:

$$\left| \sum_{j=k+1}^{n} g_j v_j \right| \lessapprox \frac{b'}{\tau} \cdot 1.5 + 49 \cdot 1.5 + 450 \cdot 0.55 = 420 \tag{17}$$

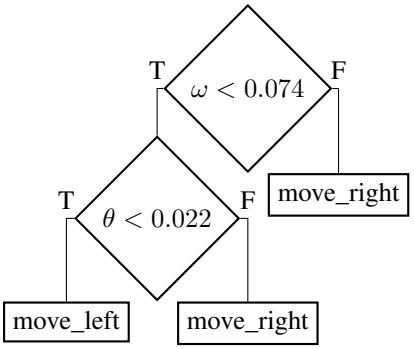
where the approximate equality holds in the worst case (i.e., $k = 0$ and all the velocities have the same sign). However, considering that in our observations the magnitude of the velocities was usually significantly smaller than the maximum and that the summation is multiplied by τ^2 ($\tau = 0.02$ in this environment), we can safely consider only the term with the highest magnitude, i.e., $\frac{b'}{\tau}v_k$. Moreover, using only v_k sets $x_n \approx 0$, which makes the system easier to understand.

Then, we obtain:

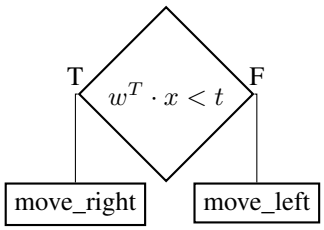
$$-(d' + \tau c')\theta_k + d'\theta_{k-1} < \tau b'v_k \tag{18}$$

$$c\theta_k > -(bv_k + d\omega_k). \tag{19}$$

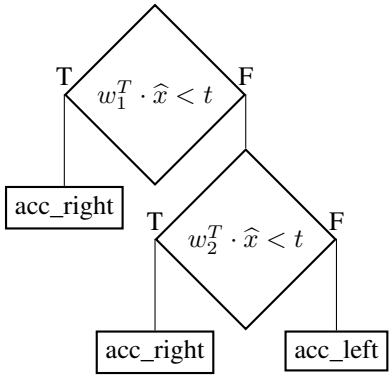
[9] https://github.com/StepNeverStop/RLs
[10] https://github.com/createamind/DRL
[11] https://github.com/nextgrid/deep-learning-labs-openAI
[12] https://github.com/sanketsans/openAIenv
[13] https://github.com/cpow-89/Extended-Deep-Q-Learning-For-Open-AI-Gym-Environments
[14] https://github.com/danielnbarbosa/angela
[15] https://github.com/XinliYu/Reinforcement_Learning-Projects
[16] https://github.com/RMiftakhov/LunarLander-v2-drlnd
[17] https://github.com/Cozmo25/openai-lunar-lander-v2
[18] https://github.com/nikhilbarhate99/Actor-Critic-PyTorch
[19] https://github.com/udacity/deep-reinforcement-learning
[20] https://evgiz.net/article/2019/02/02/



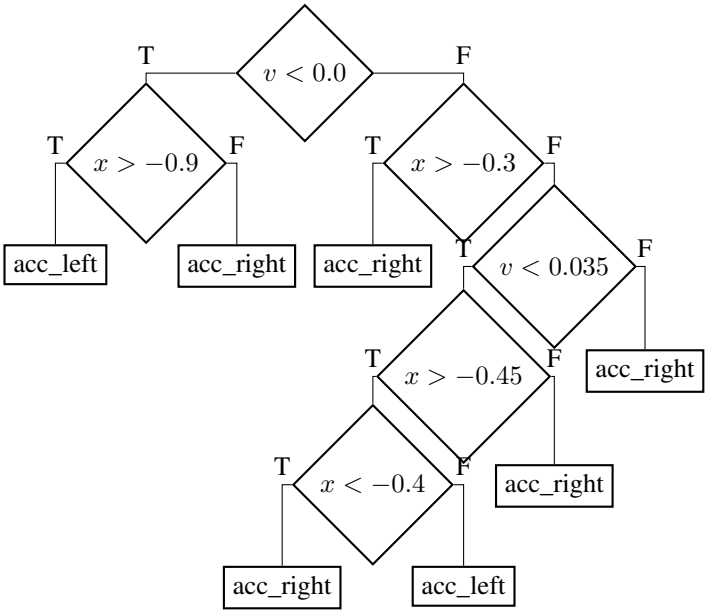
(a) Best orthogonal DT (w.r.t. score) evolved in the CartPole-v1 environment.



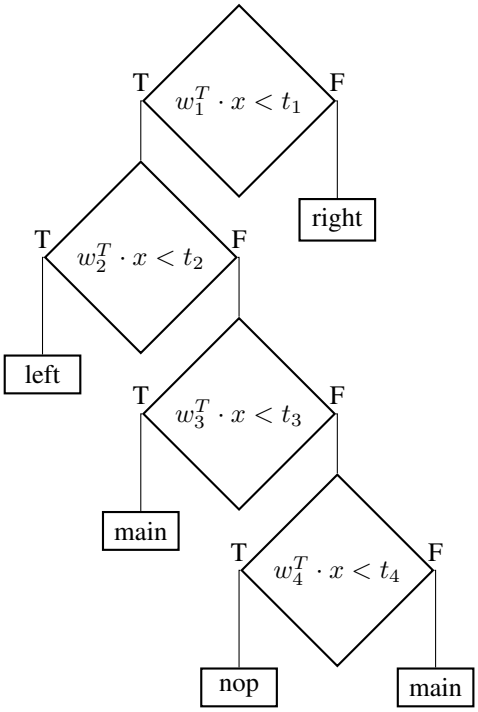
(b) Best oblique DT (w.r.t. score) evolved in the CartPole-v1 environment.



(c) Best oblique DT (w.r.t. score) evolved in the MountainCar-v0 environment.



(d) Best orthogonal DT (w.r.t. score) evolved in the MountainCar-v0 environment.



(e) Best oblique DT (w.r.t. score) evolved in the LunarLander-v2 environment.

FIGURE 3: Best tree (w.r.t. score) for each tested scenario. The weights for the oblique trees are shown in the Supplementary Material.

Approximating the constants, we set $b = 0.543 \approx 0.5$, $c = 0.904 \approx 1$, $d = 0.559 \approx 0.5$, so the final policy is[21]:

$$\pi(x, v, \theta, \omega) = \begin{cases} move\_right & \text{if } \theta_k > -\frac{1}{2}(v_k + \omega_k) \\ move\_left & \text{otherwise.} \end{cases}$$

A dimensionally consistent policy is $\theta_k + \frac{1}{2}(v_k/l + \omega)\frac{n_{ts}\tau}{\tau} > 0$, where $l = 1$ is the pole length and n_{ts} is the number of steps that we are taking into consideration to balance the pole (in our case, $n_{ts} = 1$). This policy can be interpreted as follows. If the sum of the current angle and the mean angle given by the two contributions (i.e., the linear velocity of the cart and the angular velocity of the pole) are positive (it is a kind of "prediction" of the future angle), then move the cart to the right, because it is going to fall to the right. Otherwise, move the cart to the left.

[21]Implementing this policy by using the ω value given by the environment may give slightly lower than perfect scores: in our opinion, this is due to the error carried by the integration method used. Alternatively, using $\omega_k = (\theta_k - \theta_{k-1})/\tau$ gives the desired results.

2) MountainCar-v0

a: Orthogonal tree

Also in this case, the orthogonal tree (Figure 3d) is easy to interpret. If we look at the leaves, we see that the agent accelerates to the left only in two cases: $(v < 0 \wedge x > -0.9) \vee (v \in [0, 0.035) \wedge x \in [-0.4, -0.3])$. This means

that the agent accelerates to the left when:

- it is going towards the left hill to build momentum, and it is far from the border ($x > -0.9$), so it tries to maximize the potential energy of the car;
- velocity is positive but not enough to reach the target hill ($v < 0.035$) and it is near the valley.

In all the other cases, the agent accelerates to the right.

b: Oblique tree

In this case (Figure 3c), the agent accelerates to the left when both conditions are False. This means that we have to solve the following system of two inequalities:

$$\begin{cases} 0.717\widehat{x} - 0.697\widehat{v} \geq -0.229 \\ 0.138\widehat{x} - 0.883\widehat{v} \geq -0.389 \end{cases}$$

In practice, the agent accelerates to the left when $v \leq 7.5799 \cdot 10^{-2} \cdot x + 6.6955 \wedge v \leq 1.1516 \cdot 10^{-2} \cdot x + 5.495 \cdot 10^{-3}$. Note that, to facilitate interpretability, here the variables have been denormalized. The decision hyperplane can be easily obtained by combining the two inequalities.

It is important to note that the lack of robustness for this solution does not allow us to further approximate the constants.

3) LunarLander-v2

In this case, since the oblique tree (Figure 3e) has 4 conditions and 8 unknowns, it is a bit harder to interpret.

a: First condition

This condition, when it evaluates to False, turns on the right engine for a timestep. So, we turn on the right engine when:

$$ap_x - bp_y + cv_x - dv_y - e\theta - f\omega - gc_l - hc_r \geq 0 \quad (20)$$

where $a, b, \ldots, h$ replace the constants shown in Figure 3e.

To simplify the analysis, let us assume $c_l = c_r = 0$, since they can assume only two values: $0, 1$. This simplification does not affect the generality of our analysis, since we are only assuming that there is no contact with the ground. We can simply say that, when contact with the ground happens, then the threshold is not 0 anymore, but it can take one of the following values: 0.2 (only right leg touches the ground), 0.597 (only right leg touches the ground), 0.797 (both legs touch the ground).

So, we can rewrite condition 20 as follows:

$$ap_x + cv_x - bp_y - dv_y - e\theta - f\omega \geq 0. \quad (21)$$

By merging some terms we obtain:

$$a(p_x + v_x c') - b(p_y - v_y d') - e(\theta - \omega f') \geq 0. \quad (22)$$

We analyzed the terms in parenthesis and we discovered that they approximate the position (or the angle) in the following timestep. The constants $c' \approx f' \approx 1.23$ lead to an overestimation of the magnitude of the future position (or angle), while the constant $d' \approx 0.53$ increases the precision of the approximation. By denoting the predictions of the next position on x, y and θ with p_x^{k+1}, p_y^{k+1}, θ^{k+1}, respectively, we can write:

$$ap_x^{k+1} - bp_y^{k+1} \geq e\theta^{k+1}. \quad (23)$$

To understand how this condition works, let us suppose that $p_x^{k+1} \approx 0$ (i.e., the lander is in the center of the environment). Then, if $p_y^{k+1} \approx 1$ (i.e., near the starting point), the agent fires the right engine if $\theta^{k+1} \leq -b/e \approx -0.15 rad$, i.e., the angle of the lander is going to fall to the right. When $p_y^{k+1} \approx 0$ (i.e., near the landing pad), the agent fires the right engine if $\theta^{k+1} \leq 0$. So, we can say that the farther the lander is from the landing pad (vertically), the more margin we have on the threshold of the angle. Let us now suppose that $p_y^{k+1} = 0$, to study the effect of p_x^{k+1} on the policy. In this case, we can say that the agent turns on the right engine when $\theta \leq \frac{a}{e} p_x^{k+1}$. So, when the agent is in the right part of the environment, it uses a linear threshold to activate the engine to avoid both high angles and high displacements from the landing pad location. Similarly, when p_x^{k+1} is negative, the threshold is negative so the agent tries both to compensate for negative angles (that would move it farther on the left) and distance from the landing point.

b: Second condition

The second condition, when evaluates to True, leads to the firing of the left engine. Also in this case, let us neglect the terms c_l and c_r. We can write the condition as:

$$ap_x - bp_y + cv_x - dv_y - e\theta - f\omega < 0. \quad (24)$$

It should be noted that the coefficients $a, \ldots, f$ are different from the previous ones. By grouping the terms as before, we obtain:

$$a(p_x + v_x c') - b(p_y + v_y d') - e(\theta + \omega f') < 0. \quad (25)$$

Also in this case, the constants seem to have the same role (i.e., some lead to an overestimation of the next position while some to a better estimate), so we can write:

$$ap_x^{k+1} - bp_y^{k+1} < e\theta^{k+1}. \quad (26)$$

This condition is easy to understand given the previous one: in fact, it is the opposite condition. This means that we can use the same reasoning used above to understand it.

c: Third condition

This condition handles the firing of the main engine. For this reason, we expect it to work differently from the previous two. We can easily observe that the signs of the terms in x and y are inverted. Moreover, the two angular terms do not have the same sign. Also in this case, let us use $a, \ldots, f$ to rename the constants and ignore c_l and c_r. This leads to:

$$-ap_x + bp_y - cv_x + dv_y - e\theta + f\omega < 0. \quad (27)$$

By performing a grouping of the variables similar to the previous conditions we obtain:

$$-a(p_x + v_x) + b(p_y + v_y) - (c - a)v_x + (d - b)v_y - e\theta + f\omega < 0. \quad (28)$$

Then, by denoting with v^{k+1} and v^{k-1} the value of the variable v in the next and the previous timestep respectively, we can write:

$$-ap_x^{k+1}+bp_y^{k+1}-c'v_x+d'v_y-e\theta+f\frac{\theta-\theta^{k-1}}{\tau}<0. \quad (29)$$

Experimental measurement of the τ variable led us to set $\tau = 0.05$. By multiplying all the members by τ we obtain:

$$-\tau ap_x^{k+1}+\tau bp_y^{k+1}-\tau c'v_x+\tau d'v_y-\tau e\theta+f(\theta-\theta^{k-1})<0. \quad (30)$$

Then, by noting that $\tau a \approx 5\cdot 10^{-3}$, $\tau b \approx 6.7\cdot 10^{-3}$, $\tau c' \approx 3.5\cdot 10^{-2}$, $\tau d' \approx 2.6\cdot 10^{-2}$ and $\tau e \approx 10^{-2}$, we can decide to neglect the effects of the first two terms. So we have:

$$-\tau c'v_x+\tau d'v_y+(f-\tau e)\theta-f\theta^{k-1}<0. \quad (31)$$

By merging the terms in θ and θ^{k-1} we obtain:

$$-c'v_x+d'v_y+(f-\tau e)\omega+\tau e\theta^{k-1}<0. \quad (32)$$

By moving all the terms except the one in ω to the second member we get:

$$\omega<\frac{1}{f-\tau e}(c'v_x-d'v_y-\tau e\theta^{k-1}). \quad (33)$$

Then, by noting that all the states that are tested in this condition have $c'|\overline{v_x}| \approx 5d'|\overline{v_y}|$ and $c'|\overline{v_x}| \approx 120e|\overline{\theta^{k-1}}|$ (where $\overline{v}$ is the mean value of the variable v), we can neglect (as shown by experimental results) the effects of v_y and θ^{k-1}. Finally, the condition checked to fire the main engine is:

$$\omega<c''v_x. \quad (34)$$

While we expected the main engine to depend on p_y or v_y, by analyzing the activation of the condition in several episodes we found that this condition represents the landing phase. The goal of this check is to balance angular velocity and linear velocity to make the agent gently stop on the landing pad.

d: Fourth condition

When this condition evaluates to True, it does not fire any engine. Conversely, when it evaluates to False, it fires the main engine.

The condition is the following (also in this case we replace the constants with symbols):

$$ap_x-bp_y-cv_x-dv_y-e\theta+f\omega<0. \quad (35)$$

By analyzing the mean values of the variables and their coefficients we obtain: $a|\overline{p_x}| \approx 8.5\cdot 10^3$, $b|\overline{p_y}| \approx 8.7\cdot 10^3$, $c|\overline{v_x}| \approx 7\cdot 10^2$, $d|\overline{v_y}| \approx 2.5\cdot 10^2$, $e|\overline{\theta}| \approx 1.3\cdot 10^2$, $f|\overline{\omega}| \approx 4.7\cdot 10^2$. This suggests that we can neglect the values of p_x, p_y, and θ because their mean value is lower than the maximum. The experiments confirmed that these variables have a low impact on the performance of the agent.

So, the agent does not fire any engine when:

$$\omega<\frac{c}{f}v_x+\frac{d}{f}v_y. \quad (36)$$

This seems an extension of what we obtained in the previous condition, where we also have a dependency from v_y. Moreover, it is important to note that this check is performed only when the third condition is not True. Finally, from experiments, we observed that this condition is True usually when the agent has successfully landed. In this case, the terms in c_l and c_r can be seen as a further margin to the agent, so that when a leg touches the ground the agent is more likely to not fire any engine.

In the opposite case, i.e., when $\omega \geq c'v_x + d'v_y$, the agent turns on the main engine to balance the high angular velocity of the lander. Note, again, that if the angular velocity is too low, the lander is balanced by the previous condition.

VI. EXAMPLE APPLICATION

To show the potential applicability of our method in high-stakes domains, we conclude our experimentation with an application related to pandemic control.

In [6], the authors propose a pandemic simulator to optimize policies to reduce both the spreading and the economic damage due to COVID-19 (in the following, we refer to this environment as "PandemicSimulator"). In the same study, they consider as baseline a simple policy that always applies the same restriction level, as well as three handmade policies, and finally, they propose a deep reinforcement learning policy to control the diffusion of the pandemic. These policies are briefly described below:

- Stage i: always apply stage i restrictions (See Table 6);
- S0-4-0: Apply stage 0 at the beginning. When the number of infected people reaches the 10% apply stage 4 and then, after 30 days, go back to stage 0;
- S0-4-0GI: similar to S0-4-0, with a gradual return to stage 0, moving back one stage every 10 days;
- S0-4-0FI: similar to S0-4-0GI, but the restrictions move a level back every 5 days;
- Soft Actor-Critic: uses a Deep Neural Network trained using the Soft Actor-Critic method.

In this task, the agent represents a national decision maker (e.g., the government) that, based on the information regarding the pandemic, has to make decisions about the regulations to apply. The state of the environment is composed of: the status of the pandemic (Infected, Recovered, Critical, Dead, None); daily report (Infected, Recovered, Critical, Dead, None); current stage (referred to the restrictions applied); and a binary value that indicates whether hospitals are saturated. Note that "None" refers to people that have never been infected. The actions that an agent can take correspond to 5 stages, shown in Table 6. Finally, the reward is the weighted sum between two objectives: one that seeks to minimize the number of people in critical conditions (proportional to the hospital's capacity), and one that tries to minimize the severity of the regulations applied (see [6] for more details on the reward formulation). By maximizing this reward, the agent is encouraged to find a trade-off between the number of critical cases and economic damage.

TABLE 6: Stages for COVID regulations used as actions for the regulatory agents.

Stage	Stay home	Mask	Social distancing	Gatherings	Closed activities
0	No	No	No	Allowed	None
1	Yes	Yes	No	Low risk	None
2	Yes	Yes	Light	Moderate risk	Schools, hairstylists
3	Yes	Yes	Moderate	Not allowed	Schools, hairstylists
3	Yes	Yes	Heavy	Not allowed	Schools, hairstylists, offices, stores

To evolve interpretable agents, we employ an orthogonal grammar that performs splits on the normalized variables. To normalize the status of the pandemic, we divide each value by the sum of all the values, and we normalize the current stage by dividing it by the number of stages.

Note that, in preliminary experiments, we observed that the actual values of the daily report and the number of people in critical conditions (w.r.t. the hospital's capacity) were not crucial to the agents' performance. For this reason, for these two features, we only allow for the evolution of conditions in the form $x > 0$.

Our final results are summarized in Table 7, in comparison to the aforementioned policies proposed in [6]. Additional results are provided in the Supplementary Material. The table shows the average results obtained across 10 runs for both the best score and the best $\mathcal{M}$.

Our best DT is shown in Figure 4, which can be interpreted as follows. If the number of daily recovered patients is equal to zero (i.e., we are in the initial phase of the pandemic), then the policy applies stage 3 restrictions to stop the pandemic at the beginning. Otherwise, the policy checks if the number of daily deaths is equal to zero (i.e., the pandemic is beginning to fade away). If so, it does not apply any restriction, otherwise, it applies stage 2 restrictions to keep the pandemic under control.

While the evolved policy is very simple, it can largely outperform all the other policies described earlier, as shown in Figure 5 and Table 7. On the other hand, it is important to note that there is no guarantee that such a policy would be more effective also in the real world. For such validation, a separate study would be needed.

VII. DISCUSSION

In the first part of our experimentation, we considered three classic control tasks from OpenAI Gym. Our results show that the proposed approach can generate DTs that are comparable to, or even better than, the non-interpretable state-of-the-art (from the performance point of view) while having significantly better interpretability. Moreover, we have shown that the produced agents can be practically interpreted, demonstrating the advantage of interpretable models w.r.t. black boxes.

These advantages are even more evident in the second part of the experimentation, where we considered a high-stakes problem related to pandemic control. The policy evolved by our method resulted not only much easier than state-of-the-art methods based on deep reinforcement learning, but also more effective. At the same time, the obtained policy was more effective than the handmade policies presented in [6].

Overall, our suggestion is that the widely thought performance-interpretability trade-off does not always hold (as also suggested e.g. in [2]), and that interpretable models can be competitive with state-of-the-art techniques, even on hard-to-solve high-stakes RL tasks such as pandemic control. For this reason, we believe that research in this field must be encouraged.

One of the points of attention of this study regards the metric of interpretability adopted for our comparative analysis. While we expect that changing the metric of interpretability should not significantly affect the difference (in terms of the measure of complexity) between our models and those from the literature (especially the black-box ones), we believe that future work should nevertheless focus on more tailored interpretability metrics (i.e., tailored on machine learning models).

TABLE 7: Comparison of the solutions obtained by using the proposed approach and the state-of-the-art in the PandemicSimulator environment. Note that (*) indicates that the numerical result is not publicly available and it has been approximated from graphical data reported in [6].

Method	Score	$\mathcal{M}$
Stage 0	-6.09	0.0
Stage 1	-6.78	0.0
Stage 2	-4.80	0.0
Stage 3	-6.80	0.0
Stage 4	-10.28	0.0
S0-4-0	-4.70	9.9
S0-4-0FI	-3.84	49.7
S0-4-0GI	-3.77	49.7
Soft Actor-Critic (*)	-5.00	34542.1
Orthogonal DT (Best score)	**-2.37**	71.2
Orthogonal DT (Best $\mathcal{M}$)	**-3.14**	35.6

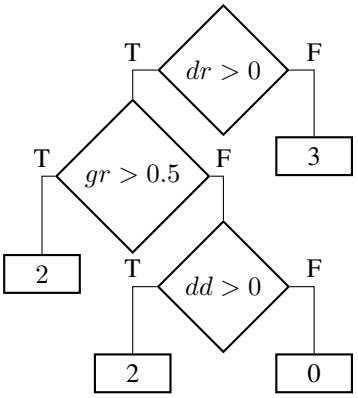


FIGURE 4: Best DT evolved for the PandemicSimulator environment. dr stands for "daily recovered"; gr stands for "globally recovered", meaning the number of people recovered in the whole pandemic so far; dd stands for "daily deaths".

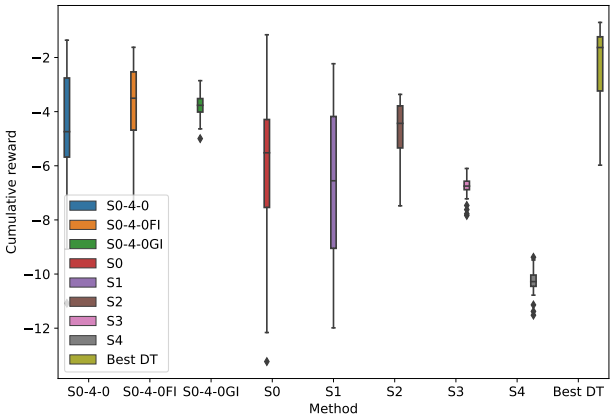


FIGURE 5: Cumulative reward obtained in the Pandemic-Simulator environment by all the approaches considered in the experimentation. All the results for the compared policies are taken from [6]. Note that the "Soft Actor-Critic" approach proposed in [6] has not been shown because the corresponding data are not publicly available. "Best DT" refers to the best DT w.r.t. score found with our method.

VIII. CONCLUSIONS

While in recent years AI has made significant progress, understanding *how* a model works is becoming more and more critical. To overcome this limitation, many efforts have been made to advance the field of XAI. However, XAI is not always a suitable solution. XAI methods typically provide only a-posteriori explanations, and their use can be unsafe in safety-critical or high-stakes processes.

IAI, instead, consists of using transparent approaches to have a complete understanding of what happens in the model, a priori. However, these models are not widely used in practice because of their widely thought lower performance.

In this paper, we proposed a two-level optimization method for inducing DTs in RL settings. The trees obtained demonstrate comparable (or, superior) performance to state-of-the-art approaches, while having a much higher interpretability, as demonstrated by the results shown in the results' Section.

Future research directions include: broader experimentation on more complex RL domains; the extension of the proposed method to the imitation learning domain; the development of an algorithm that can automatically tune the constants, reducing the prior knowledge included in the grammar; a flexible grammar that easily allows oblique trees to become orthogonal, to automatically choose the appropriate type of splits depending on the problem.

REFERENCES

[1] 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, and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 82–115, Jun. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1566253519308103

[2] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, May 2019. [Online]. Available: https://www.nature.com/articles/s42256-019-0048-x

[3] A. Silva, M. Gombolay, T. Killian, I. Jimenez, and S.-H. Son, "Optimization Methods for Interpretable Differentiable Decision Trees Applied to Reinforcement Learning," in *International Conference on Artificial Intelligence and Statistics*, Jun. 2020, pp. 1855–1865. [Online]. Available: http://proceedings.mlr.press/v108/silva20a.html

[4] Y. Dhebar, K. Deb, S. Nageshrao, L. Zhu, and D. Filev, "Interpretable-AI Policies using Evolutionary Nonlinear Decision Trees for Discrete Action Systems," *arXiv:2009.09521*, 2020. [Online]. Available: http://arxiv.org/abs/2009.09521

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv:1606.01540*, 2016.

[6] V. Kompella*, R. Capobianco*, S. Jong, J. Browne, S. Fox, L. Meyers, P. Wurman, and P. Stone, "Reinforcement learning for optimization of COVID-19 mitigation policies," 2020.

[7] B. Shams and M. Khansari, "Immunization of complex networks using stochastic hill-climbing algorithm," in *ICCKE 2013*, Oct. 2013, pp. 283–288.

[8] I. De Falco, A. Della Cioppa, U. Scafuri, and E. Tarantino, "Coronavirus Covid-19 spreading in Italy: optimizing an epidemiological model with dynamic social distancing through Differential Evolution," Apr. 2020, arXiv:2004.00553 [physics, q-bio]. [Online]. Available: http://arxiv.org/abs/2004.00553

[9] A. K. Mccallum and D. Ballard, "Reinforcement learning with selective perception and hidden state," Ph.D. dissertation, The University of Rochester, Eastman School of Music, 1996.

[10] W. T. Uther and M. M. Veloso, "Tree based discretization for continuous state space reinforcement learning," in *AAAI/IAAI*, 1998, pp. 769–774.

[11] L. D. Pyeatt and A. E. Howe, "Decision tree function approximation in reinforcement learning," Colorado State University, Tech. Rep., 1998.

[12] A. M. Roth, N. Topin, P. Jamshidi, and M. Veloso, "Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy," *arXiv:1907.01180*, 2019.

[13] R. Paleja, Y. Niu, A. Silva, C. Ritchie, S. Choi, and M. Gombolay, "Learning Interpretable, High-Performing Policies for Autonomous Driving," May 2022, arXiv:2202.02352 [cs]. [Online]. Available: http://arxiv.org/abs/2202.02352

[14] A. Ghosh, Y. Dhebar, R. Guha, K. Deb, S. Nageshrao, L. Zhu, E. Tseng, and D. Filev, "Interpretable ai agent through nonlinear decision trees for lane change problem," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2021, pp. 01–08.

[15] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon, "Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution," in *Applications of Evolutionary Computation*. Berlin, Heidelberg: Springer, 2011, pp. 123–132. [Online]. Available: http://link.springer.com/10.1008/978-3-642-20525-5_13

[16] C. Ryan, J. Collins, and M. O. Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *European Conference on Genetic Programming*. Berlin, Heidelberg: Springer, 1998, pp. 83–96. [Online]. Available: http://link.springer.com/10.1007/BFb0055930

[17] A. Hallawa, T. Born, A. Schmeink, G. Dartmann, A. Peine, L. Martin, G. Iacca, A. E. Eiben, and G. Ascheid, "EVO-RL: Evolutionary-Driven Reinforcement Learning," *arXiv:2007.04725 [cs, stat]*, Jul. 2020, arXiv: 2007.04725. [Online]. Available: http://arxiv.org/abs/2007.04725

[18] M. Krętowski, "A memetic algorithm for global induction of decision trees," in *International Conference on Current Trends in Theory and Practice of Computer Science*. Berlin, Heidelberg: Springer, 2008, pp. 531–540.

[19] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, Mass: MIT Press, 1992.

[20] M. Czajkowski and M. Krętowski, "A multi-objective evolutionary approach to Pareto-optimal model trees," *Soft Computing*, vol. 23, no. 5, pp. 1423–1437, Mar. 2019. [Online]. Available: https://doi.org/10.1007/s00500-018-3646-3

[21] X. Llora and J. M. Garrell, "Evolution of decision trees," in *4th Catalan Conference on Artificial Intelligence*, 2001, pp. 115–122.

[22] X. Llorà and J. M. Garrell, "Knowledge-independent data mining with fine-grained parallel evolutionary algorithms," in *3rd Annual Conference on Genetic and Evolutionary Computation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, p. 461–468.

[23] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is NP-complete," *Information Processing Letters*, vol. 5, no. 1, pp. 15–17, May 1976. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0020019076900958

[24] G. Fan and J. B. Gray, "Regression tree analysis using TARGET," *Journal of Computational and Graphical Statistics*, vol. 14, no. 1, pp. 206–218, 2005.

[25] J. M. Baldwin, "A new factor in evolution," *Adaptive individuals in evolving populations: Models and algorithms*, pp. 59–80, 1896.

[26] C. Watkins, "Learning From Delayed Rewards," Ph.D. dissertation, King's College, May 1989.

[27] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "Deap: Evolutionary algorithms made easy," *J. Mach. Learn. Res.*, vol. 13, no. 1, p. 2171–2175, Jul. 2012.

[28] M. Virgolin, A. De Lorenzo, E. Medvet, and F. Randone, "Learning a formula of interpretability to learn interpretable formulas," in *Parallel Problem Solving from Nature – PPSN XVI*, T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, and H. Trautmann, Eds. Cham: Springer International Publishing, 2020, pp. 79–93.

[29] Z. C. Lipton, "The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery," *Queue*, vol. 16, no. 3, p. 31–57, Jun. 2018. [Online]. Available: https://doi.org/10.1145/3236386.3241340

[30] P. Barceló, M. Monet, J. Pérez, and B. Subercaseaux, "Model interpretability through the lens of computational complexity," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[31] W. Meng, Q. Zheng, L. Yang, P. Li, and G. Pan, "Qualitative Measurements of Policy Discrepancy for Return-Based Deep Q-Network," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–7, 2019.

[32] J. Xuan, J. Lu, Z. Yan, and G. Zhang, "Bayesian Deep Reinforcement Learning via Deep Kernel Learning," *International Journal of Computational Intelligence Systems*, vol. 12, no. 1, pp. 164–171, Nov. 2018. [Online]. Available: https://www.atlantis-press.com/journals/ijcis/25905189

[33] R. Beltiukov, "Optimizing Q-Learning with K-FAC Algorithm," in *International Conference on Analysis of Images, Social Networks and Texts*. Cham: Springer, 2020, pp. 3–8. [Online]. Available: http://link.springer.com/10.1007/978-3-030-39575-9_1

[34] J. Liu, X. Gu, S. Liu, and D. Zhang, "Soft Q-network," *arXiv:1912.10891 [cs]*, 2019.

[35] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, "Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning," *arXiv:1910.00177 [cs, stat]*, Oct. 2019, arXiv: 1910.00177. [Online]. Available: http://arxiv.org/abs/1910.00177

[36] Z. Xu, L. Cao, and X. Chen, "Deep Reinforcement Learning with Adaptive Update Target Combination," *The Computer Journal*, vol. 63, no. 7, pp. 995–1003, Jul. 2020. [Online]. Available: https://academic.oup.com/comjnl/article/63/7/995/5543068

[37] M. Malagon and J. Ceberio, "Evolving Neural Networks in Reinforcement Learning by means of UMDAc," *arXiv:1904.10932 [cs]*, Apr. 2019, arXiv: 1904.10932. [Online]. Available: http://arxiv.org/abs/1904.10932

LEONARDO L. CUSTODE is a PhD student at Department of Information Engineering and Computer Science of the University of Trento, Italy. He obtained his Bachelor's degree in computer engineering in 2017 and a Master's degree in information engineering in 2019, both at the University of Salerno, Italy. His main areas of interest are: reinforcement learning, program synthesis, and optimization. He is currently working on methodologies for training interpretable AI models that can be easily understood and inspected by humans.

GIOVANNI IACCA (M'2011) is an Associate Professor at the Department of Information Engineering and Computer Science of the University of Trento, Italy. Previously, he worked as post-doctoral researcher in Germany (RWTH Aachen, 2017-2018), Switzerland (University of Lausanne and EPFL, 2013-2016) and The Netherlands (INCAS3, 2012-2016), as well as in industry in the field of industrial automation. He was co-PI of the EU project "PHOENIX" (2015-2019), and currently is co-PI of the EU project "SUSTAIN" (2022-2026). He has received two best paper awards (EvoApps 2017 and UKCI 2012). His research focuses on computational intelligence, stochastic optimization, and distributed systems. To date, he co-authored more than 110 peer-reviewed publications. He is actively involved in the organization of tracks and workshops at some of the top conferences in the field of computational intelligence. He also regularly serves as reviewer for several journals and conference committees.