# Evolutionary learning of interpretable decision trees

Leonardo Lucio Custode, Giovanni Iacca

*Department of Information Engineering and Computer Science*
*University of Trento*
*Via Sommarive 9, 38123 Povo (Trento), Italy*

**Abstract**

Reinforcement learning techniques achieved human-level performance in several tasks in the last decade. However, in recent years, the need for interpretability emerged: we want to be able to understand how a system works and the reasons behind its decisions. Not only we need interpretability to assess the safety of the produced systems, we also need it to extract knowledge about unknown problems. While some techniques that optimize decision trees for reinforcement learning do exist, they usually employ greedy algorithms or they do not exploit the rewards given by the environment. This means that these techniques may easily get stuck in local optima. In this work, we propose a novel approach to interpretable reinforcement learning that uses decision trees. We present a two-level optimization scheme that combines the advantages of evolutionary algorithms with the advantages of Q-learning. This way we decompose the problem into two sub-problems: the problem of finding a meaningful and useful decomposition of the state space, and the problem of associating an action to each state. We test the proposed method on three well-known reinforcement learning benchmarks, on which it results competitive with respect to the state-of-the-art in both performance and interpretability. Finally, we perform an ablation study that confirms that using the two-level optimization scheme gives a boost in performance in non-trivial environments with respect to a one-layer optimization technique.

*Email addresses:* leonardo.custode@unitn.it (Leonardo Lucio Custode), giovanni.iacca@unitn.it (Giovanni Iacca)

*Keywords:* Reinforcement Learning, Decision Tree, Interpretability, Evolutionary algorithm

## 1. Introduction

While machine learning is achieving very promising results in a variety of fields, there is an emergent need to understand what happens in the learned model, for testing, security and safety purposes.

There are mainly two approaches that try to address this problem: explainable AI (XAI) and interpretable AI (IAI) (which is actually a subset of XAI).

The field of explainable AI, in recent years, has seen a significant increase in the number of scientific contributions related to the topic. It is important to note, however, that these techniques are not applicable to all the tasks. In fact, as stated by [1], it is not safe to apply XAI techniques to safety-critical or high-stakes systems. This is due to the fact that explanations are usually *approximations* of the actual models and, as a consequence, do not represent *exactly* the models.

Interpretable AI techniques, instead, are based on the use of interpretable models, i.e. models that can be easily understood and inspected by an human operator [2]. These techniques, besides the ability to assess security and safety of the produced models, can also serve to better *understand* a problem. In fact, by looking at an interpretable model (with good performance), an human operator can *extract* knowledge about the problem fac

However, interpretable AI techniques are not widely used in practice, due to their (usually) lower performance. In fact, it is widely accepted (although not proved) that a trade-off between interpretability and performance exists.

Recent work has addressed the problem of building interpretable reinforcement learning models. In [3], the authors implement a differentiable version of decision trees and optimize them by using backpropagation. Dhebar et al. [4] propose nonlinear decision trees to approximate and refine an oracle policy.

While the results of these approaches seem very promising, the structure of the tree must be defined a-priori. This requires us to either perform a trial-and-error search or to include prior knowledge.

In this work, we present a novel approach to the training of interpretable reinforcement learning agents that combines artificial evolution and lifelong reinforcement-learning. This two-level optimization algorithm allows us to decrease the amount of prior knowledge given to the algorithm. Our approach is able to generate agents in the form of decision trees that are able to learn both a decomposition of the space and the state-action mapping.

The contributions of this paper are the following:

- We propose a two-level optimization approach that optimizes both the topology of the tree and the decisions taken for each state
- We perform experimental tests on classic reinforcement learning problems: CartPole, MountainCar and LunarLander
- We perform a comparison of the produced agents w.r.t. the interpretable and the non-interpretable state-of-the-art
- We quantitatively measure the interpretability of our solutions and compare it to the state-of-the-art
- We interpret the solutions produced to understand how the agents work

This paper is structured as follows. In Section 2 we give some background on the field and related work, while Section 3 describes the method used in our approach. Then, in Section 4 we present the results of our experiments. In Section 5 we will discuss our results by comparing them to the interpretable state-of-the-art, performing an ablation study and interpreting the produced solutions. Finally, in Section 6 we draw the conclusions of this work.

## 2. Related work

In this section we will give some background on the research problem being faced.

The use of decision trees to learn in reinforcement learning tasks has been explored in several previous work .

McCallum, in [5], proposes U-Trees: a kind of trees able to perform reinforcement learning that handle the following sub-problems: choice of the memories, selective perception and value function approximation. In [6], the authors extend U-Trees in order to make them able to cope with continuous environment. 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 respect to other approaches.

Pyeatt and Howe, in [7], propose a novel splitting criterion to build trees that are able to 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 [8] the authors propose a method that predicts the gain obtained by adding a split to the tree and select the best split to grow the tree. The experimental results show that this method is more effective than the method proposed in [7] on the tested environment.

Silva et al. [3] propose an approach to interpretable reinforcement learning that uses Proximal Policy Optimization on differentiable decision trees. Moreover, they provide an analysis of the learning process while using either Q-learning or PPO. The experimental results show that this approach is able to produce competitive solutions in some of the tasks. However, it is also shown that when discretizing the differentiable decision trees into typical decision trees, the performance may heavily decrease.

In [4], the authors used evolutionary algorithms to evolve non-linear decision trees. 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 is able to obtain competitive performance with respect to a neural network based approach.

In [9], the authors use the grammatical evolution algorithm [10] to evolve behavior trees (tree-based structures that allow more complex operations than a decision tree) for the Mario AI competitions. The proposed agent can perform basic actions or pre-determined combinations of basic actions. Their solution achieved the fourth place in the Mario AI competition. However, in this work 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 [11], use behavior trees as evolved instinctive behaviors in agents that are then combined with a learned behavior. While behavior trees are usually interpretable, the authors did not take explicitly into account the interpretability of the whole model, which comprises both a behavior tree and either a neural network or a Q-learning table.

Several work applied the evolutionary computation paradigm to evolve tree-based structures outside the reinforcement learning domain.

Krętowski, in [12], proposes a memetic algorithm based on genetic programming [13] and local search to optimize decision trees. The results presented show that this approach is able to obtain performance that is comparable to the state-of-the-art, while keeping the size of the tree significantly lower.

In [14], the authors propose a multi-objective EA to evolve regression trees and model trees. They use a Pareto front to optimize RMSE, number of nodes, number of attributes. The experimental results show that this approach is able to obtain performance that are comparable or better than the state-of-the-art while using less nodes and less attributes.

In [15, 16], the authors use the Genetic and Artificial Life Environment (GALE) to evolve decision trees. Their results show that GALE is able to produce decision trees that are competitive with the state-of-the-art.

## 3. Method

In this work, we aim to produce interpretable agents in the form of decision trees. Decision trees are trees (usually binary trees) where each inner node represents a “split” (i.e. a test on a condition) and each leaf node contains a decision. A representation of the proposed decision trees is shown in Figure 1.

When using decision trees for reinforcement learning tasks, there are two problems that need to be assessed:

1. How do we choose the splits?
2. Given a leaf, what action do we need to assign to this leaf?

Of course, there is an important relationship between splits and decisions taken in the leaves, so changing one of these without changing the other may lead to significant changes in performance.

Several works [8, 5, 6, 7] use greedy heuristics to induce the trees. However, this approaches have the following drawbacks:

- They use greedy rules to expand the trees: since inducing decision trees is an NP-complete problem [17], this may cause the induction of sub-optimal trees (i.e. stuck in local optima) of poor quality [5, 18].

- They use tests to expand the trees: 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 [5, 6].

Other works [9] (and [12, 14, 15, 16], even if they are not applied to reinforcement learning tasks) induce trees by means of evolutionary approaches. However, these approaches only rely on the evolutionary algorithm. In RL tasks, not exploiting the reinforcement signals given by the environment may slow down the evolution and so result in a less-efficient process.

Our approach, instead, aims to combine artificial evolution and reinforcement-learning methods to take the best of both worlds. We propose a Baldwinian-evolutionary approach to optimize simultaneously the structure of the tree and the state-action function. Baldwinian evolution is an evolutionary theory that, opposed to Darwinian evolution and Lamarckian evolution, states that what an individual learns during his life is not passed to their offspring. However, the

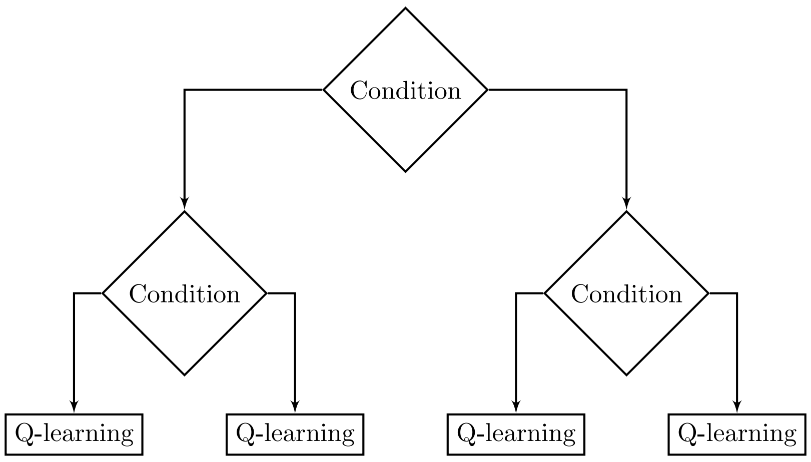

Figure 1: A high-level representation of the proposed agent in the form of a decision tree.

knowledge acquired by the individual may be an evolutionary advantage that modifies the fitness landscape.

We do so by using an evolutionary algorithm to evolve the structure of the decision tree, while using $\mathcal{Q}$-learning to learn the state-action function. This way, we search for trees that decompose the state-space in such a manner that, when taking optimal actions, maximize the reward of the agent.

The evolutionary algorithm we use is the Grammatical Evolution (GE) [10]. This evolutionary algorithm evolves (context-free) grammars in the Backus-Naur Form.

Figure 2 shows a block diagram that clarifies the inner working of the proposed algorithm. The blue-colored parts are the processes inherent the evolutionary part of our algorithm, while the red-colored ones are the processes inherent the reinforcement-learning part.

## *3.1. Evolutionary algorithm*

To evolve decision trees, we evolve an associated grammar, similarly to the approach described in [10]. In this subsection we will describe our algorithm design, highlighting the differences with the original Grammatical Evolution.

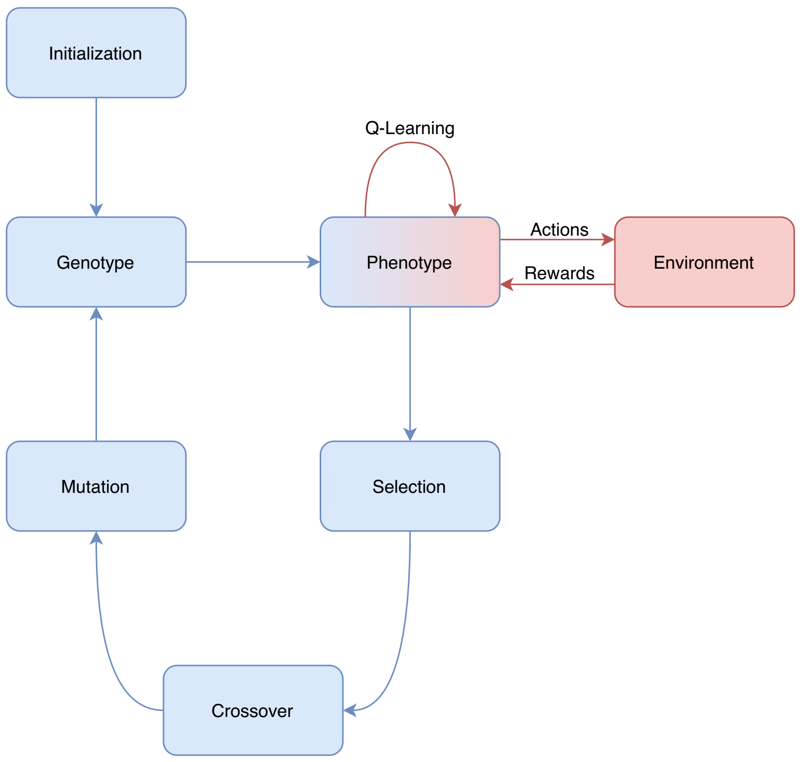

Figure 2: A scheme of the inner working of the proposed algorithm. The blue blocks are the ones that derive from the evolutionary part of our algorithm, while the red blocks are the ones that derive from the Q-learning part.

### *3.1.1. Individual encoding*

The genotype of an individual is encoded as a list of codons (represented as integers). However, differently from [10], the genotype has fixed length.

### *3.1.2. Mutation operator*

Instead of the mutation operator described in [10], we use a classical uniform mutation. This operator mutates each gene according to a probability. The new value of the gene is drawn uniformly from the range of variation of the variable. However, since the grammar may have a different number of productions for each rule, we uniformly draw a random number between 0 and a number bigger than the maximum number of productions in the grammar. Then, by using the modulo operator, we choose the production from the production rule.

### *3.1.3. Crossover operator*

As a crossover operator, we use the standard one-point crossover operator. This operator simply sets a random cutting point and creates two individuals by mixing the two sub-strings of the genotype. This means that we do not prune the individuals that have genes that are not expressed in the phenotype.

### *3.1.4. Replacement of the individuals*

Instead of replacing all the individuals with their offspring (intended as the copies that undergo mutation/crossover), we replace a parent only when the fitness of its offspring is better than the fitness of the parent. Moreover, when an offspring has two parents (i.e. is a product of crossover), it replaces the parent with lowest fitness. In case two offspring have better fitness than only one of the parents, the best one replaces the worst parent.

This mechanism allows us to preserve diversity between the individuals and at the same time makes the fitness trend monotonically increasing.

### *3.1.5. Fitness evaluation*

The fitness evaluation process consists in the following steps. First of all, the genotype is translated to the corresponding phenotype. Then, for each timestep, the policy encoded by the tree is executed and the reward signals obtained from the environment are used to update the Q-values of the leaves.

## 4. Results

To test our approach, we test it in the following OpenAI Gym [19] environments:

- CartPole-v1
- MountainCar-v0
- LunarLander-v2

In this section, we will show the results obtained and compare them to the state-of-the-art using two metrics: the score given by the simulator and a metric of complexity (from the interpretability point of view).

We adopted the interpretability metric proposed in [20]:

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

where:

- $l$ is the size of the formula (i.e. 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 was meant to be bounded between 0 and 100, so we modified the metric in order to make it work as a *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 an higher complexity is more hard to interpret

- We replaced the constant with -0.2, so that when we have a constant (best case from the point of view of the interpretability) its complexity becomes 0

Furthermore, it is important to note that this metric is in line with what Lipton states in [21]. In fact, we can easily note that huge decision trees will be as interpretable as black-box methods, because the terms $l, n_o, n_{nao}$ and $n_{naoc}$ will have a high magnitude. Also, $\mathcal{M}$ seems to be (loosely) in line with what the authors say in [22]. In fact, by using the number of operations (although there are other variables in the metric we use) we loosely resemble the computational complexity of the model that we are executing.

To assess the statistical repeatability of our experiments, we perform 10 independent runs for each setting. For each run, as required by [19], we test the best model for each run over 100 independent episodes to assess its performance. By "testing", we mean that the policy is executed in 100 unseen episodes.

### *4.1. Simplification mechanism*

To make our solutions even more interpretable, we introduce a simplification mechanism that is executed on the final solutions. The simplification mechanism is the following. First of all, we execute the given policy for 100 episodes in a validation environment. Here, we keep a counter for each node of the tree that is increased each time the node is visited. Then, once this phase is finished, 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 such a node is found, it is replaced with a leaf that contains the action common to the two leaves. The iteration stops when the tree does not contain nodes of this type.

### *4.2. Description of the environments*

In this subsection we will describe the environments used and their properties.

### *4.2.1. CartPole-v1*

In this task the agent has to balance a pole that stands on top of a cart by moving the cart either to the left or to the right.

*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: $\omega \in \,]-\infty, \infty[\, rad/s$

*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 ($move\_right$)

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

*Termination criterion.* The simulation terminates if:

- The cart position lies outside the bounds for the $x$ variable
- The angle of the pole lies outside the bounds for the $\theta$ variable

*Resolution criterion.* This task is considered as solved if the agent receives a mean total reward $R \geq 475$ on 100 runs.

### *4.2.2. MountainCar-v0*

In this environment the agent has to drive a car, which is initially in a valley, up on a hill. However, the engine of the car is not powerful enough so the agent has to learn how to build momentum by exploiting the two hills.

*Observation space.* The state of the environment consists in the following variables:

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

*Action space.* The agent can perform 3 actions:

1. Accelerate to the left by applying a force of 0.001N
2. Do not accelerate
3. Accelerate to the right by applying a force of 0.001N

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

*Termination criterion.* The simulation terminates after 200 timesteps.

*Resolution criterion.* This task is considered as solved if the agent receives a mean total reward $R \geq -110$ on 100 runs.

### *4.2.3. LunarLander-v2*

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

*Observation space.* The state of the environment consists of 8 variables:

- Horizontal position: $p_x$
- Vertical position: $p_y$
- Horizontal velocity: $v_x$
- Vertical velocity: $v_y$
- Angle w.r.t. the vertical axis: $\theta$
- Angular velocity: $\omega$
- Left leg contact: $c_l$
- Right leg contact: $c_r$

*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*

*Rewards.* The reward for moving from the initial point to the landing pad with 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 in a timestep gives a reward of -0.3 points, while firing a side-engine gives a reward of -0.03.

*Termination criterion.* The simulator ends if either 1000 timesteps are passed, the lander crashes or it passes the bounds of the environment.

*Resolution criterion.* This task is considered as solved if the mean total reward $R \geq 200$ on 100 runs.

## *4.3. CartPole-v1*

### *4.3.1. Experimental setup*

In this setting, we tested two different grammars: one to evolve *orthogonal* decision trees and one to evolve *oblique* decision trees. Orthogonal decision trees are decision trees in which each condition tests a single variable. This results in hyperplanes that are orthogonal to the axis of the variable tested. On the other hand, oblique decision trees handle multiple variables for each split, resulting in oblique hyperplanes.

The orthogonal and oblique grammars are shown in Tables 1 and 2 respectively. The settings used for the grammatical evolution in the orthogonal and oblique cases are shown in Tables 3 and 4. Finally, the settings used for the Q-learning algorithm are shown in Table 14. All the parameters have been chosen by performing a manual tuning.

| Rule | Production |
|---|---|
| dt | $< if >$ |
| if | $if \ < condition > \ then \ < action > \ else \ < action >$ |
| condition | $input\_var \ < comp\_op > < const_{input\_var} >$ |
| action | $leaf \mid < if >$ |
| comp_op | $lt \mid gt$ |
| $const_x$ | [-4.8, 4.8) with step 0.5 |
| $const_v$ | [-5, 5) with step 0.5 |
| $const_\theta$ | [-0.418, 0.418) with step 0.01 |
| $const_\omega$ | [-0.836, 0.836) with step 0.01 |

Table 1: Grammar used to evolve orthogonal decision trees in the CartPole-v1 environment. The symbol "|" denotes the possibility to choose between different symbols. "comp_op" is a short version of "comparison operator" and "lt" and "gt" are respectively the "less than" and "greater than" operators. *input_var* represents one of the possible inputs in the given environment. Note that each input variable has a separate set of constants.

| Rule | Production |
|---|---|
| dt | $< if >$ |
| if | $if \ < condition > \ then \ < action > \ else \ < action >$ |
| condition | $lt((\sum_{i=1}^{n\_variables} < const > input_i), < const >)$ |
| action | $leaf \mid < if >$ |
| *const* | $[-1, 1]$ with step $10^{-3}$ |

Table 2: Grammar used to evolve oblique decision trees in the CartPole-v1 environment. The symbol "|" denotes the possibility to choose between different symbols. "lt" refers to the "less than" operator.

| Parameter | Value |
|---|---|
| Population size | 200 |
| Generations | 100 |
| Genotype length | 1024 |
| Crossover probability | 0 |
| Mutation probability | 1 |
| Mutation type | Uniform, with gene probability=0.1 |

Table 3: Parameters used for the Grammatical Evolution with orthogonal grammar in the CartPole-v1 environment.

| Parameter | Value |
|---|---|
| Population size | 200 |
| Generations | 50 |
| Genotype length | 100 |
| Crossover probability | 0 |
| Mutation probability | 1 |
| Mutation type | Uniform, with gene probability=0.1 |

Table 4: Parameters used for the Grammatical Evolution with oblique grammar in the CartPole-v1 environment.

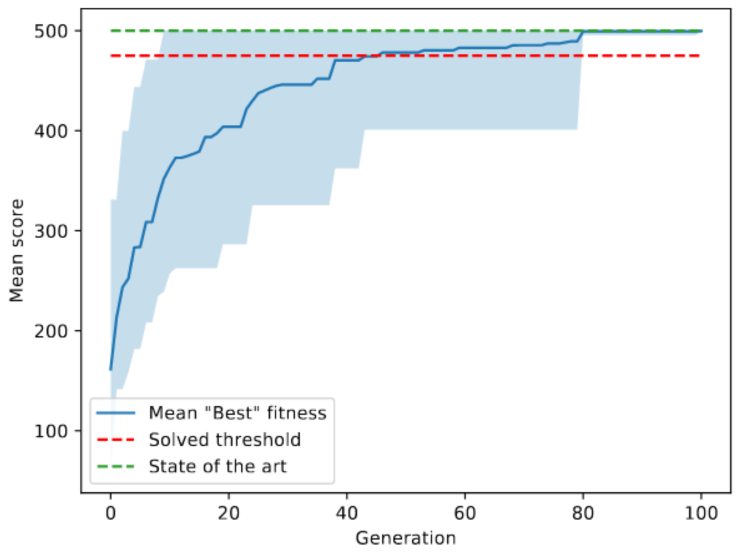

Figure 3: Fitness of the best solution on CartPole-v1, obtained by using the orthogonal grammar, at each generation averaged across 10 runs.

The number of episodes used for Q-learning is quite low. This is because, since this is a “simple” environment, we want to lower the computational cost of the search by *exploiting* the randomness used to initialize the state-action function. This means that, in this case, $\mathcal{Q}$-learning is used to “fine-tune” the state-action function instead of learning it from scratch.

### *4.3.2. Results*

The results are shown in Tables 5 and 6. These tables show an interesting result. In fact, while the orthogonal grammar is able to solve the task in the 100% of the cases, the test score was the optimal one ($500 \pm 0$) only in the 40% of the cases. On the other hand, the oblique grammar solves the task in the 90% of the cases, but achieves the optimal score in the 80% of the runs. This suggests us 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.

| **Run** | **Training mean** | **Testing mean** | **Testing std** | $\mathcal{M}$ |
|---|---|---|---|---|
| R1 | 500.00 | **498.24** | 9.49 | 53.40 |
| R2 | 500.00 | **499.13** | 8.66 | 89.00 |
| R3 | 500.00 | **500.00** | 0.00 | 35.60 |
| R4 | 500.00 | **500.00** | 0.00 | 53.40 |
| R5 | 499.10 | **497.51** | 12.89 | 53.40 |
| R6 | 500.00 | **500.00** | 0.00 | 53.40 |
| R7 | 498.40 | **483.05** | 62.16 | 35.60 |
| R8 | 500.00 | **500.00** | 0.00 | 53.40 |
| R9 | 500.00 | **499.44** | 5.57 | 35.60 |
| R10 | 500.00 | **496.05** | 13.84 | 35.60 |

Table 5: Scores obtained by training interpretable agents on the CartPole-v1 environment by using the orthogonal grammar.

| **Run** | **Training mean** | **Testing mean** | **Testing std** | $\mathcal{M}$ |
|---|---|---|---|---|
| R1 | 500.00 | **500.00** | 0.00 | 24.10 |
| R2 | 500.00 | **495.68** | 42.98 | 24.10 |
| R3 | 500.00 | **500.00** | 0.00 | 48.20 |
| R4 | 500.00 | **500.00** | 0.00 | 24.10 |
| R5 | 500.00 | **500.00** | 0.00 | 24.10 |
| R6 | 500.00 | **500.00** | 0.00 | 24.10 |
| R7 | 500.00 | **500.00** | 0.00 | 24.10 |
| R8 | 500.00 | **500.00** | 0.00 | 24.10 |
| R9 | 500.00 | **500.00** | 0.00 | 24.10 |
| R10 | 500.00 | 460.95 | 132.43 | 24.10 |

Table 6: Scores obtained by training interpretable agents on the CartPole-v1 environment by using the oblique grammar.

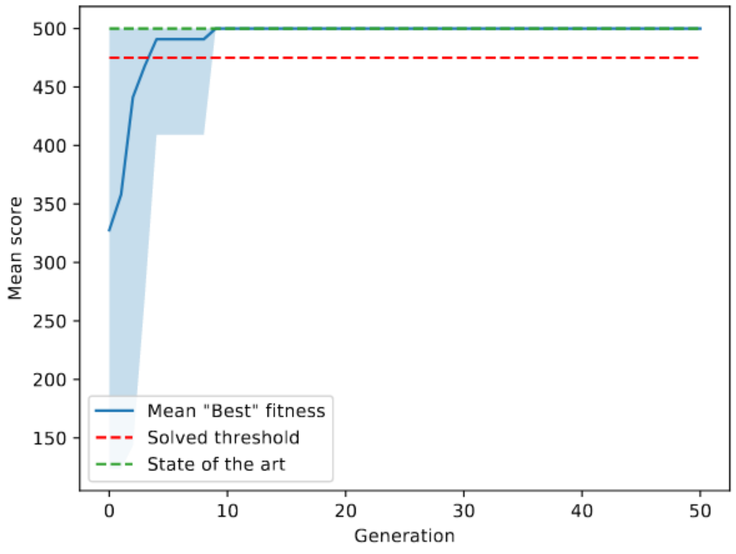

Figure 4: Fitness of the best solution on CartPole-v1, obtained by using the oblique grammar, at each generation averaged across 10 runs.

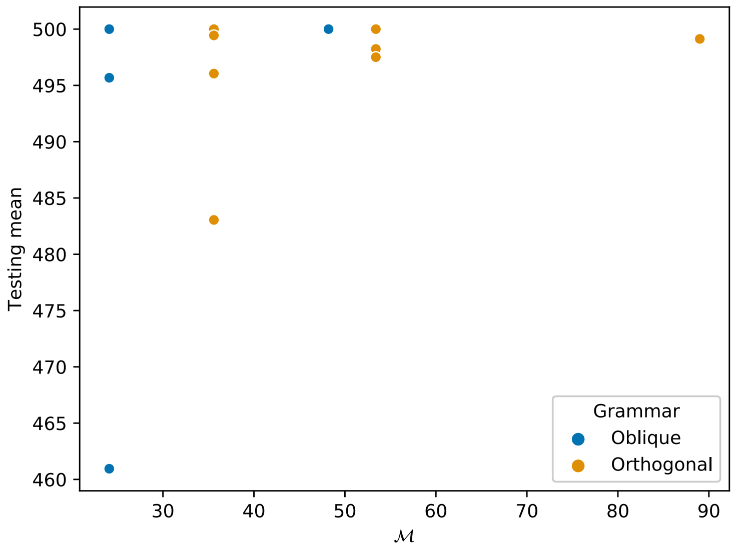

Figure 5: A score-$\mathcal{M}$ plot of the solutions obtained by using the two types of grammars.

In Figures 3 and 4 we show the mean best fitness for each generation averaged across all the runs. We observe that, while both settings are able to converge towards the optimal fitness, the oblique grammar converges more quickly than the orthogonal one to the maximum fitness.

In Figure 5, a comparison of the solutions obtained by using the orthogonal and oblique grammars is shown. We can easily observe that, usually, the solutions obtained with the oblique grammar have a lower $\mathcal{M}$ than the ones obtained by using the orthogonal grammar. This is due to the fact that most of the produced oblique trees use only one split, resulting in a lower number of non-arithmetical operations.

To better assess the hypothesis made earlier, i.e. that oblique trees are more stable than orthogonal ones, in Tables 7 and 8 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 obtaining a perfect score (i.e. $10^4 \pm 0$) also in this setting. Figure 8 shows how the testing mean score varies by varying the number of maximum timesteps for the best agents.

In Figures 6 and 7 we show 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 seems to be stable (according to the Lyapunov's concept of stability) while the orthogonal policy does not.

Moreover, we also tested the robustness of the produced agents with respect to noise on the inputs received by the sensors. In Figure 9 we show how the performance of the two best agents vary with respect to additive input noise ( distributed as $\mathcal{N}(0, \sigma^2)$). The orthogonal tree was robust to noises with $\sigma$ in the order of twice the sampling step used for the constants. On the other hand, the oblique tree proved to be significantly more robust, being able to cope with noises that have a $\sigma$ about 50 times bigger than the sampling step used for the constants.

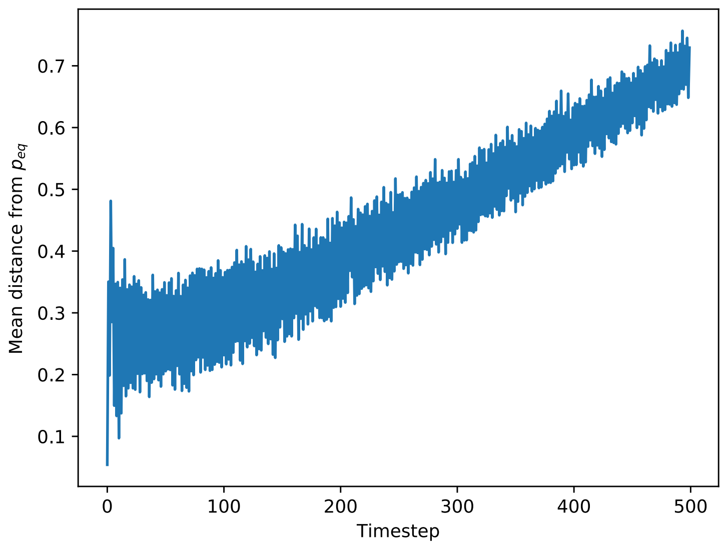

Figure 6: Mean distance of the cart-pole system from the point of equilibrium when using the best orthogonal tree as policy.

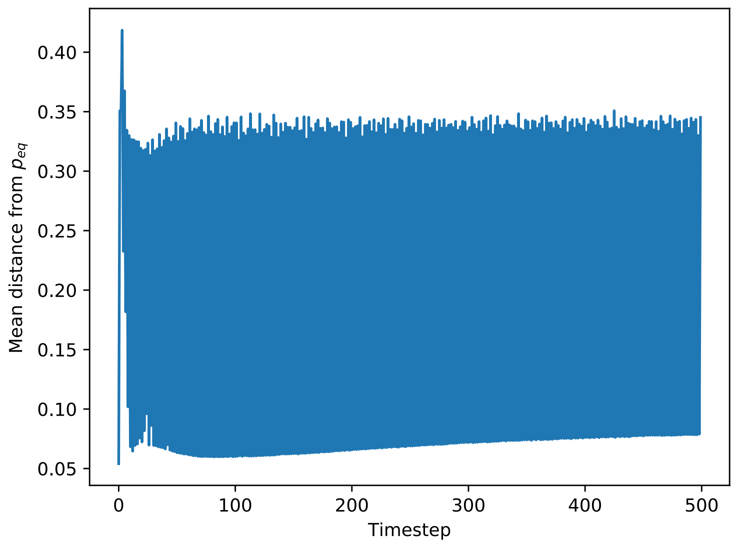

Figure 7: Mean distance of the cart-pole system from the point of equilibrium when using the best oblique tree as policy.

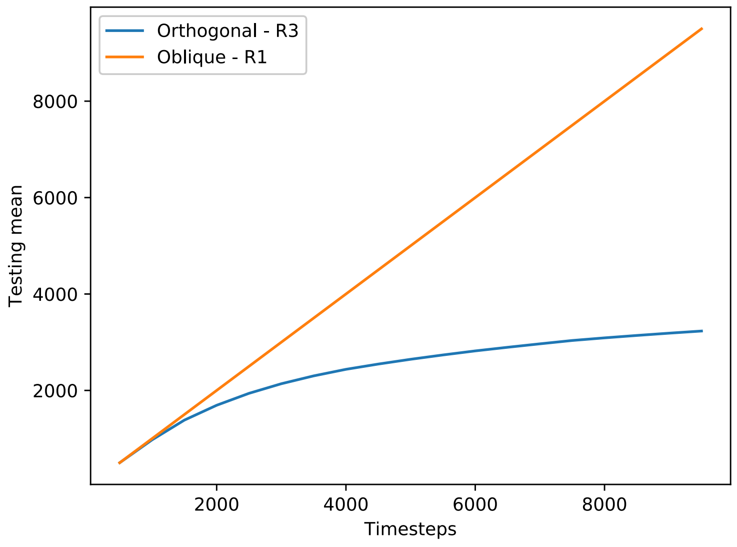

Figure 8: Comparison between the best orthogonal tree with the best oblique at different maximum timesteps for the CartPole-v1 environment

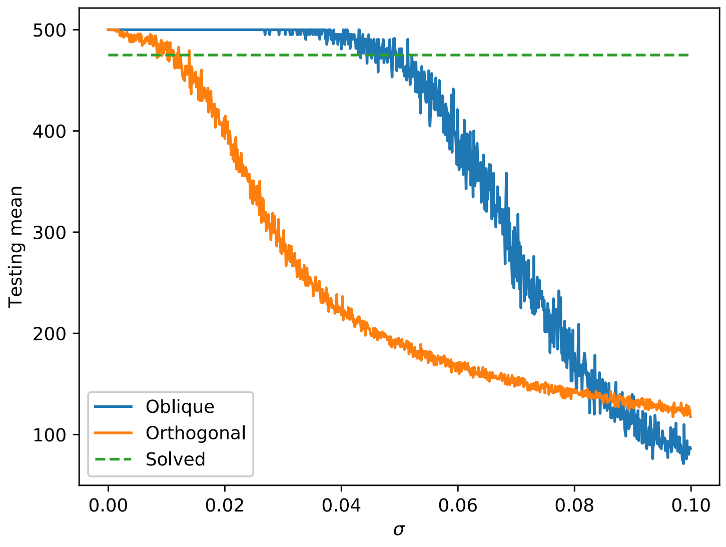

Figure 9: Performance of the two best agents on CartPole-v1 at the variation of the input noise.

| **Run** | **Testing mean** | **Testing std** |
|---|---|---|
| R1 | 878.08 | 346.85 |
| R2 | 767.94 | 202.24 |
| R3 | 3271.99 | 2718.79 |
| R4 | 5845.54 | 2898.37 |
| R5 | 1237.18 | 775.238 |
| 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 |

Table 7: Scores obtained by testing the solutions obtained by using an orthogonal grammar on a $10^4$-steps-long version of CartPole-v1

| **Run** | **Testing mean** | **Testing std** |
|---|---|---|
| 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 |

Table 8: Scores obtained by testing the solutions obtained by using an oblique grammar on a $10^4$-steps-long version of CartPole-v1

Finally, in Table 9 and Figure 10 we compare our best solutions with other solutions found in 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 the network architectures, resulting in a slightly lower complexity. In our opinion, for the purpose of comparing our solution with the non-interpretable state-of-the-art, these small differences are negligible. Our solutions that have been used for the comparison are shown in Figures 11 and 12. The other produced solutions can be found in the repository of the project[1].

[1]https://gitlab.com/leocus/ge_q_dts, accessed: 11 dec 2020.

| **Method** | **Score** | $\mathcal{M}$ |
|---|---|---|
| Deep Q Network [23] | 327.30 | 1157.20 |
| Tree-Backup($\lambda$) [23] | 494.70 | 1157.20 |
| Importance-Sampling [23] | 498.70 | 1157.20 |
| Q$\pi$ [23] | 489.90 | 1157.20 |
| Retrace($\lambda$) [23] | 461.10 | 1157.20 |
| Qualitatively measured policy discrepancy w/ $\beta$ [23] | 499.90 | 1157.20 |
| Qualitatively measured policy discrepancy w/ $\eta$ [23] | 493.20 | 1157.20 |
| Watkins's Q($\lambda$) [23] | 484.30 | 1157.20 |
| Qualitatively measured policy discrepancy w/ $\beta$ [23] | 494.90 | 1157.20 |
| Qualitatively measured policy discrepancy w/ $\eta$ [23] | 493.30 | 1157.20 |
| Peng & Williams's Q($\lambda$) [23] | 496.70 | 1157.20 |
| Qualitatively measured policy discrepancy w/ $\beta$ [23] | **500.00** | 1157.20 |
| Qualitatively measured policy discrepancy w/ $\eta$ [23] | 499.40 | 1157.20 |
| General Q($\lambda$) [23] | 499.90 | 1157.20 |
| Qualitatively measured policy discrepancy w/ $\beta$ [23] | **500.00** | 1157.20 |
| Qualitatively measured policy discrepancy w/ $\eta$ [23] | **500.00** | 1157.20 |
| Deep Q Network [24] | 98.33 | 5170174.80 |
| Bayesian Deep Reinforcement Learning [24] | 113.52 | 8090.40 |
| Bayesian Deep Reinforcement Learning weighted [24] | 136.75 | 8090.40 |
| Kronecker-Factored Approximate Curvature [25] | 321.00 | 70786.20 |
| Differentiable Decision Trees [3] | 388.76 | 89.20 |
| Differentiable Decision Trees [3] (*) | **500.00** | 106.80 |
| Differentiable Decision Trees [3] (**) | **500.00** | **53.40** |
| Ours – Orthogonal | **500.00** | **35.60** |
| Ours – Oblique | **500.00** | **24.10** |

Table 9: Comparison of the solutions obtained by using the proposed approach with respect to the state-of-the-art. The results from [23] are averaged over ten independent runs. The results from [3] regard the discretized tree shown in Figure 23 (From Figure 3 - right 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.

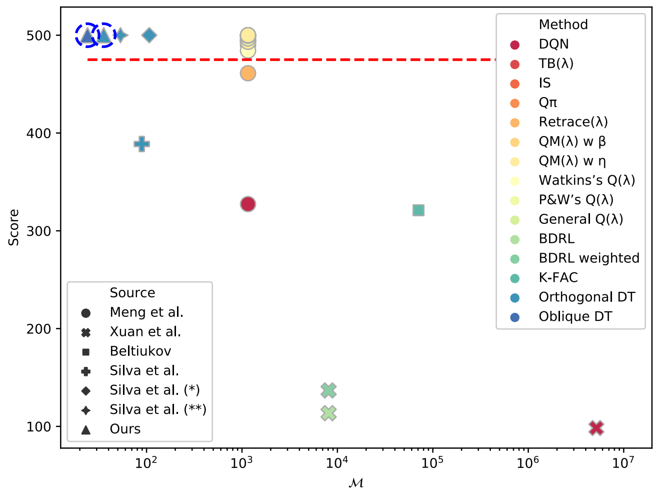

Figure 10: Score-$\mathcal{M}$ comparison of our solutions and the state-of-the-art in the CartPole-v1 environment. It is compared to the ones described in [23], [24] [3], and [25]. (*): Result confirmed by personal communication with the first author of the study. (**): The tree has been simplified by using the technique used in our work.

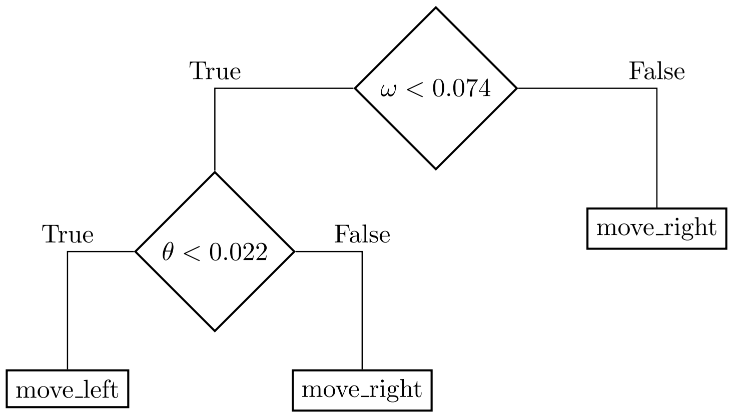

Figure 11: Tree representation of one of the best individuals evolved in the CartPole-v1 environment by using the orthogonal grammar.

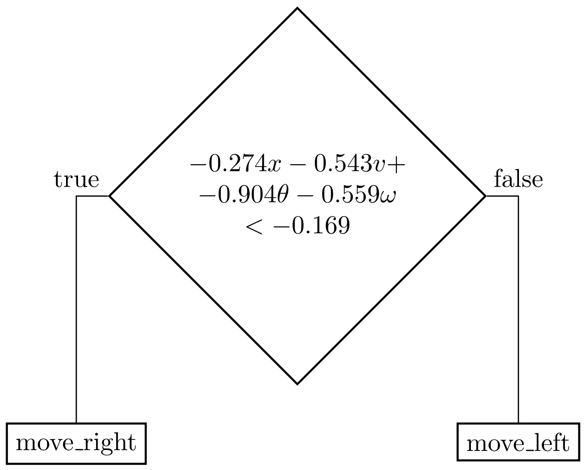

Figure 12: Tree representation of one of the best individuals evolved in the CartPole-v1 environment by using the orthogonal grammar.

### *4.4. MountainCar*

#### *4.4.1. Experimental setup*

Also in this task, we tested both an orthogonal and an oblique grammar. The two grammars are shown in Tables 10 and 11. Note that, in this environment we normalize the variables in the oblique case whereas in the others we do not. This is 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.

The parameters used for the Grammatical Evolution are shown in Tables 12 and 13. The settings for the Q-learning algorithm are shown in Tables 14 and 15. Also in this case, we set the number of episodes to 10 to exploit the randomness of the initialization, since this is considered as a "simple" task.

#### *4.4.2. Results*

The results obtained by the best solution for each run are shown in Tables 16 and 17. In Table 17 there are some values in parenthesis. This is because, given the difference in performance between training and testing scores, we proceeded with further investigation of the results. We deduced that in the latest steps of the training of such agents a change happened in the Q-values

| **Rule** | **Production** |
|---|---|
| dt | $< if >$ |
| if | $if \ < condition > \ then \ < action > \ else \ < action >$ |
| condition | $input\_var \ < comp_op > < const_{input\_var} >$ |
| action | $leaf \mid \ < if >$ |
| comp_op | $lt \mid gt$ |
| $const_x$ | [-1.2, 0.6) with step 0.05 |
| $const_v$ | [-0.07, 0.07) with step 0.005 |

Table 10: Grammar used to evolve orthogonal decision trees in the MountainCar-v0 environment. The symbol "|" denotes the possibility to choose between different symbols. "comp_op" is a short version of "comparison operator" and "lt" and "gt" are respectively the "less than" and "greater than" operators. *input_var* represents one of the possible inputs in the given environment. Note that each input variable has a separate set of constants.

| **Rule** | **Production** |
|---|---|
| dt | $< if >$ |
| if | $if \ < condition > \ then \ < action > \ else \ < action >$ |
| condition | $lt((\sum_{i=1}^{n\_variables} < const > \widehat{input}_i, < const >)$ |
| action | $leaf \mid \ < if >$ |
| *const* | $[-1, 1]$ with step $10^{-3}$ |

Table 11: Grammar used to evolve oblique decision trees in the MountainCar-v0 environment. The symbol "|" denotes the possibility to choose between different symbols. "lt" refers to the "less than" operator. $\widehat{input}_i$ refers to the normalized $input_i$ variable. For the normalization, the bounds [-1.2, 0.7] and [-0.07, 0.07] were used.

| **Parameter** | **Value** |
|---|---|
| Population size | 200 |
| Generations | 1000 |
| Genotype length | 1024 |
| Crossover probability | 0 |
| Mutation probability | 1 |
| Mutation type | Uniform, with gene probability=0.05 |

Table 12: Parameters used for the Grammatical Evolution with orthogonal grammar in the MountainCar-v0 environment.

| **Parameter** | **Value** |
|---|---|
| Population size | 200 |
| Generations | 2000 |
| Genotype length | 100 |
| Crossover probability | 0.1 |
| Crossover operator | One-point crossover |
| Selection operator | Tournament selection with size 2 |
| Mutation probability | 1 |
| Mutation type | Uniform, with gene probability=0.1 |

Table 13: Parameters used for the Grammatical Evolution with oblique grammar in the MountainCar-v0 environment.

| **Parameter** | **Value** |
|---|---|
| Algorithm | $\varepsilon$-greedy Q-learning |
| $\varepsilon$ | 0.05 |
| Initialization strategy | Uniform $\in [-1, 1]$ |
| Learning rate | 0.001 |
| Number of episodes | 10 |

Table 14: Parameters used for the Q-learning algorithm in the CartPole-v1 and MountainCar-v0 (only with orthogonal trees) environments.

| **Parameter** | **Value** |
|---|---|
| Algorithm | $\varepsilon$-greedy Q-learning |
| $\varepsilon$ | 0.01 |
| Initialization strategy | Uniform $\in [-1, 1]$ |
| Learning rate | 0.001 |
| Number of episodes | 10 |

Table 15: Parameters used for the Q-learning algorithm in the MountainCar-v0 environment when evolving oblique trees.

| **Run** | **Training score** | **Testing mean** | **Testing std** | $\mathcal{M}$ |
|---|---|---|---|---|
| R1 | -109.3 | **-106.17** | 4.69 | 89 |
| R2 | -110.5 | **-108.62** | 16.72 | 124.6 |
| R3 | -105.6 | **-102.26** | 9.51 | 71.2 |
| R4 | -108.1 | **-101.72** | 3.14 | 106.8 |
| R5 | -112.9 | -116.15 | 1.03 | 71.2 |
| R6 | -107.2 | **-101.72** | 3.14 | 106.8 |
| R7 | -120.5 | -117.84 | 0.95 | 35.6 |
| R8 | -115.7 | -115.51 | 1.18 | 35.6 |
| R9 | -109.3 | **-106.63** | 4.68 | 89 |
| R10 | -107.1 | **-104.94** | 3.56 | 53.4 |

Table 16: Scores obtained by training interpretable agents on the MountainCar-v0 environment when using the orthogonal grammar.

| **Run** | **Training score** | **Testing mean** | **Testing std** | $\mathcal{M}$ |
|---|---|---|---|---|
| R1 | -108.90 | **-106.66** | 9.30 | 70.00 |
| R2 | -106.50 | -110.18 | 24.90 | 46.60 |
| R3 | -105.60 | **-106.50** | 15.27 | 23.40 |
| R4 | -109.10 | -200.00 (-112.62) | 0.00 (23.08) | 0.00 (46.6) |
| R5 | -106.10 | **-106.06** | 12 | 46.80 |
| R6 | -110.40 | -116.66 | 16.01 | 46.60 |
| R7 | -112.80 | -114.44 | 10.74 | 46.40 |
| R8 | -105.00 | -200.00 (**-107.5**) | 0.00 (13.46) | 0.00 (23.4) |
| R9 | -103.20 | **-106.02** | 15.41 | 46.80 |
| R10 | -111.40 | -116.49 | 16.75 | 46.80 |

Table 17: Scores obtained by training interpretable agents on the MountainCar-v0 environment when using the oblique grammar.

of a leaf. This change made the Q-values of the action taken with the current greedy policy have a value approximately equal to the another action. This caused a destructive change in the policy, so, in order to give more information, we included the test score of the solution by reverting the destructive change. Moreover, this change has only been used in this table. For the remainder of this work, we will assume that their test score is $-200$.

As we can see from 16, the solutions obtained by using the orthogonal grammar solve the task in the 70% of the cases. On the other hand, as we can see from Table 17, oblique trees perform poorly on this problem. This suggests us that this problem is harder 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 grammar (the one used to produce oblique trees) makes it difficult to obtain an orthogonal decision tree.

The fitness trend for the best individual averaged on each run are shown in Figure 13 and 14 for the orthogonal and oblique cases, respectively.

To compare the two approaches, we compare the robustness to input noise for both versions. The result is shown in Figure 15. In this case both approaches proved to be not so robust to noise. Surprisingly, we can observe that the orthogonal tree was not even robust to input noise that had $\sigma < \underset{i}{min}(step_i)$ 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 18 and Figure 16 we show the results of our comparison. The best trees (on testing mean score) that have been used for the comparison are shown in Figures 17 and 18.

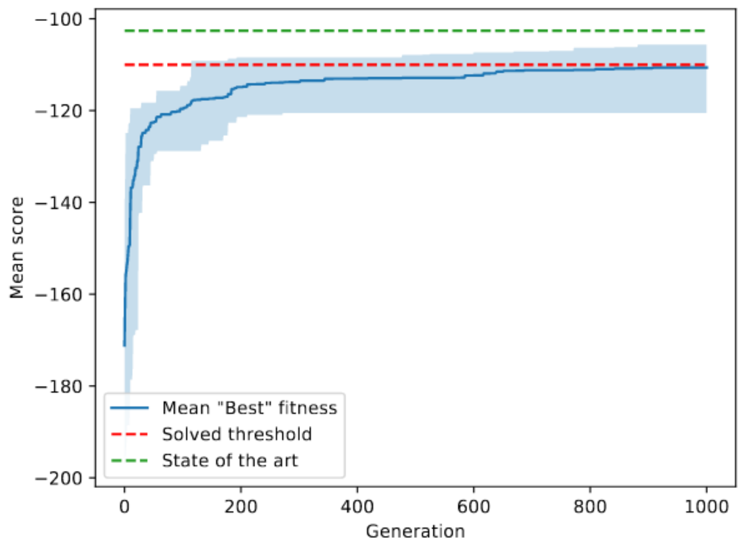

Figure 13: Fitness trend for the best individual in the MountainCar-v0 environment averaged on all the runs, when using orthogonal trees.

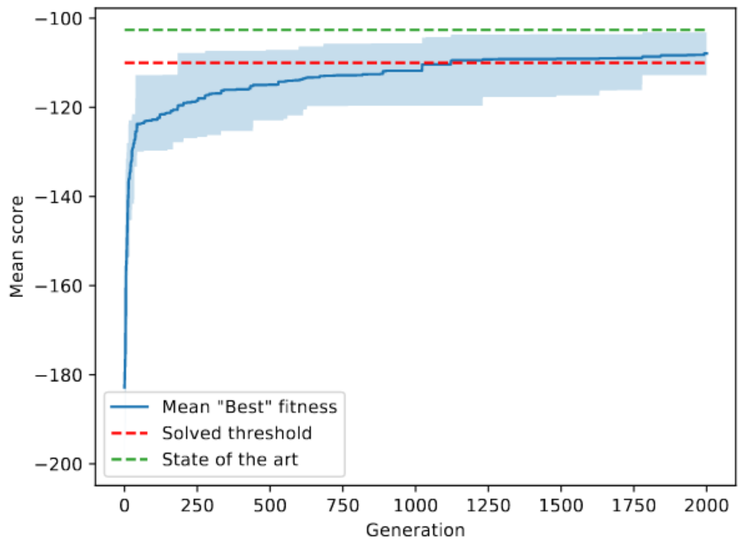

Figure 14: Fitness trend for the best individual in the MountainCar-v0 environment averaged on all the runs, when using oblique trees.

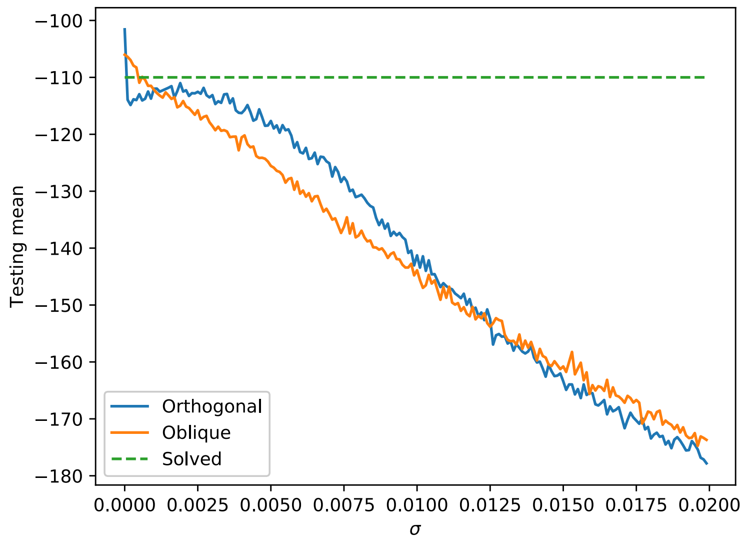

Figure 15: Plot of the mean testing score with different input noises for the best orthogonal and oblique models on MountainCar-v0.

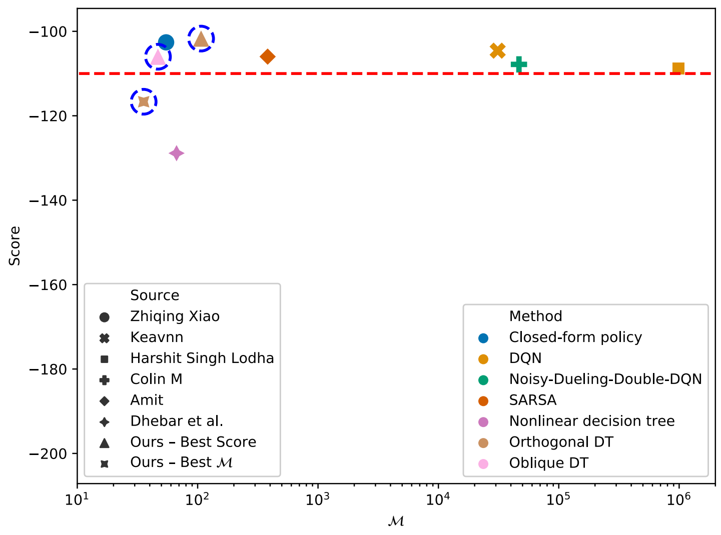

Figure 16: Score-$\mathcal{M}$ comparison of our solutions and the state-of-the-art in the MountainCar-v0 environment.

| **Source** | **Method** | **Score** | $\mathcal{M}$ |
|---|---|---|---|
| Zhiqing Xiao[2] | Closed-form policy | **-102.61** | **54.7** |
| Keavnn[3] | Soft Q Networks [26] | -104.58 | 31079.2 |
| Harshit Singh[4] | Deep Q Network | -108.85 | 984160.3 |
| Colin M[5] | Double Deep Q Network | -107.83 | 46681.6 |
| Amit[6] | Tabular SARSA | -105.99 | 381.5 |
| Dhebar et al. [4] | Nonlinear DT (Open loop) | -128.87 | 66.8 |
| 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 | -200.00 | **0.00** |

Table 18: Comparison of the mean (testing) score of the solutions obtained by using the proposed approach versus the state-of-the-art.

[2]github.com/ZhiqingXiao/OpenAIGymSolution, accessed: 11 dec 2020.
[3]github.com/StepNeverStop/RLs, accessed: 11 dec 2020.
[4]github.com/harshitandro/Deep-Q-Network, accessed: 11 dec 2020.
[5]github.com/CM-Data/Noisy-Dueling-Double-DQN-MountainCar, accessed: 11 dec 2020.
[6]github.com/amitkvikram/rl-agent, accessed: 11 dec 2020.

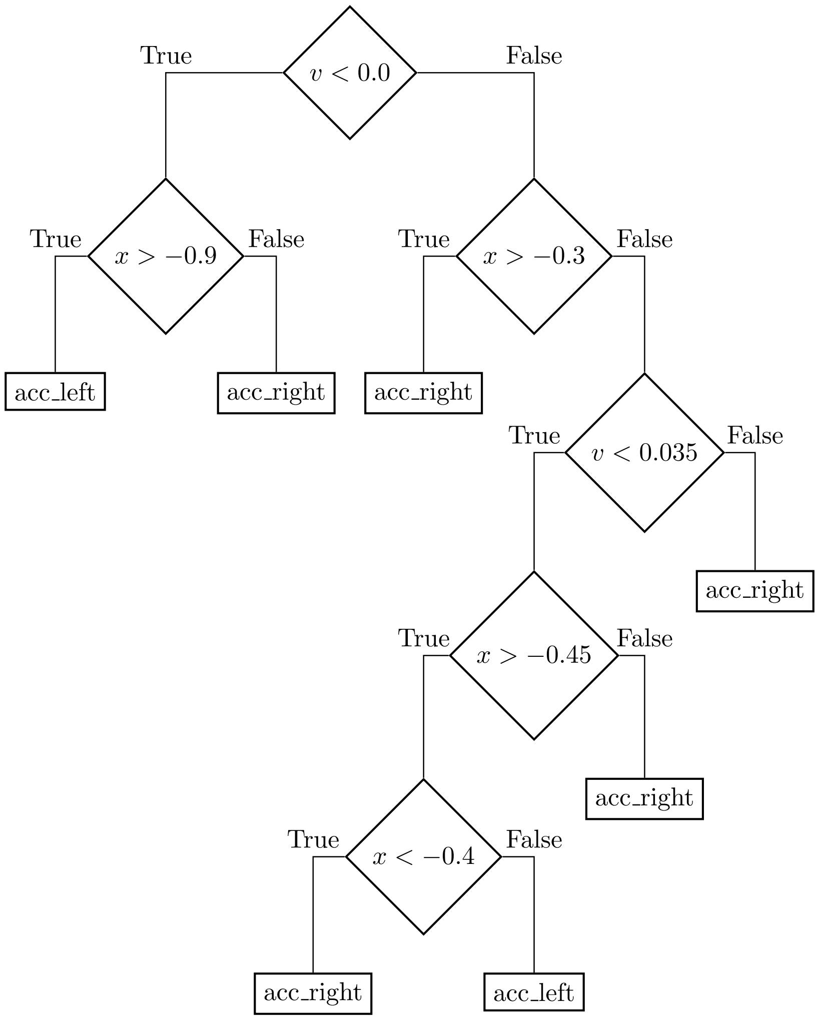

Figure 17: Best orthogonal decision tree (w.r.t. score) evolved in the MountainCar-v0 environment.

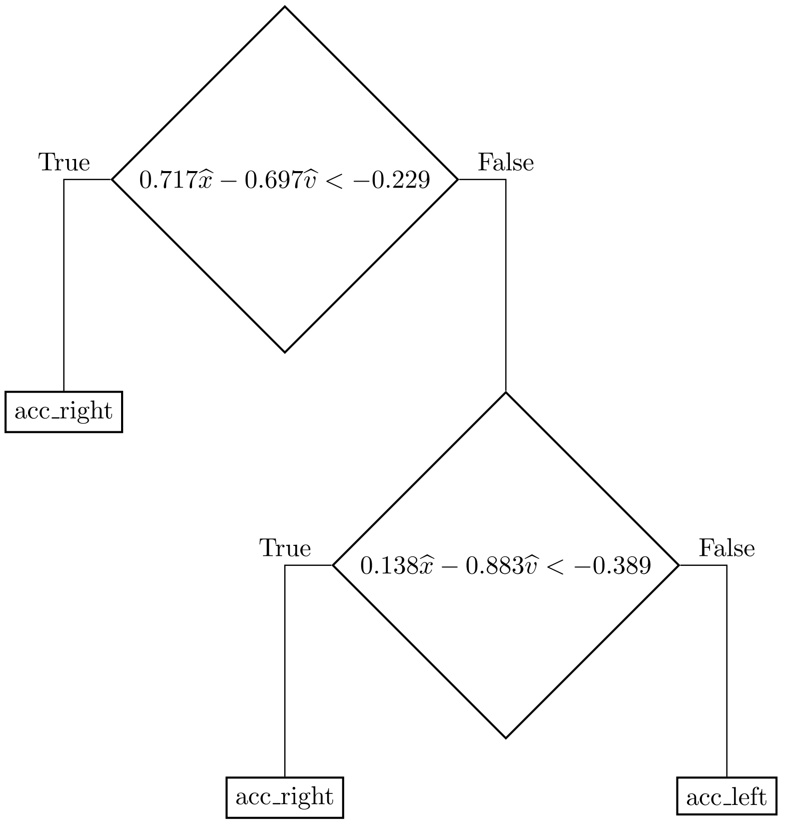
True
$0.717\widehat{x} - 0.697\widehat{v} < -0.229$
False
acc_right
True
$0.138\widehat{x} - 0.883\widehat{v} < -0.389$
False
acc_right
acc_left

Figure 18: Best oblique decision tree (w.r.t. score) evolved in the MountainCar-v0 environment.

### *4.5. LunarLander*

#### *4.5.1. Experimental setup*

In this case, we were not able to find a configuration that gave satisfying results with orthogonal trees. For this reason, in this case we will show only the results obtained by using an oblique grammar.

The grammar used for this task is shown in Table 19, while the parameters used for the grammatical evolution and the Q-learning are shown in Tables 20 and 21, respectively.

In this case, as shown in Table 21, we significantly increased the number of episodes used for the training. This is due to the following reasons:

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

Moreover, as shown in Table 21, we used a slightly different $\mathcal{Q}$-learning approach for this environment. In fact, in this case, we are using a decay for the $\varepsilon$ parameter, in order to explore better 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. The learning rate has been set to $\frac{1}{k}$, where k is the number of visits to the state-action pair. This guarantees that the state-action function converges to the optimum with $k \to \infty$. Finally, to save computation time, we implemented an early stopping criterion such that if the mean score over the current period is smaller than the one obtained in the previous period, then the training is stopped. This is based on the following assumption: if the current mean score is worse than the previous one, then we can assume that the state-action function is converging to its optimum, so the small oscillations due to the randomness made it worse than the previous mean.

#### *4.5.2. Results*

The results obtained in this environment are summarized in Table 22 and plotted in Figure 19. We can easily observe that there is a local Pareto front

| Rule | Production |
|---|---|
| dt | $< if >$ |
| if | $if\ < condition >\ then\ < action >\ else\ < action >$ |
| condition | $lt((\sum_{i=1}^{n\_variables} < const > input_i), 0)$ |
| action | $leaf \mid < if >$ |
| *const* | $[-1, 1]$ with step $10^{-3}$ |

Table 19: Grammar used to evolve oblique decision trees in the LunarLander-v2. The symbol "|" denotes the possibility to choose between different symbols. "lt" refers to the "less than" operator.

| Parameter | Value |
|---|---|
| Population size | 100 |
| Generations | 100 |
| Genotype length | 100 |
| Crossover probability | 0.1 |
| Crossover operator | One-point crossover |
| Selection operator | Tournament selection with size 2 |
| Mutation probability | 1 |
| Mutation type | Uniform, with gene probability=0.05 |

Table 20: Parameters used for the Grammatical Evolution with oblique grammar in the LunarLander-v2 environment.

| Parameter | Value |
|---|---|
| Algorithm | $\varepsilon$-greedy Q-learning with $\varepsilon$-decay |
| $\varepsilon_0$ | 1 |
| Decay multiplier | 0.99 |
| Initialization strategy | Constant, with value 0 |
| Learning rate | $1/k$, k is the number of visits of the action |
| Number of episodes | 1000 with early stopping |
| Early stopping period | 30 episodes |

Table 21: Parameters used for the Q-learning algorithm in the LunarLander-v2 environment.

between interpretability and performance composed of the solutions obtained in Runs 1 and 6.

| **Run** | **Training score** | **Testing mean** | **Testing std** | $\mathcal{M}$ |
|---|---|---|---|---|
| R1 | 265.77 | **262.18** | 29.32 | 86.90 |
| R2 | 256.07 | **252.40** | 21.80 | 146.70 |
| R3 | 251.95 | **240.50** | 37.21 | 145.30 |
| R4 | 248.19 | **234.45** | 79.07 | 117.50 |
| R5 | 220.59 | **206.48** | 64.74 | 87.60 |
| R6 | 274.77 | **272.14** | 28.31 | 118.90 |
| R7 | 254.33 | **230.65** | 78.29 | 207.90 |
| R8 | 256.09 | **251.21** | 32.81 | 87.60 |
| R9 | 265.21 | **257.75** | 31.45 | 147.40 |
| R10 | 262.86 | **252.70** | 43.11 | 87.60 |

Table 22: Summary of the best interpretable agents obtained for each run on the LunarLander-v2 environment. The task is considered solved when the mean score on 100 independent runs is greater or equal to 200.

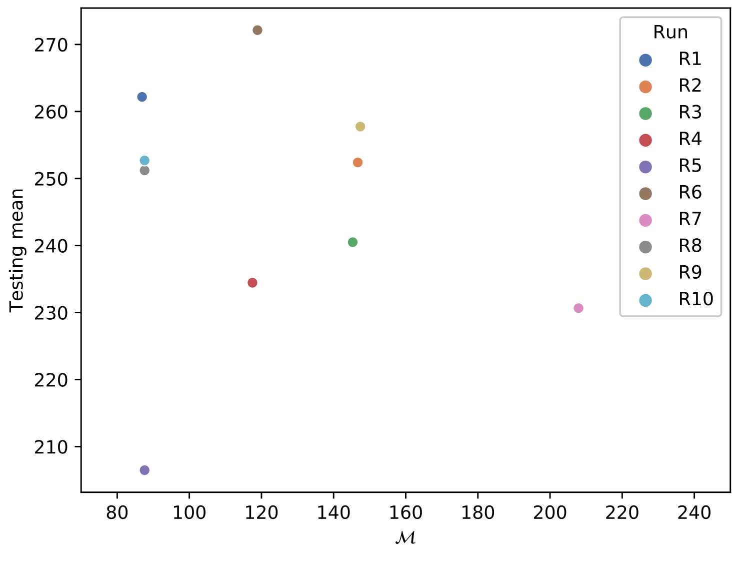

Figure 19: Score-$\mathcal{M}$ plot of the solutions obtained for the LunarLander-v2 environment.

In this case, our approach is able to solve the task in the 100% of the cases. Figure 20 shows the average fitness trend for the best solution in each run.

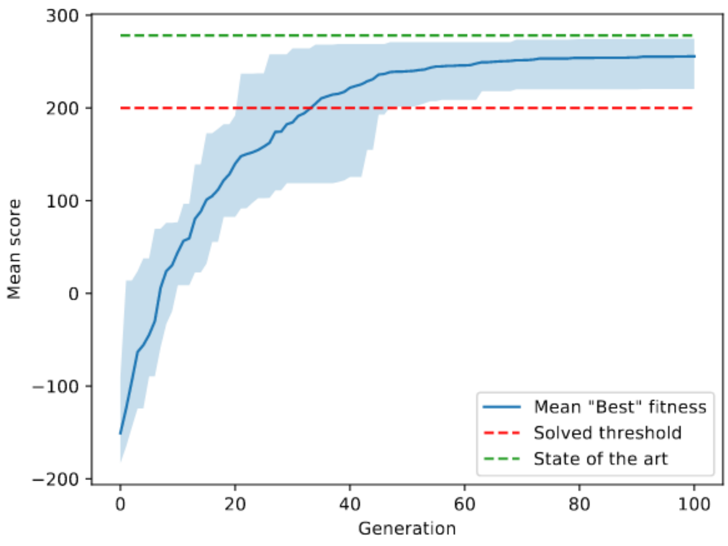

Figure 20: Mean fitness trend for the best individual in the population on the LunarLander-v2 environment.

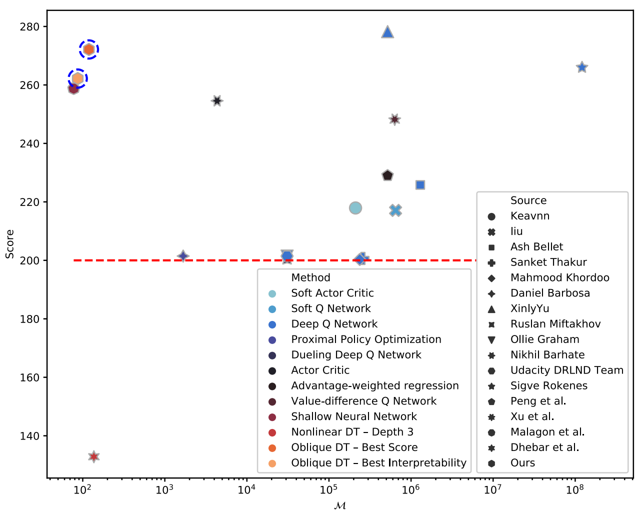

Figure 21: Plot of the score and $\mathcal{M}$ of our solutions compared to the state-of-the-art. Our solutions are the ones with a dashed circle around the symbol. The red line represents the "solved" threshold, fixed at 200 for the LunarLander-v2 task.

A comparison of our two best solutions (w.r.t score and interpretability) and the state-of-the-art is shown in Table 23 and Figure 21. As we can observe, even though we do not achieve (in absolute) the best performance and the best $\mathcal{M}$, our solutions represent the best compromise between the two metrics. Moreover, in Figure 21 we can observe that a Pareto front that explains the trade-off between interpretability and performance seems to exist. However, our best solution achieves a comparable performance w.r.t. the best score of the state-of-the-art, while having a substantially smaller complexity. Our best solution is shown in Figure 22

| **Source** | **Method** | **Score** | $\mathcal{M}$ |
|---|---|---|---|
| Keavnn[7] | Soft Actor Critic | 217.92 | 210733.2 |
| liu[8] | Soft Q Network | 217.09 | 647691.1 |
| Ash Bellet[9] | Deep Q Network | 225.79 | 1295307.1 |
| Sanket Thakur[10] | Deep Q Network | 200.65 | 259285.8 |
| Mahmood[11] | Deep Q Network | 200.3 | 237079.7 |
| Daniel Barbosa[12] | Proximal Policy Opt. | 201.47 | 1673 |
| XinlyYu[13] | Deep Q Network | **278.23** | 518153 |
| Ruslan[14] | Dueling Deep Q N. | 200.22 | 30878.1 |
| Ollie Graham[15] | Deep Q Network | 201.46 | 30878.1 |
| Nikhil Barhate[16] | Actor Critic | 254.58 | 4337.3 |
| Udacity[17] | Deep Q Network | 201.46 | 30878.1 |
| Sigve Rokenes[18] | Deep Q Network | 266 | $1.21 \cdot 10^8$ |
| Peng et al.[27] | Advantage-weighting | $229 \pm 2$ | 518153 |
| Xu et al.[28] | Value-difference | $248.2 \pm 21$ | 632620.2 |
| Malagon et al.[29] | Shallow NN | 258.8 | **77.6** |
| Silva et al.[3] | Rule List | -78.4 | 89 |
| 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 |
| Ours – Mean | Oblique DT | $246.05 \pm 18.72$ | $123.34 \pm 39$ |

Table 23: Comparison of the proposed solution with respect to the state-of-the-art on the LunarLander-v2 environment. The results from [27] are averaged on 5 runs. The results from [28] and [29] are averaged on 10 runs.

[7]github.com/StepNeverStop/RLs, accessed: 11 dec 2020.
[8]github.com/createamind/DRL, accessed: 11 dec 2020.
[9]github.com/nextgrid/deep-learning-labs-openAI, accessed: 11 dec 2020.
[10]github.com/sanketsans/openAIenv, accessed: 11 dec 2020.
[11]github.com/cpow-89/Extended-Deep-Q-Learning-For-Open-AI-Gym-Environments, accessed: 11 dec 2020.
[12]github.com/danielnbarbosa/angela, accessed: 11 dec 2020.
[13]github.com/XinliYu/Reinforcement_Learning-Projects, accessed: 11 dec 2020.
[14]github.com/RMiftakhov/LunarLander-v2-drlnd, accessed: 11 dec 2020.
[15]github.com/Cozmo25/openai-lunar-lander-v2, accessed: 11 dec 2020.
[16]github.com/nikhilbarhate99/Actor-Critic-PyTorch, accessed: 11 dec 2020.
[17]github.com/udacity/deep-reinforcement-learning, accessed: 11 dec 2020.
[18]evgiz.net/article/2019/02/02/, accessed: 11 dec 2020.

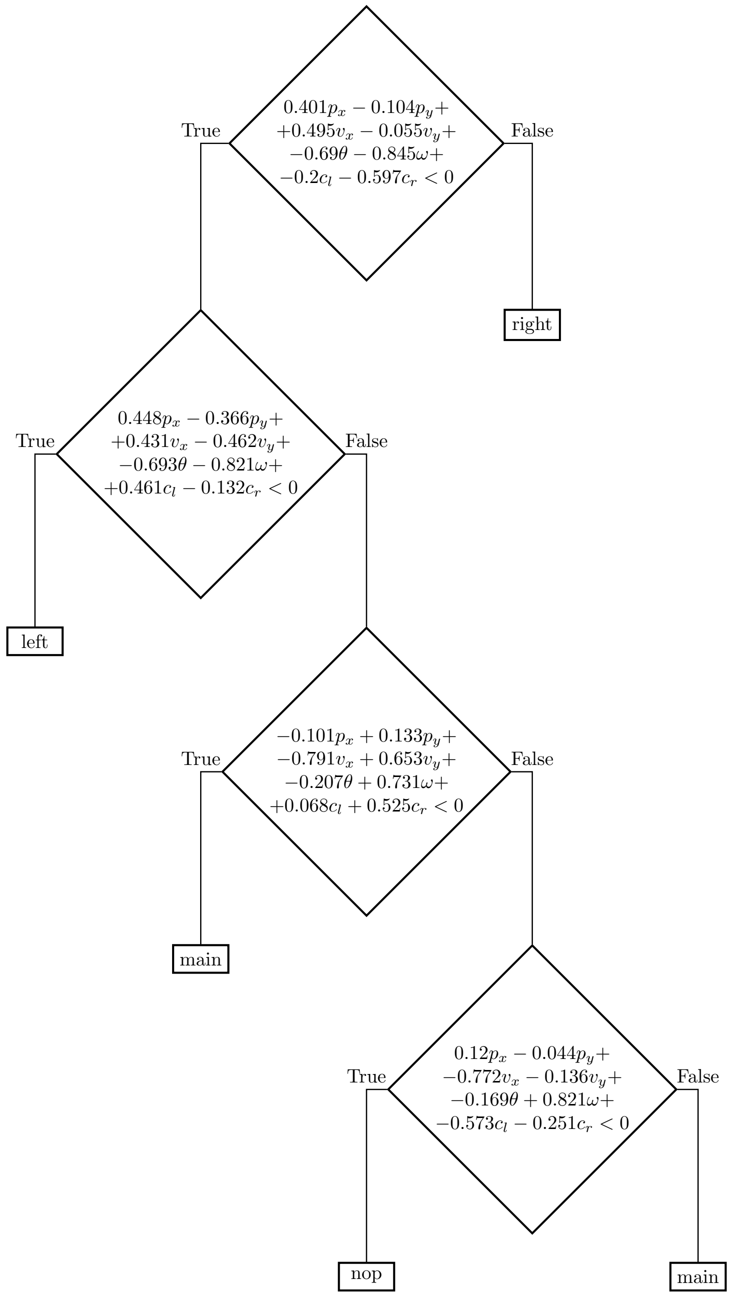

Figure 22: Best oblique decision tree (w.r.t. score) evolved in the LunarLander-v2 environment.

## 5. Discussion

In this section we briefly describe the interpretable techniques proposed in literature and we discuss our results in comparison to them. Then, we will perform an ablation study and, finally, we will interpret the produced trees.

### *5.1. CartPole*

#### *5.1.1. Differentiable Decision Trees*

Silva et al. [3] propose an approach based on differentiable decision trees, i.e. decision trees that replace hard splits with sigmoids. This means that they refactor the conditions from $variable > constant$ to $\sigma(variable - constant)$. By replacing hard splits with sigmoids, the decision of the tree can be seen as the sum of all the leaves weighted by the product of the outputs of the sigmoids for that path (i.e. the product of all the $\sigma(variable - threshold)$ for the true branch and $(1 - \sigma(variable - threshold))$ for the false branch for each split encountered). They optimize the splits of the tree and the actions taken by using PPO [30] and backpropagation. The solution proposed for the CartPole-v1 environment is the decision tree shown in Figure 23.

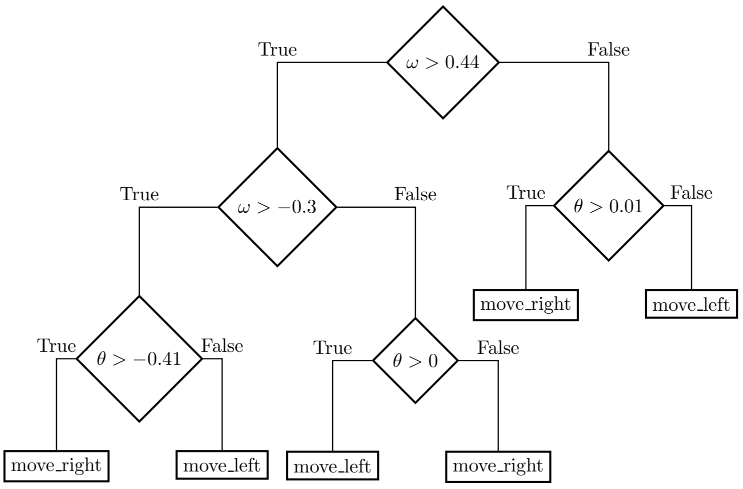

Figure 23: Tree representation of the solution proposed in [3].

It is interesting to observe that their optimization process “selects” the same variables that have been selected in our case by artificial evolution.

Moreover, the tree proposed by them is slightly more complex than our best tree. In fact, while our best tree has a maximum depth of 2 conditions, their has a maximum depth of 3 conditions. This increase in complexity is reflected by the difference in the $\mathcal{M}$ measure.

Moreover, since the performance of this solution are not satisfactory, the authors gave us the solution coming from a follow-up work in a personal communication. This solution is shown in Figure 24.

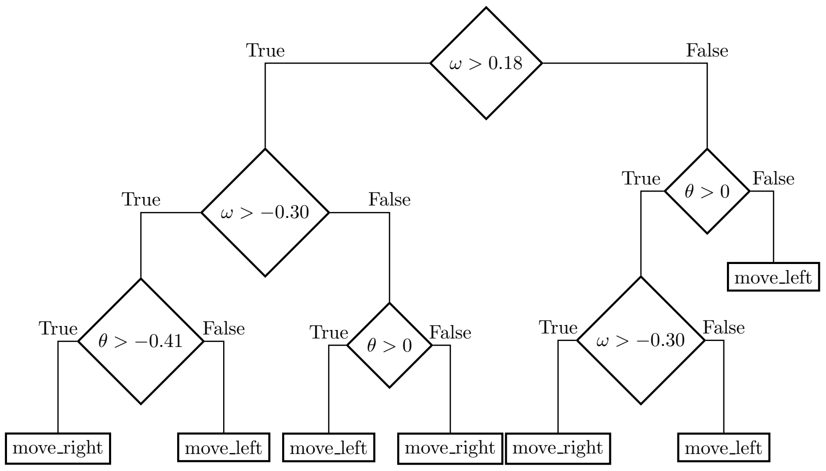

Figure 24: Tree representation of the solution obtained by private communication with the first author of [3]

Besides the performance comparison performed in Table 9, we compare here the robustness to noise, similarly to what we did in Figure 9.

As we can see in Figure 25, the orthogonal trees obtained by Silva et al. have a robustness that is comparable to our orthogonal tree. This suggests us that orthogonal trees may be intrinsically less robust than oblique ones.

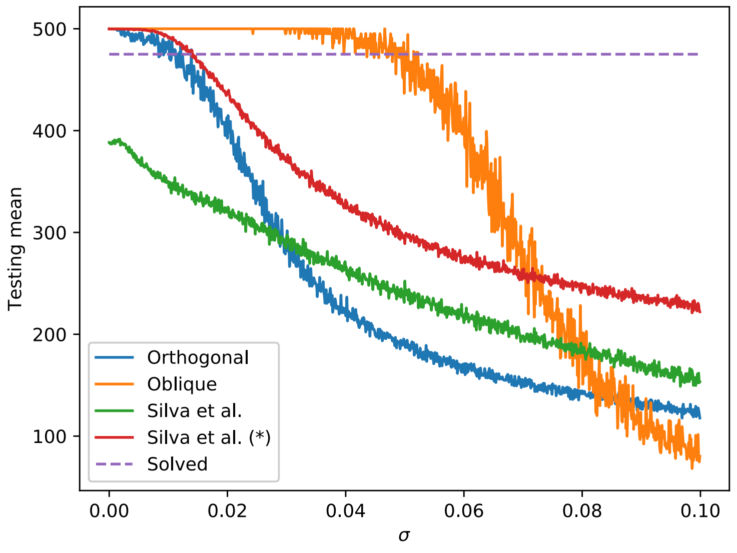

Figure 25: Robustness to input noise on the CartPole-v1 environment. (*): Private communication with the first author.

## *5.2. MountainCar*

Most of the approaches we used for the comparison in the MountainCar-v0 environments come from the OpenAI Gym leaderboard.

### *5.2.1. Zhiqing Xiao*

The system proposed by this entry[19] consists in a closed-form policy. However, it is not clear whether the policy has been derived by a human or learned by a machine.

Anyway, this solution achieves the best performance (let alone our solutions) on this task while also having the best degree of interpretability (according to our modification of the metric proposed in [20]). The policy is the following:

$$a = min(-0.09(x+0.25)^2 + 0.03, 0.3(x+0.9)^4 - 0.008)$$

[19]github.com/ZhiqingXiao/OpenAIGymSolution/tree/master/MountainCar-v0_close_form

$$b = -0.07(x+0.38)^2 + 0.07$$

$$\pi(x,v) = \begin{cases} acc\_right & \text{if } a < v < b \\ acc\_left & \text{else} \end{cases}$$

While $\mathcal{M}$ is lower for this policy than for our best tree, it may be a bit harder to interpret this model. We think that this is due to the fact that the $\mathcal{M}$ metric has been proposed to evaluate the interpretability of mathematical formulae, while we are interested in interpreting *hyperplanes*. While hyperplanes are defined by mathematical formulae, the interpretability of an hyperplane may also depend on the number on non-linear operations that are used to determine the hyperplane.

### *5.2.2. Amit*

This entry[20] uses SARSA to solve the task.

While tabular approaches like SARSA and Q-learning are transparent, their interpretability depends heavily on the number of states and actions. Table 18 shows that, even if this approach is transparent and easily interpretable, our solutions are able to achieve a better degree of interpretability. In our opinion, this is due to the fact that using decision trees as function approximators leads to the "grouping" of some states of the table used in tabular approaches. This is especially useful when we want to *extract* knowledge. In fact, by grouping some states, we take into account only the variables and the thresholds that have a big impact on the policy, discarding irrelevant details.

### *5.2.3. Dhebar et al.*

Dhebar et al., in [4], propose an approach to reinforcement learning that uses nonlinear decision trees. They first approximate an oracle policy and then they fine-tune it by using evolutionary algorithms. The policies obtained in these two phases are called "open-loop" and "closed-loop" policies.

In this case, we only had access to the open-loop policy for the MountainCar-v0 environment, which is shown in Figure 26.

[20]github.com/amitkvikram/rl-agent/blob/master/mountainCar-v0-sarsa.ipynb

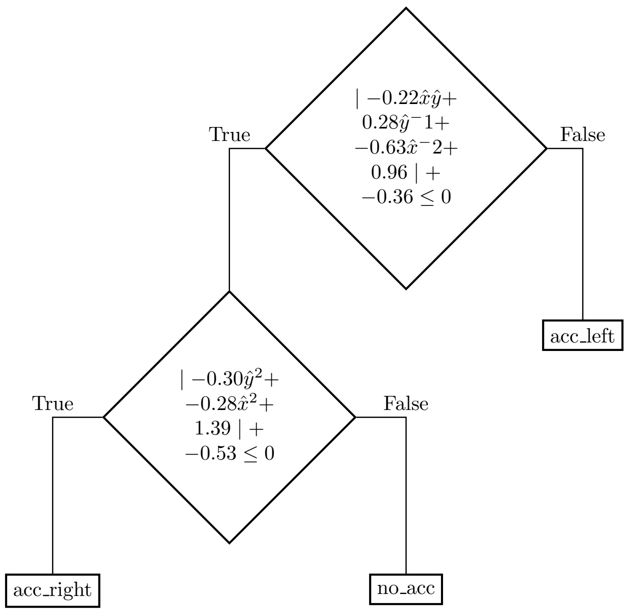

Figure 26: Tree representation of the solution proposed in [4] for the MountainCar-v0 environment. The variables with a hat are normalized by using this way: $1 + \frac{x - x_{min}}{x_{max} - x_{min}}$.

Also in this case, while $\mathcal{M}$ for this solution is better than our best solution (w.r.t. test score), it seems harder to interpret, due to the non-linearity of the hyperplanes. In fact, in our solution $\mathcal{M}$ is higher due to the higher number of splits in the tree, but that does not take into account the fact that in our case the hyperplanes that divide the feature space are simpler than the ones proposed in [4].

Finally, we perform a comparison on the robustness to input noise with the solutions provided by "Zhiqing Xiao" and the one provided by Dhebar et al. Figure 27 shows how performance vary by varying the standard deviation of the additive Gaussian noise. We observe that there is no significant difference between the solutions, meaning that all of them have high sensitivity to input noise.

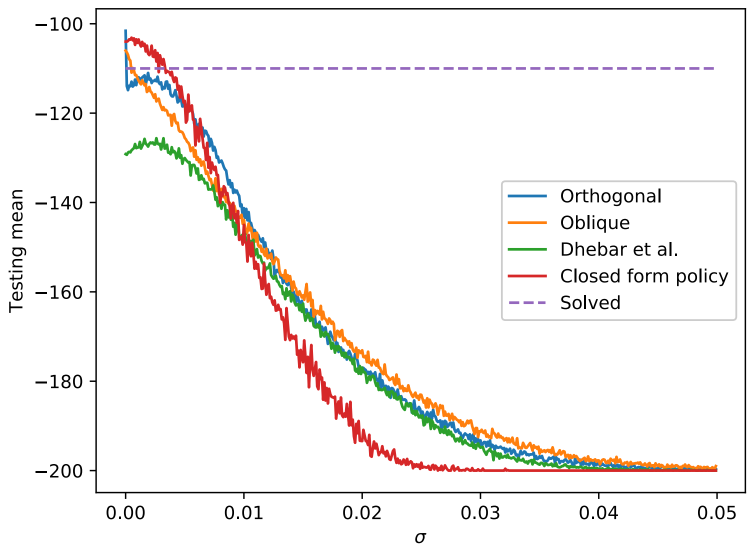

Figure 27: Comparison of the robustness to input noise between our solutions and the interpretable ones on the MountainCar-v0 environment.

### *5.3. LunarLander-v2*

#### *5.3.1. Silva et al.*

In [3] the authors, besides regular trees, use also decision lists. A decision list is a tree that is extremely unbalanced, i.e. il collapses to a list.

Figure 28 shows the solution obtained. However, as shown in Table 23, it does not achieve satisfactory performance. This is due to the fact that, while the differentiable tree is able to achieve better performance (even though it does not solve the task), its discretization modifies the final distribution of the actions.

#### *5.3.2. Malagon et al.*

In [29] the authors use the Univariate Marginal Distribution Algorithm to evolve a neural network without hidden layers in the LunarLander-v2 domain.

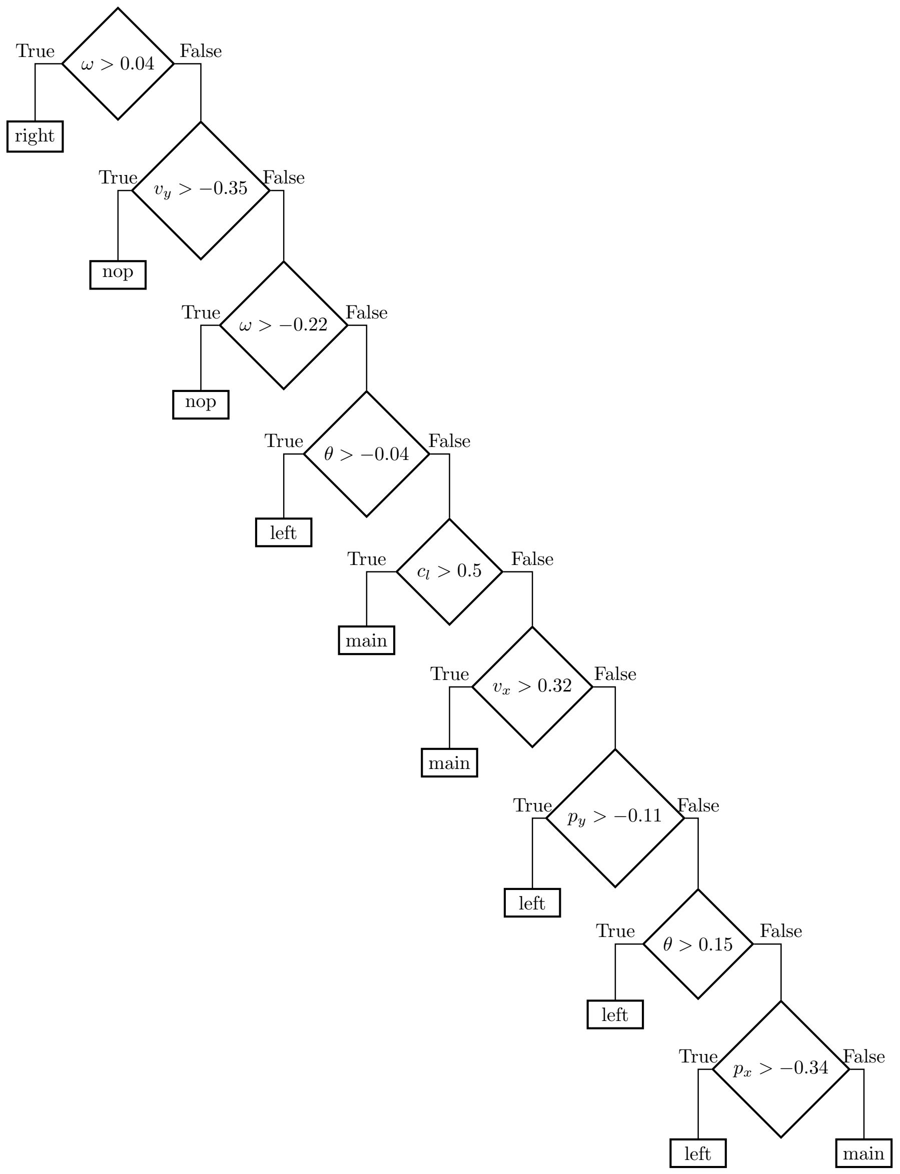

Figure 28: Tree representation of the solution proposed in [3] for the LunarLander-v2 environment.

Since the neural network has no hidden layers the whole system reduces to

$$a = \underset{i}{argmax}(\sigma(\mathbf{w_i}^T \cdot \mathbf{x} + b_i))$$

where $i$ refers to the output neurons.

This results in an easy-to-interpret system, according to both [21] and the metric $\mathcal{M}$.

### *5.3.3. Dhebar et al.*

In [4] the authors propose a nonlinear decision tree that achieves a mean testing score of 234.98 points. However, the rules associated with this tree are not shown, so we only had access to the 3-levels-deep NLDT.

The tree is shown in Figure 29. It is important to note that even if the solution obtained is a tree, the interpretation is not easy, since the hyperplanes contained in each split are not linear.

Also in this case, we performed a comparison on the robustness to input noise, the result is shown in Figure 30. However, for this comparison we could not include the results from Malagon et al. since the weights were not publicly accessible.

## *5.4. Ablation study*

In order to assess whether our two-level optimization approach is convenient with respect to a single-level optimization approach, we perform an ablation study in which we use Grammatical Evolution alone to evolve decision trees. Moreover, we perform statistical tests to test whether the difference are statistically significant by fixing a threshold for the p-value of $\alpha = 0.05$.

### *5.4.1. CartPole*

*Orthogonal trees.* To evolve orthogonal trees, we used the grammar shown in Table 24, which has been evolved by using the same parameters described in Table 3. Also in this case, the fitness was computed as the mean score on 10 episodes.

The results are shown in Table 25. As we can observe, while in most cases the evolution is able to evolve agents that achieve a perfect training score, they

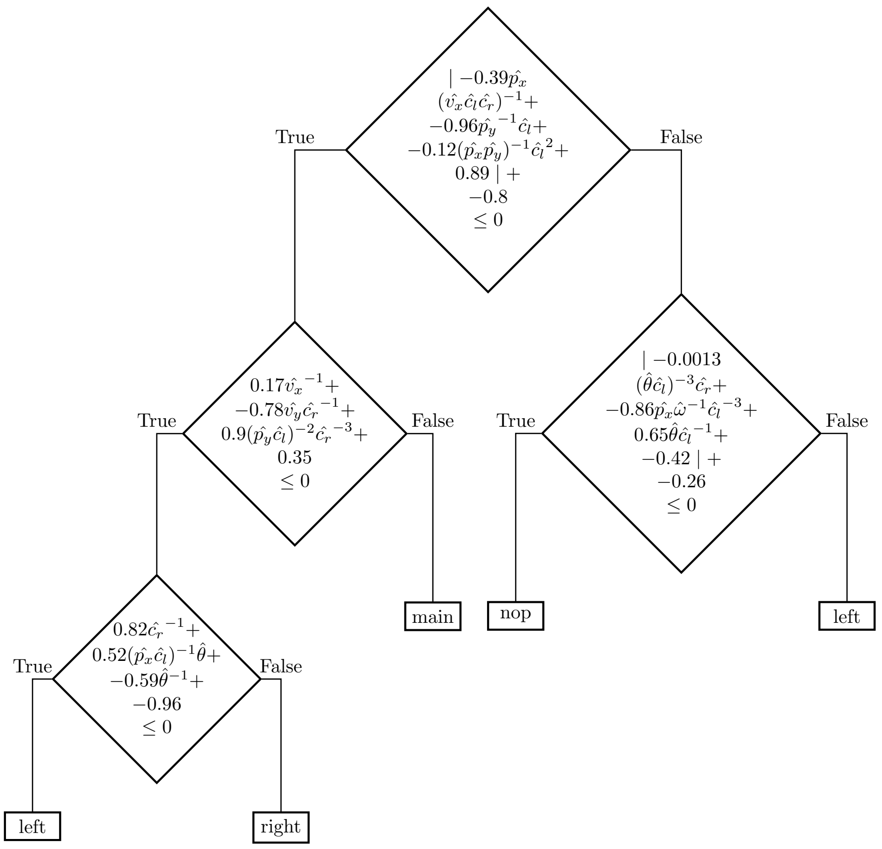
$| -0.39\hat{p_x}$
$(\hat{v_x}\hat{c_l}\hat{c_r})^{-1}+$
$-0.96\hat{p_y}^{-1}\hat{c_l}+$
$-0.12(\hat{p_x}\hat{p_y})^{-1}\hat{c_l}^2+$
$0.89 | +$
$-0.8$
$\leq 0$
True
False
$0.17\hat{v_x}^{-1}+$
$-0.78\hat{v_y}\hat{c_r}^{-1}+$
$0.9(\hat{p_y}\hat{c_l})^{-2}\hat{c_r}^{-3}+$
$0.35$
$\leq 0$
True
False
$| -0.0013$
$(\hat{\theta}\hat{c_l})^{-3}\hat{c_r}+$
$-0.86\hat{p_x}\hat{\omega}^{-1}\hat{c_l}^{-3}+$
$0.65\hat{\theta}\hat{c_l}^{-1}+$
$-0.42 | +$
$-0.26$
$\leq 0$
True
False
main
nop
left
$0.82\hat{c_r}^{-1}+$
$0.52(\hat{p_x}\hat{c_l})^{-1}\hat{\theta}+$
$-0.59\hat{\theta}^{-1}+$
$-0.96$
$\leq 0$
True
False
left
right

Figure 29: Tree representation of the solution proposed in [4] for the LunarLander-v2 environment. The variables with a hat are normalized by using this way: $1 + \frac{x - x_{min}}{x_{max} - x_{min}}$.

have poor generalization capabilities. In our opinion, this is akin to overfitting. In fact, in this case, the agents did not understand the "value" of going in a certain state, but just learned a rule that worked in the tested cases. Moreover, a two-tailed Mann-Whitney U-Test gives us a p-value of $9 \cdot 10^{-3}$ that allows us to reject the null hypothesis (i.e. that the mean testing score come from the same distribution) with threshold $\alpha = 0.05$.

*Oblique trees.* We perform the same test also in the oblique setting. We use the grammar shown in Table 26 and the parameters used in Table 4.

The results are shown in Table 27. We can easily observe that in this case

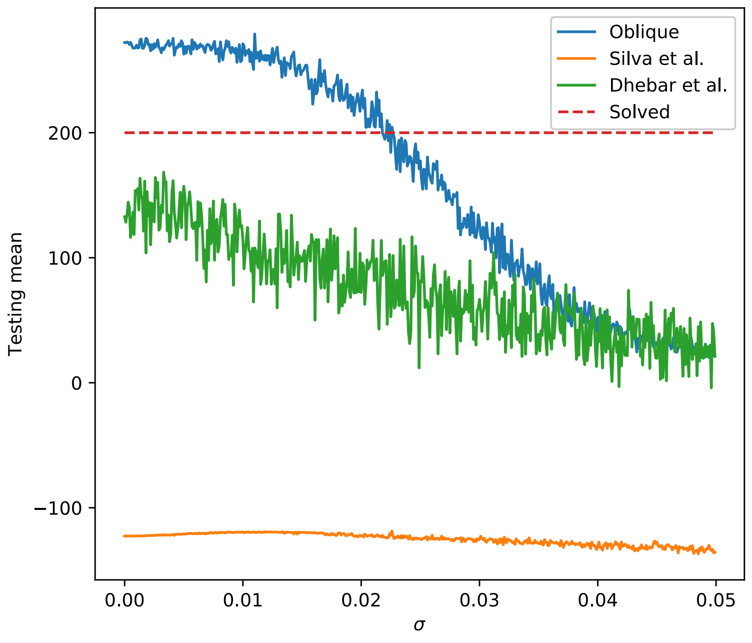

Figure 30: Comparison on the robustness to input noise w.r.t. other interpretable solutions on the LunarLander-v2 environment.

the results are similar to the ones shown in Table 6. To check whether this similarity has a statistical significance, we perform a Two-tailed Mann-Whitney U-Test. The null hypothesis states that the results obtained by using the GE with Q-learning and GE alone come from the same statistical distribution. The p-value obtained with this test is 0.73, so we are not able to reject the null hypothesis with threshold $\alpha = 0.05$. For this reason, we will assume that they come from the same distribution.

This suggests us that, since oblique trees seem to be both more robust to noise and more stable than orthogonal trees, an agent can learn good policies in *simple* environments without the need for $\mathcal{Q}$-learning.

| **Rule** | **Production** |
|---|---|
| dt | $< if >$ |
| if | $if \ < condition > \ then \ < action > \ else \ < action >$ |
| condition | $input\_var \ < comp\_op > < const_{input\_var} >$ |
| action | $< output > \ \mid \ < if >$ |
| output | $move\_left \mid move\_right$ |
| comp_op | $lt \mid gt$ |
| $const_x$ | [-4.8, 4.8) with step 0.5 |
| $const_v$ | [-5, 5) with step 0.5 |
| $const_\theta$ | [-0.418, 0.418) with step 0.01 |
| $const_\omega$ | [-0.836, 0.836) with step 0.01 |

Table 24: Grammar used to evolve orthogonal decision trees in the CartPole-v1 environment without Q-learning.

| **Run** | **Training score** | **Testing mean** | **Testing std** | $\mathcal{M}$ |
|---|---|---|---|---|
| R1 | 500.00 | **500.00** | 0 | 53.40 |
| R2 | 500.00 | 436.39 | 120.91 | 71.20 |
| R3 | 500.00 | 473.32 | 80.42 | 35.60 |
| R4 | 500.00 | **498.3** | 11.21 | 53.40 |
| R5 | 500.00 | 418.30 | 136.11 | 35.60 |
| R6 | 500.00 | **489.6** | 23.46 | 35.60 |
| R7 | 500.00 | **486.63** | 45.32 | 35.60 |
| R8 | 500.00 | 468.20 | 82.34 | 35.60 |
| R9 | 500.00 | 455.57 | 126.74 | 71.20 |
| R10 | 500.00 | **483.11** | 41.99 | 71.20 |

Table 25: Results obtained by evolving orthogonal decision trees for the CartPole-v1 environment by using Grammatical Evolution alone.

### *5.4.2. MountainCar*

*Orthogonal trees.* We evolve orthogonal trees for the MountainCar-v0 environment by using the grammar shown in Table 28 and the parameters shown in Table 10. Since in this case the number of episodes is low and the environment is harder to explore than CartPole, we expect GE alone to perform comparably with our approach.

The results are shown in Table 29. As we expected, the performance are quite similar. To ensure that there are no statistical significant differences between the two approaches, we performed a Two-tailed Mann-Whitney U-Test on the testing mean score obtained by using the two approaches, which stated that the null hypothesis (i.e. the scores obtained come from the same distribution)

| Rule | Production |
|---|---|
| dt | $< if >$ |
| if | $if \ < condition > \ then \ < action > \ else \ < action >$ |
| condition | $lt((\sum_{i=0}^{n\_variables} < const > *input_i), < const >)$ |
| action | $< output > \ \mid \ < if >$ |
| output | $move\_left \mid move\_right$ |
| *const* | $[-1, 1]$ with step $10^{-3}$ |

Table 26: Grammar used to evolve oblique decision trees in the CartPole-v1 environment without Q-learning.

| Run | Training score | Testing mean | Testing std | $\mathcal{M}$ |
|---|---|---|---|---|
| R1 | 500.00 | **500.00** | 0.00 | 24.10 |
| R2 | 500.00 | **500.00** | 0.00 | 24.10 |
| R3 | 500.00 | **500.00** | 0.00 | 24.10 |
| R4 | 500.00 | **500.00** | 0.00 | 24.10 |
| R5 | 500.00 | **477.29** | 71.38 | 48.20 |
| R6 | 500.00 | **500.00** | 0.00 | 24.10 |
| R7 | 500.00 | **500.00** | 0.00 | 24.10 |
| R8 | 500.00 | **500.00** | 0.00 | 24.10 |
| R9 | 500.00 | **500.00** | 0.00 | 46.80 |
| R10 | 500.00 | **500.00** | 0.00 | 24.10 |

Table 27: Results obtained by evolving oblique decision trees for the CartPole-v1 environment by using Grammatical Evolution alone.

cannot be rejected with threshold $\alpha = 0.05$.

*Oblique trees.* We perform the test also by using oblique trees. We use the grammar described in Table 30 with the parameters shown in Table 13.

The results are shown in Table 31. While these results seem to be better than the ones shown in 17, they do not seem to be statistically significant according to a two-tailed Mann-Whitney U-Test (p-value 0.38). Thus, this seems to confirm our hypothesis that states that solving MountainCar-v0 with oblique trees seems to be harder than the case with orthogonal trees (with the proposed grammar).

### *5.4.3. LunarLander*

Finally, we perform the same test on LunarLander-v2, using only oblique trees. We use the grammar described in Table 32 and the parameters present in Table 20. We expect that in this task, since it is harder than the previous

| Rule | Production |
|---|---|
| dt | $< if >$ |
| if | $if\ < condition >\ then\ < action >\ else\ < action >$ |
| condition | $input\_var\ < comp\_op > < const_{input\_var} >$ |
| action | $< output >\ \mid\ < if >$ |
| output | $acc\_left \mid no\_acc \mid acc\_right$ |
| comp_op | $lt \mid gt$ |
| $const_x$ | [-1.2, 0.6) with step 0.05 |
| $const_v$ | [-0.07, 0.07) with step 0.005 |

Table 28: Grammar used to evolve orthogonal decision trees in the CartPole-v1 environment without Q-learning.

| Run | Training score | Testing mean | Testing std | $\mathcal{M}$ |
|---|---|---|---|---|
| R1 | -104.10 | **-107.44** | 16.02 | 106.8 |
| R2 | -115.60 | -115.60 | 1.31 | 35.60 |
| R3 | -119.40 | -119.34 | 3.66 | 17.80 |
| R4 | -118.70 | -125.86 | 28.68 | 71.20 |
| R5 | -119.40 | -119.34 | 3.66 | 35.60 |
| R6 | -103.00 | **-106.02** | 15.21 | 106.80 |
| R7 | -103.20 | **-108.31** | 18.53 | 124.60 |
| R8 | -103.00 | **-105.98** | 15.03 | 89.00 |
| R9 | -105.40 | **-104.71** | 3.66 | 106.80 |
| R10 | -101.80 | -114.09 | 30.81 | 89.00 |

Table 29: Results obtained by evolving orthogonal decision trees for the MountainCar-v0 environment by using Grammatical Evolution alone.

two, GE performs worse than our approach.

The results of this experiment are shown in Table 33. According to our expectations, we are able to solve the task only in the 60% of the cases. Moreover, we perform a two-tailed Mann-Whitney U-Test to test the statistical significance of the differences between the two approaches (on the mean testing score). We obtain a p-value of 0.017 that allows us to reject the null hypothesis. We can thus hypothesize that the use of the two-level optimization technique gives us a boost in performance in complex environments such as LunarLander-v2.

## *5.5. Interpretation of the solutions*

In this subsection, we will look at the agents produced and try to interpret the policies.

| **Rule** | **Production** |
|---|---|
| dt | $< if >$ |
| if | $if\ < condition >\ then\ < action >\ else\ < action >$ |
| condition | $lt((\sum_{i=0}^{n\_variables} < const > * input_i), < const >)$ |
| action | $< output >\ \mid\ < if >$ |
| output | $acc\_left \mid no\_acc \mid move\_right$ |
| *const* | $[-1, 1]$ with step $10^{-3}$ |

Table 30: Grammar used to evolve oblique decision trees in the CartPole-v1 environment without Q-learning.

| **Run** | **Training score** | **Testing mean** | **Testing std** | $\mathcal{M}$ |
|---|---|---|---|---|
| R1 | -102.00 | **-105.83** | 16.49 | 139.80 |
| R2 | -97.10 | **-106.8** | 23.61 | 93.40 |
| R3 | -102.00 | **-107.74** | 20.33 | 70.20 |
| R4 | -101.40 | -111.36 | 23.98 | 70.00 |
| R5 | -101.80 | **-109.71** | 23.12 | 93.40 |
| R6 | -101.90 | **-108.79** | 21.58 | 116.80 |
| R7 | -101.30 | -110.56 | 25.63 | 93.40 |
| R8 | -97.20 | **-108.09** | 30.34 | 116.60 |
| R9 | -101.80 | **-107.48** | 20.00 | 93.20 |
| R10 | -105.80 | **-107.14** | 14.66 | 93.20 |

Table 31: Results obtained by evolving oblique decision trees for the MountainCar-v0 environment by using Grammatical Evolution alone.

### *5.5.1. CartPole*

*Orthogonal tree.* The tree shown in Figure 11 is extremely easy to interpret. In fact, this agent moves the cart to the left if

$$\omega < 0.074 \wedge \theta < 0.022 \tag{1}$$

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.

*Oblique tree.* In this case, the interpretation of the policy is a bit harder. The condition used by the agent to discriminate between the two states is:

$$-0.274x_k - 0.543v_k - 0.904\theta_k - 0.559\omega_k < -0.169 \tag{2}$$

| **Rule** | **Production** |
|---|---|
| dt | $< if >$ |
| if | $if \ < condition > \ then \ < action > \ else \ < action >$ |
| condition | $lt((\sum_{i=0}^{n\_variables} < const > *input_i), 0)$ |
| action | $< output > \ \mid \ < if >$ |
| output | $nop \mid left\_engine \mid main\_engine \mid right\_engine$ |
| $const$ | $[-1, 1]$ with step $10^{-3}$ |

Table 32: Grammar used to evolve oblique decision trees in the LunarLander-v2 environment without Q-learning.

| **Run** | **Training score** | **Testing mean** | **Testing std** | $\mathcal{M}$ |
|---|---|---|---|---|
| R1 | -88.83 | -90.25 | 34.46 | 147.40 |
| R2 | 216.45 | 172.83 | 76.83 | 60.50 |
| R3 | 241.37 | **228.12** | 47.7 | 59.10 |
| R4 | 272.25 | **252.88** | 54.27 | 115.40 |
| R5 | 231.83 | **216.65** | 60.59 | 117.50 |
| R6 | 103.36 | 49.15 | 127.21 | 89.00 |
| R7 | 266.90 | **251.28** | 42.95 | 120.30 |
| R8 | 247.40 | **205.25** | 73.41 | 58.40 |
| R9 | 254.00 | **243.95** | 34.58 | 88.30 |
| R10 | 5.47 | -57.84 | 120.95 | 122.40 |

Table 33: Results obtained by evolving oblique decision trees for the LunarLander-v2 environment by using Grammatical Evolution alone.

where $k$ refers to the current timestep. To simplify the process, we write Equation 1 as the following:

$$-ax_k - bv_k - c\theta_k - d\omega_k < t \tag{3}$$

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:

$$x_n \approx -\frac{t}{a} \tag{4}$$

where $n$ is the index of the last timestep. For simplicity, let's assume that $x_n = -\frac{t}{a}$. This means that we can rewrite Equation 3 as follows:

$$-x_k - \frac{b}{a}v_k - \frac{c}{a}\theta_k - \frac{d}{a}\omega_k < \frac{t}{a} = -x_n \tag{5}$$

We can then perform other steps and obtain:

$$-x_k - b'v_k - c'\theta_k - d'\omega_k < -x_n \Rightarrow \tag{6}$$

$$-b'v_k - c'\theta_k - d'\omega_k < -x_n + x_k \Rightarrow \tag{7}$$

$$-b'v_k - c'\theta_k - d'\omega_k < -x_n + x_{n-1} - x_{n-1} + ... + x_k = \sum_{j=n}^{k+1} -x_j + x_{j-1} \tag{8}$$

Then, by noting that

$$\frac{x_k - x_{k-1}}{\tau} = v_k \tag{9}$$

we can rewrite Equation 8 as:

$$-b'v_k - c'\theta_k - d'\omega_k < -\sum_{j=k+1}^{n} v_j\tau \tag{10}$$

$$-c'\theta_k - d'\omega_k < -\sum_{j=k}^{n} g_j v_j \tau \tag{11}$$

where

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

Now, by observing that

$$\frac{\theta_k - \theta_{k-1}}{\tau} = \omega \tag{12}$$

we obtain

$$-c'\theta_k - d'\frac{\theta_k - \theta_{k-1}}{\tau} < -\sum_{j=k+1}^{n} g_j v_j \tau \tag{13}$$

$$-(d' + \tau c')\theta_k + d'\theta_{k-1} < -\tau^2 \sum_{j=k+1}^{n} g_j v_j \tag{14}$$

Finally, noting that after that usually, in the first 50 timesteps of the simulations the velocities are high ($\underset{k}{max} \mid v_k \mid < 1.5$) and then the velocity become small ($\underset{k}{max} \mid v_k \mid < 0.55$) because the pole is balanced, we can write that:

$$\mid \sum_{j=k+1}^{n} g_j v_j \mid \lessapprox \frac{b'}{\tau} \cdot 1.5 + 49 \cdot 1.5 + 450 \cdot 0.55 = 420 \tag{15}$$

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 $\tau^2$ ($\tau = 0.02$ 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 intuitively.

Then, we obtain

$$-(d' + \tau c')\theta_k + d'\theta_{k-1} < \tau b' v_k \tag{16}$$

$$c\theta_k > -(bv_k + d\omega_k) \tag{17}$$

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. linear velocity of the cart and 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.

### *5.5.2. MountainCar*

*Orthogonal tree.* Also in this case, the orthogonal tree (Figure 17) is easy to interpret. In fact, if we look at the leaves, we see that the agent accelerates to

[21] Implementing this policy by using the $\omega$ 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. On the other hand, using $\omega_k = (\theta_k - \theta_{k-1})/\tau$ gives the desired results.

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 agents accelerates to the left when:

- it is going towards the hill on the left 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 ($v < 0.035$) and it is near the valley

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

*Oblique trees.* In this case, the agent accelerates to the left when both conditions are false. This means that we have to solve the following system of two inequalities:

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

This means that the agent accelerates to the left when $v \leq 7.5799 \cdot 10^{-2} \cdot x + 6.6955 \wedge v \leq 1.1516 \cdot 10^{-2} \cdot x + 5.495 \cdot 10^{-3}$. This corresponds to the decision regions shown in Figure 31.

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

### *5.5.3. LunarLander*

In this case, since the oblique tree (Figure 22) has 4 conditions and 8 unknowns, it is a bit harder to interpret.

*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

$$ap_x - bp_y + cv_x - dv_y - e\theta - f\omega - gc_l - hc_r \geq 0 \tag{18}$$

where $a, b, ..., h$ replace the constants shown in Figure 22.

To simplify the analysis, let's 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

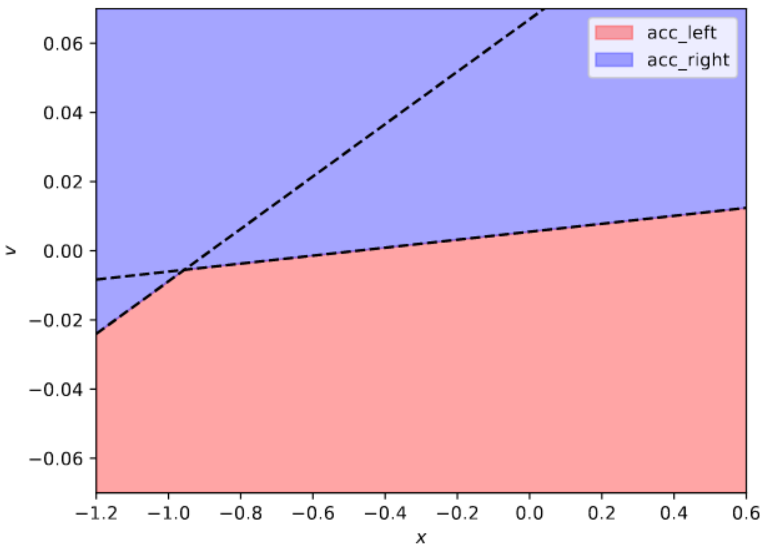

Figure 31: Decision regions for the best oblique tree evolved in the MountainCar-v0 environment.

simply say that, when contact with the ground happens, then the threshold is not 0 anymore, but it can take 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 18 as follows:

$$ap_x + cv_x - bp_y - dv_y - e\theta - f\omega \geq 0 \tag{19}$$

By merging some terms we obtain:

$$a(p_x + v_x c') - b(p_y - v_y d') - e(\theta - \omega f') \geq 0 \tag{20}$$

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 $\theta$ with $p_x^{k+1}$, $p_y^{k+1}$,

$\theta^{k+1}$ respectively, we can write:

$$ap_x^{k+1} - bp_y^{k+1} \geq e\theta^{k+1} \tag{21}$$

To understand how this condition works, let's 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), we will fire the right engine if $\theta^{k+1} \leq -b/e \approx -0.15rad$, 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 will fire 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's now suppose that $p_y^{k+1} = 0$ to study the effect of $p_x^{k+1}$ on the policy. Then, 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 on the right part of the environment, the agent uses a linear threshold to activate the engine in order 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 negative angles (that would move it farther on the left) and distance from the landing point.

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

$$ap_x - bp_y + cv_x - dv_y - e\theta - f\omega < 0 \tag{22}$$

Of course, in this case the coefficients $a$, ..., $f$ are different from the previous ones. By grouping the terms as before we obtain

$$a(p_x + v_x c') - b(p_y + v_y d') - e(\theta + \omega f') < 0 \tag{23}$$

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

$$ap_x^{k+1} - bp_y^{k+1} < e\theta^{k+1} \tag{24}$$

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

*Third condition.* This condition handles the firing of the main engine. For this reason, we expect it to work differently from the previous two. In fact, 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's use $a$, ..., $f$ to rename the constants and ignore $c_l$ and $c_r$. This leads to:

$$-ap_x + bp_y - cv_x + dv_y - e\theta + f\omega < 0 \tag{25}$$

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

$$-a(p_x + v_x) + b(p_y + v_y) - (c-a)v_x + (d-b)v_y - e\theta + f\omega < 0 \tag{26}$$

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:

$$-ap_x^{k+1} + bp_y^{k+1} - c'v_x + d'v_y - e\theta + f\frac{\theta - \theta^{k-1}}{\tau} < 0 \tag{27}$$

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

$$-\tau ap_x^{k+1} + \tau bp_y^{k+1} - \tau c'v_x + \tau d'v_y - \tau e\theta + f(\theta - \theta^{k-1}) < 0 \tag{28}$$

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:

$$-\tau c'v_x + \tau d'v_y + (f - \tau e)\theta - f\theta^{k-1} < 0 \tag{29}$$

By merging the terms in $\theta$ and $\theta^{k-1}$ we obtain:

$$-c'v_x + d'v_y + (f - \tau e)\omega + \tau e\theta^{k-1} < 0 \tag{30}$$

By moving all the terms except the one in $\omega$ to the second member we get:

$$\omega < \frac{1}{f - \tau e}(c' v_x - d' v_y - \tau e \theta^{k-1}) \tag{31}$$

Then, by noting that all the states that are tested in this condition have $c'\overline{\mid v_x \mid} \approx 5d'\overline{\mid v_y \mid}$ and $c'\overline{\mid v_x \mid} \approx 120e\overline{\mid \theta^{k-1} \mid}$ (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 $\theta^{k-1}$. Finally, the rule used to fire the main engine is:

$$\omega < c'' v_x \tag{32}$$

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 rule represents the landing phase. In fact, the goal of this rule is to balance angular velocity and linear velocity to make the agent gently stop on the landing pad.

*Fourth condition.* This condition, when evaluates to True, does not fire any engine. On the other hand, when it evaluates to False, it fires the main engine.

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

$$ap_x - bp_y - cv_x - dv_y - e\theta + f\omega < 0 \tag{33}$$

By analyzing the mean values of the variables and their coefficients we obtain: $a\overline{\mid p_x \mid} \approx 8.5 \cdot 10^3$, $b\overline{\mid p_y \mid} \approx 8.7 \cdot 10^3$, $c\overline{\mid v_x \mid} \approx 7 \cdot 10^2$, $d\overline{\mid v_y \mid} \approx 2.5 \cdot 10^2$, $e\overline{\mid \theta \mid} \approx 1.3 \cdot 10^2$, $f\overline{\mid \omega \mid} \approx 4.7 \cdot 10^2$. This suggests that we can neglect the values of $p_x$, $p_y$, $\theta$ because their mean value is low w.r.t. 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:

$$\omega < \frac{c}{f} v_x + \frac{d}{f} v_y \tag{34}$$

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 $\omega \geq c'v_x + d'v_y$, we the agent turns on the main engine to balance the high angular velocity of the agent. Note, again, that if the angular velocity is too low it is balanced by the previous condition.

*5.5.4. Considerations*

In this subsection, we interpreted the policy produced in various settings. We showed that the decision trees produced are interpretable and give an understanding about how the agent works. It is important to note that in several cases we performed approximations to ease the understanding process. However, this is not a limitation of the method, because more exact interpretations can be obtained by not neglecting details. This is especially important in high-stakes or safety-critical settings, where humans need to have a thorough understanding to validate and trust the systems produced.

Finally, while some solutions may seem hard to interpret (i.e. oblique decision trees), it is important to see the them in a bigger context: while they may not be easy to interpret at a first sight, their analysis is pretty straightforward (as shown earlier). On the other hand, black-box models (such as deep neural networks) are way harder to inspect, due to the significantly bigger number of operations performed in the decision making process.

## 6. Conclusions

While in recent years AI made a huge progress, the need of being able to understand *how* a model works is becoming more and more important. To overcome this issue, significant effort was put to advance the XAI field. However, XAI is not always a suitable solution. In fact, they suffer from some problems that make their use unsafe in safety-critical or high-stakes processes.

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

In this paper, we propose a two-level optimization method that allows to induce decision trees that can perform reinforcement learning.

Our results show that the proposed approach is able to generate decision trees that are comparable or even better than the non-interpretable state-of-the-art (from the performance point of view) while having significantly better interpretability. Furthermore, the results obtained in this work suggest that the widely though performance-interpretability trade-off does not always hold (as suggested by [1]) and that interpretable models can be competitive with state-of-the-art techniques. For this reason, research in this field must be encouraged.

Moreover, we compared the solutions obtained to the state-of-the-art from the point of view of the interpretability. While the metric of interpretability does not perfectly suit our purpose, we can easily observe the difference in complexity with respect to black-box models. While we expect that changing the metric of interpretability does not significantly affect the difference w.r.t. black-box models, we think that future work should focus on the study of more tailored interpretability metrics (i.e. tailored on machine learning models).

Since it is important for practical applications, we also compared our solutions to the interpretable (and publicly available) state-of-the-art w.r.t. robustness to input noise. The results show that our approach is comparably or more robust than the other solutions.

Finally, we demonstrated that the produced agents can be interpreted, practically showing the advantage of interpretable models w.r.t. black boxes.

Other future developments include: experimental tests on more complex reinforcement learning domains; the extension of the proposed method to the imitation learning domain; the development of a method that can automatically tune the constants, reducing the prior knowledge that must be 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] C. Rudin, Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, Nature Machine Intelligence 1 (5) (2019) 206–215.

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

[3] A. Silva, M. Gombolay, T. Killian, I. Jimenez, S.-H. Son, Optimization Methods for Interpretable Differentiable Decision Trees Applied to Reinforcement Learning, in: International Conference on Artificial Intelligence and Statistics, 2020, pp. 1855–1865.

[4] Y. Dhebar, K. Deb, S. Nageshrao, L. Zhu, D. Filev, Interpretable-AI Policies using Evolutionary Nonlinear Decision Trees for Discrete Action Systems, arXiv:2009.09521 (2020).

[5] A. K. Mccallum, D. Ballard, Reinforcement learning with selective perception and hidden state, Ph.D. thesis, The University of Rochester, Eastman School of Music (1996).

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

[7] L. D. Pyeatt, A. E. Howe, Decision tree function approximation in reinforcement learning, Tech. rep., Colorado State University (1998).

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

[9] D. Perez, M. Nicolau, M. O'Neill, A. Brabazon, Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution, in: Applications of Evolutionary Computation, Springer, Berlin, Heidelberg, 2011, pp. 123–132.

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

[11] A. Hallawa, T. Born, A. Schmeink, G. Dartmann, A. Peine, L. Martin, G. Iacca, A. E. Eiben, G. Ascheid, EVO-RL: Evolutionary-Driven Reinforcement Learning, arXiv:2007.04725 [cs, stat]ArXiv: 2007.04725 (Jul. 2020).

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

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

[14] M. Czajkowski, M. Krętowski, A multi-objective evolutionary approach to Pareto-optimal model trees, Soft Computing 23 (5) (2019) 1423–1437.

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

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

[17] L. Hyafil, R. L. Rivest, Constructing optimal binary decision trees is NP-complete, Information Processing Letters 5 (1) (1976) 15–17.

[18] G. Fan, J. B. Gray, Regression tree analysis using TARGET, Journal of Computational and Graphical Statistics 14 (1) (2005) 206–218.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, arXiv:1606.01540 (2016).

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

[21] Z. C. Lipton, The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery, Queue 16 (3) (2018) 31–57.

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

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

[24] J. Xuan, J. Lu, Z. Yan, G. Zhang, Bayesian Deep Reinforcement Learning via Deep Kernel Learning, International Journal of Computational Intelligence Systems 12 (1) (2018) 164–171.

[25] R. Beltiukov, Optimizing Q-Learning with K-FAC Algorithm, in: International Conference on Analysis of Images, Social Networks and Texts, Springer, Cham, 2020, pp. 3–8.

[26] J. Liu, X. Gu, S. Liu, D. Zhang, Soft Q-network, arXiv:1912.10891 [cs] (2019). `arXiv:1912.10891`.

[27] X. B. Peng, A. Kumar, G. Zhang, S. Levine, Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning, arXiv:1910.00177 [cs, stat]ArXiv: 1910.00177 (Oct. 2019).

[28] Z. Xu, L. Cao, X. Chen, Deep Reinforcement Learning with Adaptive Update Target Combination, The Computer Journal 63 (7) (2020) 995–1003.

[29] M. Malagon, J. Ceberio, Evolving Neural Networks in Reinforcement Learning by means of UMDAc, arXiv:1904.10932 [cs]ArXiv: 1904.10932 (Apr. 2019).

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal Policy Optimization Algorithms, arXiv:1707.06347 [cs]ArXiv: 1707.06347 (Aug. 2017).