UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

Evolutionary Optimization of Decision Trees for Interpretable Reinforcement Learning

Leonardo Lucio Custode

Advisor

Prof. Giovanni Iacca

Università degli Studi di Trento

February 2023

Abstract

While Artificial Intelligence (AI) is making giant steps, it is also raising concerns about its trustworthiness, due to the fact that widely-used black-box models cannot be exactly understood by humans. One of the ways to improve humans' trust towards AI is to use interpretable AI models, i.e., models that can be thoroughly understood by humans, and thus trusted. However, interpretable AI models are not typically used in practice, as they are thought to be less performing than black-box models. This is more evident in Reinforcement Learning, where relatively little work addresses the problem of performing Reinforcement Learning with interpretable models. In this thesis, we address this gap, proposing methods for Interpretable Reinforcement Learning. For this purpose, we optimize Decision Trees by combining Reinforcement Learning with Evolutionary Computation techniques, which allows us to overcome some of the challenges tied to optimizing Decision Trees in Reinforcement Learning scenarios. The experimental results show that these approaches are competitive with the state-of-the-art score while being extremely easier to interpret. Finally, we show the practical importance of Interpretable AI by digging into the inner working of the solutions obtained.

Keywords
Interpretable AI, Reinforcement Learning, Evolutionary Computation

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last decade, the artificial intelligence (AI) field has seen a series of breakthroughs. In fact, in such a small period, several milestones have been reached: the rise of convolutional neural networks (CNNs) for computer vision (CV) [60, 131, 47, 29, 37], the astonishing achievements in the field of Reinforcement Learning (RL) [85, 148, 93], the advances in Natural Language Processing [14, 107], image-to-image and text-to-image generative models [147, 109, 108, 113].

The vast majority of these breakthroughs are due to the progress made in the field of artificial neural networks (NNs). NNs are computational models that (loosely) mimic the functioning of biological neural networks, in that they are composed of artificial "neurons" interconnected by synapses. While these models are extremely powerful and scalable, as confirmed by the exciting results described above, they suffer from a fundamental issue that can prevent their application in safety-critical and high-stakes scenarios: deep NNs are tremendously hard to understand for a human. More concisely, we could say that NNs are not *interpretable.* Interpretability is a structural property of a machine learning model that depicts the ability of a model to be clearly and *exactly* understood by a human [6].

The last sentence of the previous paragraph begs the question: why is interpretability so important in machine learning? In the next paragraphs, we will provide some answers to this question.

Bias is currently one of the biggest issues in machine learning. In fact, bias heavily depends on the statistical distribution of the training data.

In the past years, several events raised concerns about bias in machine learning. Some of the most widely known incidents are shown below. An ML model produced by Amazon for screening job applicants was found to be gender-biased, penalizing terms in a CV that were related to women[1]. An image labeling model from Google photos mistakenly labeled

[1]https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G

several pictures of a black couple as "gorillas" [2]. Similarly, a video-captioning model from Facebook erroneously mislabeled footage of black people as "primates"[3].

These issues are caused by the fact that it is very hard to verify the *absence* of biases in a non-interpretable machine learning model.

Interpretable ML models, instead, can prevent this type of issue by enabling inspection before the deployment, to assess that the knowledge learned by the model is not biased in any way.

Debugging a model is important to avoid catastrophic outcomes when deploying a model. When dealing with complex, non-interpretable models, testing all the possible outcomes becomes unfeasible. However, when deploying a model in a safety-critical scenario even a thoroughly tested non-interpretable model can behave differently from what the testing phase showed.

One of the most shocking cases happened when Microsoft deployed an ML chatbot on Twitter that could learn from its interactions with the users. However, the model's learning mechanism was exploited by a subset of the community to turn the chatbot into a sexist and racist chatbot[4]. Another tragic type of event is when some malfunctioning happens in ML models for autonomous driving[5][6], which can easily lead to fatalities. Finally, this type of incident also happened in the medical domain, where an AI built by IBM was found to suggest fundamentally wrong treatments to patients[7].

Discovery is what enables progress. Machine Learning can be a game-changing enabler to the automated discovery field. However, using non-interpretable approaches to automated discovery may be short-sighted. To have a sustainable approach to ML-aided research, there must be a symbiosis between human researchers and machine learning algorithms: ideally, the researcher should tell the algorithm what problem they want to solve, then the algorithm finds a solution and, most importantly, the researcher should be able to understand the solution generated by the machine learning model so that the solution can be improved and the discovery cycle can be re-iterated.

To face non-understandability issues in Machine Learning, in the last years the fields of eXplainable AI (XAI) and Interpretable AI (IAI) have seen a growing interest. While they may seem the same thing, there are some fundamental differences between them [6].

[2]https://www.bbc.com/news/technology-33347866
[3]https://www.nytimes.com/2021/09/03/technology/facebook-ai-race-primates.html
[4]https://www.bbc.com/news/technology-35902104
[5]https://www.nytimes.com/2021/04/18/business/tesla-fatal-crash-texas.html
[6]https://www.washingtonpost.com/technology/2022/06/15/tesla-autopilot-crashes/
[7]https://www.theverge.com/2018/7/26/17619382/ibms-watson-cancer-ai-healthcare-science

As stated in [6], explainable models are a set of models whose *behavior* can be explained. Thus, explainability is a *behavioral* property of a model, that tells us that it is possible, by observing the model's inputs and outputs, to obtain some *insights* about its inner functioning. On the other hand, interpretability is an *intrinsic* property of a model, i.e., it can be easily understood (by humans) by inspecting its components.

This distinction is crucial: as an example, since we can analyze an explainable model by using post-hoc techniques, we may be tempted to say that we fully understand why a decision was taken. However, this reasoning is faulty: just because we have a (perhaps convincing) explanation about the behavior of a model, we cannot be entirely sure that the explanation was faithful to the model's "reasoning". Most widely-known XAI approaches seek to characterize the behavior of a model by either:

1. Building local explanations [111, 74, 110, 112, 32, 55]

2. Approximating the behavior of a model [54, 53, 7, 8, 61]

While this may be satisfactory in some scenarios, it is important to note that XAI approaches often *approximate* the model. This is a very important caveat: as approximations, they are by definition *not exact*, which may not be acceptable in safety-critical or high-stakes scenarios. In this regard, several works show that explanations provided by XAI approaches may be brittle, and not reflective of the internal state of the explained model.

For instance, in [116], the authors explain a prediction of an image classifier using saliency maps [130]. Then, after applying an adversarial perturbation to the image, they show that the saliency map is practically unchanged, while the output of the classifier is significantly different. Similarly, in [133], the authors show that it is possible to fool popular XAI algorithms such as LIME [111] and SHAP [74] by explaining racist classifiers with non-racist explanations. This last example emphatically shows that XAI approaches are unsuitable in scenarios with potentially harmful outcomes to people, society, and the economy.

Moreover, the importance of interpretability has been pointed out by both the European Union and UNESCO. The European Union, in a document[8], proposed the AI Act[9], a set of requirements to be met by modern AI systems. In this act, the EU emphasizes the need for the right to inspect and understand models, which is deemed one of the most important rights in the AI Act. On the other hand, UNESCO released a set of recommendations for ethics in AI[10]. In this document, they state that in order to avoid

[8]`https://eur-lex.europa.eu/resource.html?uri=cellar:e0649735-a372-11eb-9585-01aa75ed71a1.0001.02/DOC_1&format=PDF`

[9]`artificialintelligenceact.eu`

[10]`https://unesdoc.unesco.org/ark:/48223/pf0000380455/`

inequality, discrimination, and threats to diversity, AI models should be understandable and transparent.

The reports released by these two institutions mark a new milestone in the field of AI. In fact, while the usefulness of AI in modern society is not only recognized but given for granted, they point out the pitfalls that come with AI, asking for transparent, interpretable machine learning models.

While the field of interpretable AI is making a lot of progress, not all subfields are growing equally. In [118], the authors list a number of challenges that, if solved, would be game-changing for the field of Interpretable Machine Learning. One of the challenges described in the paper is Interpretable Reinforcement Learning.

In fact, while the field of Interpretable Machine Learning is making good progress, Interpretable Reinforcement Learning is left behind, as little research addresses the issues appearing in Reinforcement Learning (RL) with interpretable models.

Reinforcement Learning is an ML paradigm in which there is an agent that has to learn how to complete a task by interacting with an environment. The typical flow in an RL setting is the following. The agent receives an observation of the state of the environment, and it has to decide what is the most appropriate action to perform in response to the observation. Once the action has been deployed in the environment, the agent receives a reward that describes the quality of the action taken by the agent. Then, the agent can leverage the reward provided by the environment to improve itself and increase its performance. The RL paradigm can be useful to discover policies that allow to perform well in tasks with unknown dynamic or where the dynamic of the environment can change (e.g., real-world physical systems may degrade over time, and thus slight changes of the policies may be needed). Moreover, RL is very useful when the agent has to continually adapt to new tasks. RL is the enabling technology behind several breakthrough in games [122, 93] and it was recently used to improve conversational chatbots using human feedback [95].

In this thesis, we propose methods to cope with the issues arising in Interpretable RL (IRL) and, in general, in Interpretable Machine Learning (IML). Regarding the IRL problem, we will present methods for different scenarios: single- and multi-agent setups, small and large input spaces, and their application to real-world problems.

This thesis is structured as follows. Chapter 2 makes a brief overview of related work in the literature. In Chapter 3, we present methods that combine techniques from the evolutionary optimization (and automatic program synthesis) and the reinforcement learning fields to produce interpretable agents. Chapter 4 extends the previous chapter to produce agents that can act in environments with continuous action spaces. In Chapter 5, we describe a method that, exploiting co-evolutionary algorithms, can produce interpretable agents for RL environments whose state is described through images. Chapter 6 proposes an extension of the method proposed in Chapter 3 to multi-agent scenarios. Then, Chapter 7 shows the application of the methods proposed in the previous section to real-world problems. Chapter 8 draws the conclusions and lists potential future directions for the field.

Chapter 2

Related Work

2.1 Decision Trees for Reinforcement Learning

A RL problem can be represented through a Markov Decision Process (MDP). An MDP is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{S}_0 >$, where $\mathcal{S}$ is a set of states (a single state is defined as $s \in \mathcal{S}$), $\mathcal{A}$ is a set of actions that can be taken by the agent, $\mathcal{P}(s, a, \cdot)$ describes the probability distribution over $\mathcal{S}$ when action a is taken in the state s. $\mathcal{R}(s, a, s')$ is the *reward* function, i.e., the function that associates a reward to each state transition. γ, instead, represents a discount factor that is used to give a higher weight to immediate rewards and a smaller weight to future rewards. Finally, $\mathcal{S}_0$ describes the set of initial states.

A $\mathcal{Q}$ function is a function that can be used to estimate the quality of each action-state pair in the MDP. Once a $\mathcal{Q}$ function has been learned (i.e., its error has reached a relatively small value), it can be used as a policy by choosing, for each input, the optimal action (i.e, the action that maximizes the $\mathcal{Q}$ function):

$$a^*(s) = \underset{a}{argmax}(Q(s, a))$$

Usually, $\mathcal{Q}$ functions are either encoded in tables (called $\mathcal{Q}$ tables) or approximated using function approximators. Function approximators are used to cope with the high cardinality of the set $\mathcal{S}$: $|\mathcal{S}|$. For instance, in Deep Q Learning [85], a Deep Neural Network (DNN) is used to approximate a $\mathcal{Q}$ function [151].

The use of Decision Trees (DTs) to solve RL tasks has been explored in several previous works. McCallum, in [79], proposes U-Trees: a kind of tree able to perform RL. Those are created by dividing the RL problem into three sub-problems: choice of memories, selective perception, and value function approximation. In [146], the authors extend U-Trees to make them able to cope with environments with continuous state spaces. They propose two novel tests that are used to create new conditions that split the state space. They test the proposed approach in two environments, a continuous one and an ordered-discrete one, and their results show that their approach is competitive with other methods.

Pyeatt and Howe, in [104], propose a novel splitting criterion to build trees that can perform value function approximation. In their experiments, they compare the performance obtained by using their approach to the ones obtained using other splitting criteria, a table-lookup approach, and a neural network. The results demonstrate that the proposed approach often outperforms all other methodologies.

In [114], the authors propose a method that predicts the gain obtained by adding a split (i.e., a condition) to the tree and selecting the best split to grow the tree. The experimental results show that this method is more effective than the method proposed in [104] on the tested environment.

The approaches described are capable of producing DTs with good interpretability (due to the fact that the DTs produced use simple conditions). However, they suffer from a major drawback: the curse of dimensionality. This means that these approaches do not scale well with the number of inputs, making their optimization harder as the number of inputs increases and thus hindering their application to challenging environments.

Silva et al. [129] propose an approach to interpretable RL that uses Proximal Policy Optimization (PPO) [128] on differentiable DTs. The experimental results show that this approach can produce competitive solutions in some of the tested tasks. However, it is also shown that when the differentiable DTs are discretized into traditional (i.e., discrete) DTs, their performance may heavily decrease. A continuous version of this approach has been proposed in [96].

In [28], the authors use Evolutionary Algorithms (EAs) to evolve non-linear DTs. By non-linear, the authors mean that each split does not define a linear hyperplane in the feature space. The experimental results show that this approach can obtain competitive performance w.r.t. a neural-network-based approach. A case study for this approach is presented in [42]. While this approach is able to solve a variety of tasks, the type of splits used make their interpretation hard, as they define potentially complex hyperplanes for the conditions.

In [99], the authors use the Grammatical Evolution (GE) algorithm [120] to evolve behavior trees (i.e., tree structures that allow more complex operations than a DT) for the Mario AI competition. The proposed agent can perform basic actions or pre-determined combinations of basic actions. Their solution achieved fourth place in the Mario AI competition. However, the authors only evolve a controller, not exploiting the rewards given by the environment to increase the performance of the agent.

Hallawa et al., in [44], use behavior trees as evolved "instinctive" behaviors that are then combined with a learned behavior. While behavior trees are usually interpretable, the authors do not take explicitly into account the interpretability of the whole model, which comprises both a behavior tree and either a neural network or a $\mathcal{Q}$-learning table.

2.2 Nature-inspired methods for DTs

Several approaches have been proposed to build decision trees with nature-inspired techniques. In [16] the authors replace the greedy algorithm in OC1 [89] by using simulated annealing, a genetic algorithm, and evolution strategies. The results show that the proposed variants are competitive w.r.t. the original version. Llora et al. [70, 71] use a framework called "Genetic and Artificial Life Environment" (GALE) to induce decision trees for classification. Their results show that their approach can obtain state-of-the-art performance. Czajkowski et al., in [25], use a multi-objective evolutionary algorithm (EA) to simultaneously optimize the performance of the decision trees and some of their properties. The results show that, by using the proposed method, the authors achieve competitive performance while obtaining simpler trees w.r.t. the state of the art. In [11] the author applies genetic programming (GP) (as done also in [57]) to the induction of decision trees for classification purposes. The results show that the obtained results are comparable with those obtained in the OC1 induction algorithm while having slightly worse performance than M5' [39] and C5.0 [106]. In [59], the author defines a memetic algorithm (i.e., an evolutionary algorithm combined with a local optimization algorithm) that uses GP to induce decision trees. In [36], the authors present an approach to evolve decision trees based on GP and cellular automata. The results show that this approach has comparable (often better) performance w.r.t. C4.5. Heath et al. [48] propose an approach to the induction of decision trees based on simulated annealing, obtaining trees that are comparable in performance to the ID3 [105] algorithm while being smaller in size.

2.3 Interpretability

While several work approached the RL problem by using decision trees as function approximators, to our knowledge, none of them addressed the problem of measuring the interpretability of the agents produced. Interpretability [6] is defined as an *intrinsic* property of a system that tells us whether the system can be easily understood by humans. It must not be confused with explainability, which is a *behavioral* property of a system, and tells us whether we are able to understand the decisions taken by the model by using post hoc techniques.

While interpretability is defined as a *qualitative* metric [6, 2], two works tried to build a *quantitative* metric to measure the interpretability of a model.

In [149], the authors build a formula of interpretability by analyzing the results of a survey and building a model. The model obtained from their survey is the following:

$$\mathcal{M}_o = 79.1 - 0.2l - 0.5n_o - 3.4n_{nao} - 4.5n_{naoc} \tag{2.1}$$

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.

On the other hand, in [5], the authors suggest that there could be a relationship between the computational complexity of a model and its interpretability.

Finally, in [66], the author warns that confounding transparency and interpretability (from the definitions provided in [6]) may be harmful, as huge transparent models may be as hard to interpret as black-box models.

2.4 Multi-Agent Reinforcement Learning

This section is a short summary of related work in the field of Multi-Agent Reinforcement Learning (MARL). For a more complete review, we refer the reader to [15, 94, 137, 158].

In a preliminary work [137], the authors explained the advantages of adopting a multi-agent approach instead of a single, complex agent approach. Several approaches have then been proposed for MARL. In [123] the authors compared two function approximators in the iterated prisoner's dilemma: a table-based approach and a recurrent neural network (RNN). The experiments showed that the agents based on the tabular approach were more prone to cooperate than the ones trained using the RNN, indicating that the agents trained by using the tabular approach had learned a better approximation of the Q function. Littman [67] presented a novel algorithm based on Q-learning and minimax, named "minimax Q". This algorithm, in the experimental results, proved to be able to learn policies that were more robust than the policies learned by Q-learning. In [46] the authors made use of cooperative co-evolution with strongly-typed genetic programming (GP) to evolve agents for a predator-prey game. The evolved strategies were more effective than handcrafted policies.

Independent Q-learning (IQL) [141] is another convenient approach to MARL, as it is scalable and decentralized. However, when using neural networks as function approximators for reinforcement learning, this method cannot be applied. In fact, the need for a replay buffer does not make this method suitable in settings with neural networks. To mitigate this issue, several approaches have been proposed [65, 41, 92, 78, 140]. Other approaches circumvent this problem by using instead the actor-critic model [19, 132, 76, 138, 156].

Chapter 3

Hybrid approaches for Interpretable Reinforcement Learning

Based on: Custode, Leonardo L., and Giovanni Iacca. "Evolutionary learning of interpretable decision trees." IEEE Access (2023).

3.1 Introduction

During the last decade, the Reinforcement Learning (RL) field saw various breakthroughs. The prime examples of these achievements can be seen in AlphaStar [148], i.e., a system that can beat professional StarCraft II players; OpenAI Five [93], which is a model that was able to beat the Dota2 world champion; and MuZero [125], which is a system that, by combining deep learning with Monte-Carlo Tree Search, was able to reach superhuman performance in a wide variety of tasks.

While these accomplishments are remarkable, modern RL approaches suffer from a critical issue: the lack of interpretability. Even though their performance surpassed human levels, their application in safety-critical, high-stakes real-world scenarios is limited. Indeed, since these models are huge, it is very difficult to gain knowledge about them by inspecting their inner functioning. This lack of interpretability also leads to a lack of trustworthiness, which is essential for the real-world application of RL systems.

On the other hand, research in the Interpretable RL (IRL) field has received little attention. For instance, methods that perform IRL using Decision Trees (DTs) do exist. However, the approaches used for DT induction mostly use greedy algorithms, which suffer from the curse of dimensionality.

In this chapter, we present a novel approach to DT induction for IRL, which combines the advantages of Evolutionary Algorithms (EAs) with RL methods. Combining these two methodologies allows us to take the best of both worlds. In fact, EAs allow us to optimize DTs in such a way that the computational cost can be controlled and thus it is not related to the dimensionality of the input. On the other hand, RL allows the DTs generated by the EA to learn in an online manner, increasing significantly the efficiency of the search process. The experimental results confirm this hypothesis, showing that this approach is comparable to or better than using an EA alone and that this method allows us to outperform other IRL approaches. Moreover, by comparing these results with the non-interpretable state of the art we observe that the widely-thought performance-interpretability trade-off does not always hold.

This chapter is structured as follows. The next section makes an overview of the related work. Section 3.2 explains the method used in this work and the experimental setup. In Section 3.3 we present the experimental results, a comparison with the (interpretable) state of the art, an ablation study, and the interpretation of the solutions. Finally, Section 3.4 draws the conclusions of this chapter.

3.2 Method

Our goal is to solve an RL problem described by a Markov Decision Process (MDP) using a DT as policy.

It is interesting to note that, by using a DNN to represent a $\mathcal{Q}$ function, we are using a $\mathcal{Q}$ table that is infinitely large, i.e., it can potentially compute a different value for each different state $s \in \mathbb{R}^d$, where d is the number of variables composing an observation s (please, note that s can be represented as a vector where each dimension accounts for a distinct input. With a slight abuse of notation, we will indicate this vector s).

On the other hand, using DTs we aim to solve the problem in the opposite way: given $\mathcal{S}$, we want to find an optimal partitioning (thus, denoted by a star) $\mathcal{S}_p^*$ of the space that allows us to find an optimal action for each partition $p \in \mathcal{S}_p^*$. A partition of the state space is a set of sets, each of which contains semantically similar states. Thus, the following properties hold:

- $\bigcup_{p \in \mathcal{S}_p} (p) \equiv \mathcal{S}$
- $\bigcap_{p \in \mathcal{S}_p} (p) \equiv \emptyset$

Note that these properties apply to any possible partitioning $\mathcal{S}_p$ of $\mathcal{S}$, not just to the optimal one ($\mathcal{S}_p^*$).

Thus, we can *decompose* the RL problem into two subproblems:

- RL_1: the problem of finding the optimal partitioning $\mathcal{S}_p^*$,

- RL_2: the problem of finding the optimal action a_p^* for each partition $p \in \mathcal{S}_p^*$.

To the best of our knowledge, this is the first work that decomposes the RL problem in this way.

In similar works, [79, 146, 104, 114] the authors build DTs for RL by using greedy heuristics. However, this may not be the best choice, for the following reasons:

1. the DT induction problem has been proved to be NP-complete [51]. Thus, greedy approaches may produce DTs that are suboptimal [79, 33]

2. the trees are expanded by using tests that suffer from the curse of the dimensionality [79, 146], since the tests need to be performed on each of the input variables

To solve these issues, other approaches [70, 71, 59, 99, 25] make use of EAs to induce DTs. However, since most of these works are not applied to RL problems (except for [99]), these approaches cannot exploit the reward signals given by an RL environment to improve their efficiency.

In this chapter, we propose a method that is specifically tailored to the optimization of DTs for RL. This algorithm, in fact, is designed to optimize simultaneously the two subproblems RL_1 and RL_2 discussed earlier. The way in which we approach this problem is the following:

1. we use Grammatical Evolution (GE) [120] (an EA that can evolve solutions based on a context-free grammar) to search for DT structures that perform a (not necessarily optimal) partitioning $\mathcal{S}_p$ of the state space $\mathcal{S}$,

2. all the solutions generated by GE undergo, in the fitness evaluation phase, an RL process by using the $\mathcal{Q}$-Learning algorithm [151], which allows them to learn from the feedback received from the environment.

This can be seen as a two-level optimization algorithm, in which there is an outer loop that optimizes the structure of the DT by using GE, and an inner loop that optimizes the content of the leaves by means of $\mathcal{Q}$-learning. The pseudocode for the GE algorithm is shown in Algorithm 1, while the description of the fitness function is shown in Algorithm 3. Figure 3.1 shows graphically the functioning of the proposed algorithm.

Note that, while the knowledge learned by the agent does not directly propagate to its offspring, it has a huge impact on the search landscape. Thus, it can be classified as a Baldwinian evolutionary approach.

The details of the algorithm are described below.

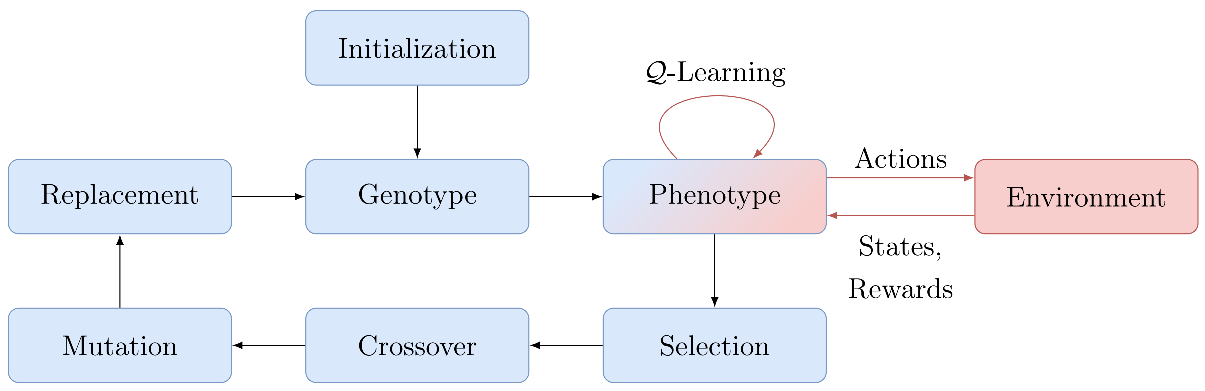


Figure 3.1: Block diagram of the functioning of the proposed algorithm. Blue blocks refer to Grammatical Evolution, while orange blocks refer to $\mathcal{Q}$-Learning.

Algorithm 1: Optimization of DTs by using GE and $\mathcal{Q}$-Learning

```
Data: $n_p, n_g, p_x, p_m, s_t, n_e, \alpha, \gamma$
Result: $\widehat{T}^*$: the best DT found
1  $pop \leftarrow initializePop(n_p)$;
2  $evaluate(pop)$;
3  $best \leftarrow getBest(pop)$;
4  for $gen \in [1, n_g]$ do
5      $new\_pop \leftarrow select(pop, s_t)$;
6      $new\_pop \leftarrow crossover(new\_pop, p_x)$;
7      $new\_pop \leftarrow mutate(new\_pop, p_m)$;
8      $new\_pop \leftarrow evaluate(new\_pop, n_e, \alpha, \gamma)$;
9      $pop \leftarrow replace(new\_pop, pop)$;
10     $best \leftarrow getBest(pop)$;
11 end
12 return $best$;
```

Individual Encoding

In the GE algorithm, a genotype corresponds to a list of indices. These indices are then used to build a phenotype (i.e., a solution) by querying the grammar. In fact, initially, each phenotype is composed of a starting rule (usually represented by a string enclosed in angle brackets). Then, for each of the codons (i.e., the elements of the list), the first non-expanded rule is expanded. To expand a rule, we query the grammar to get the production of that rule, which is a list of possible strings (that may contain other rules), and choose one of them. Then, we use the indices contained on the list: at the i-th step, we choose the index contained at the i-th location of the list that represents an individual.

In some cases, the index contained in the list may exceed the length of the production. To solve this issue, we use the modulo operator to ensure that we use valid indices.

Each individual is thus encoded as a list of s_i integers (a hyperparameter) $g_i \in [0, m]$, where m is a number much greater than the maximum number of productions for the grammar, to ensure uniform sampling probability of each production in the grammar.

Finally, we represent each DT as a `Python` program. This means that each solution encodes a DT encoded as combinations of `if-then-else` statements.

The pseudocode of the genotype-to-phenotype mapping (i.e., how a list is transformed in a solution) is shown in Algorithm 2.

Algorithm 2: Genotype-to-phenotype mapping

```
Data: $individual, grammar$
Result: $sol$: the corresponding Decision Tree encoded as a Python program
1 $sol \leftarrow$ "$\langle start \rangle$";
2 for $codon \in individual$ do
3     $rule \leftarrow getFirstRule(sol)$;
4     $production \leftarrow grammar[rule]$;
5     $nProductions \leftarrow length(grammar[rule])$;
6     $value \leftarrow production[codon \ mod \ nProductions]$;
7     $sol \leftarrow replaceFirstOccurrence(sol, rule, value)$;
8 end
9 return $sol$;
```

Selection

To decide which of the individuals must be chosen to produce new solutions (i.e., their offspring), we use tournament selection. This means that, for a given population size n_p, n_p tournaments are created, each of which contains s_t individuals (s_t is a hyperparameter). For each of the tournaments, the best individual in the tournament is selected for reproduction.

Crossover

In order to recombine two solutions, we use single-point crossover. This algorithm chooses a random point in the genome and splits each genome into two parts. Then, it creates two new individuals by mixing the parts of genomes created earlier.

Mutation

To mutate the individuals, we use the random mutation operator. This operator mutates the value of each codon with probability p_c by assigning it a new random value in $[0, m]$.

Replacement

In order to increase the exploitation capabilities of this approach, we do not replace a population with its offspring. Instead, a parent is replaced if and only if at least one of its offspring performs better than it.

3.2.1 Fitness Evaluation

Once the GE algorithm produces a DT T, this needs to be evaluated on the RL problem.

Thus, at each step, T computes the id i of the leaf that is responsible for the current state. Then, the leaf l_i, which contains a list of $\mathcal{Q}$ values, decides what action to take according to an ε-greedy $\mathcal{Q}$-Learning policy:

$$a = \begin{cases} \underset{a^*}{argmax}(l_i(a^*)) & \textit{with prob. } (1-\varepsilon) \\ random & \textit{with prob. } \varepsilon \end{cases} \tag{3.1}$$

Then, after taking action a, T observes the reward r and the next state s', which is associated to a potentially different leaf j, and updates the values of leaf i by using the Bellman equation:

$$l_i(a) = (1-\alpha)l_i(a) + \alpha[r + \gamma \cdot \underset{a'}{max}(l_j(a'))] \tag{3.2}$$

This can be rewritten in a form that is more similar to the original $\mathcal{Q}$-Learning equation by using $T(s, \cdot)$ as a proxy for the corresponding leaves:

$$T(s, a) = (1-\alpha)T(s, a) + \alpha[r + \gamma \cdot \underset{a'}{max}(T(s', a'))] \tag{3.3}$$

Thus, by using the rules shown above for choosing actions and learning their corresponding value, we simulate n_e episodes with the current tree T. At the end of the last episode, the mean score obtained by T is returned. The pseudocode for the fitness evaluation function is shown in Algorithm 3.

Finally, if a genotype cannot be translated into a phenotype (e.g., because after the expansion of the last codon, there are still non-expanded rules), it is assigned a fitness of -10^3, i.e., a value much lower than the ones that a valid agent can obtain.

Algorithm 3: Decision Tree evaluation using $\mathcal{Q}$-Learning

```
Function evaluate(pop, $n_e$, $\alpha$, $\gamma$): $fitnesses$ is
    $fitnesses \leftarrow []$;
    for $genotype \in pop$ do
        $T \leftarrow translateGenotypeIntoTree(genotype)$;
        $scores \leftarrow []$;
        for $i \in [1, n_e]$ do
            $env, state \leftarrow getEnvironment()$;
            $score \leftarrow 0$;
            $done \leftarrow False$;
            while $!done$ do
                $action \leftarrow T(state)$;
                $reward, next\_state, done \leftarrow env.step(action)$;
                $T[state, action] \leftarrow (1-\alpha)T[state, action] + \alpha(r + \gamma max(T[next\_state]))$;
                $state \leftarrow next\_state$;
                $score \leftarrow score + reward$;
            end
            $scores \leftarrow scores + [score]$;
        end
        $fitnesses \leftarrow fitnesses + [mean(scores)]$;
    end
    return $fitnesses$;
end
```

3.2.2 Experimental setup

To assess the capabilities of the method proposed in this section, we test it in three widely-used RL benchmarks, implemented in OpenAI Gym [12]: CartPole-v1, MountainCar-v0, and LunarLander-v2.

Description of the environments

CartPole-v1 This task consists in balancing a pole that is located on top of a cart. The agent has to move the cart either to the left or to the right to make the pole stand.

Observation space The variables describing the state of the cart-and-pole system are the following:

- 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 agent can balance the pole by either moving the cart:

- to the left, with a force of 10N ($left$)
- to the right, with a force of 10N ($right$)

Rewards The reward obtained by the agent for keeping the pole correctly balanced is 1 point for each timestep.

Termination criterion The simulation terminates if:

- $|x| > 2.4$
- $|\theta| > 0.418 rad$
- Length of the simulation exceeds 500 steps

Resolution criterion An agent is said to solve this task if its mean return is $R \geq 475$, computed on 100 runs.

MountainCar-v0 This environment is composed of a valley (surrounded by two hills), and a car. The car, driven by the agent, has to reach the top of the right hill. However, the car's engine cannot produce the torque needed to climb the hill, thus the agent has to learn to swing between the two hills to build momentum and climb the right hill.

Observation space The inputs to the agent are:

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

Action space The agent can perform 3 actions:

1. Accelerate, with a force of $10^{-3}N$, to the left: $left$
2. Do not accelerate in any direction: nop
3. Accelerate, with a force of $10^{-3}N$, to the right: $right$

Rewards At each timestep, the agent receives a penalty of -1 point, which should encourage the agent to solve the task as quickly as possible.

Termination criterion An episode has a length of 200 steps. No other event can terminate the simulation earlier.

Resolution criterion To solve this task, an agent has to receive a mean return $R \geq -110$, computed on 100 runs.

LunarLander-v2 In this environment, an agent has to pilot a rocket to make it land in a pre-defined area of a planet's surface.

Observation space The agent receives the following observations:

- Rocket's position: $p_x, p_y \in [-1.5, 1.5]$ m
- Rocket's velocity: $v_x, v_y \in [-5, 5]$ m/s
- Angle w.r.t. the vertical axis: $\theta \in [-\pi, \pi]$ rad
- Angular velocity: $\omega \in [-5, 5]$ rad/s
- Whether the left/right legs are touching the ground: $c_l, c_r \in \{0, 1\}$

Action space The agent can control the rocket by performing the following actions:

1. Do not fire any engine: nop
2. Fire the left engine: $left$
3. Fire the main engine: $main$
4. Fire the right engine: $right$

Rewards The agent receives a reward varying from 100 to 140 points for landing with a final velocity of zero. If it crashes, instead, it receives a reward of -100 points. Finally, firing each of the engines leads to a penalty: firing the side engines gives a penalty of -0.03, while firing the main engine leads to a penalty of -0.3.

Termination criterion An episode is terminated in the following cases: the rocket crashes, the duration exceeds 1000 timesteps or the rocket moves outside the bounds of the environment.

Resolution criterion An agent is said to solve the task if its mean return $R \geq 200$, computed on 100 runs.

Metric of interpretability

Even though the proposed method only optimizes the score, when comparing our results to the state of the art not only we will use the mean score as the metric, but also use a "complexity of interpretability" metric derived from [149]. We define the "complexity of interpretability" as:

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

It is easy to observe that the proposed metric derived straightforwardly from the one proposed in [149]. This is due to the fact that this metric was created for relatively small mathematical formulae, whereas we are interested in assessing the interpretability of potentially large machine learning models. This is an issue with the original formula, as it is meant to lie in $[0, 100]$ and, with large formulae, it may exceed the bounds. Thus, the main differences between $\mathcal{M}$ and the original formulation are:

- The signs are flipped: this makes the metric work as a complexity. Thus, from the interpretability point of view, the lower the better,
- The initial constant is different: this allows the metric to be defined in $[0, \infty]$, so that when an ML model is simply a constant, its $\mathcal{M}_{const} = 0$, while e.g., with infinite parameters we will have $\mathcal{M}_{\infty} = \infty$.

Moreover, it is interesting to note that this formulation (and the one proposed in [149]) is consistent with the formulation proposed by Barceló et al. [5], where the authors propose the use of computational complexity to measure interpretability. This metric is also consistent with the critics moved to the field of interpretable AI in [66], where the author states that huge transparent machine learning models may not be interpretable.

Hyperparameters

The grammars used for the three environments are shown in Tables 3.1 to 3.3.

To better understand how grammar affects the outcome of the search process, we used two different grammars. The first one, which we call “orthogonal grammar”, produces orthogonal DTs; while the second one, called “oblique grammar”, produces oblique DTs.

An orthogonal DT is a DT that uses one variable for each condition (a condition is also called “split”). Thus, each split of the tree draws a hyperplane in the input space which is orthogonal to the axis of the variable being used in the split. On the other hand, an oblique DT uses a linear combination of all the inputs, and thus draws a hyperplane that is (potentially) oblique w.r.t. the axes of the input variables.

The hyperparameters used for GE and $\mathcal{Q}$-Learning, instead, are shown in Tables 3.4 to 3.6.

All the hyperparameters and grammars were tuned empirically, starting from well-known values.

Note that, for the LunarLander-v2 environment, we were not able to find any configuration of hyperparameters that was able to reach satisfactory performance with the orthogonal grammar (however, it is not clear whether such a set of hyperparameters does exist). Thus, the results for orthogonal DTs on LunarLander-v2 will not be shown.

Please, note also that to evolve DTs in the LunarLander-v2 environment, we use 10^3 episodes with early stopping, which works as follows. Each Δ_e episodes (except for the

Table 3.1: Grammars used for the evolution of DTs in the CartPole-v1 environment. "|" indicates the possible choices for a given rule. "comp_op" means "comparison operator". "lt" and "gt" stand for the "less than" and "greater than" operators. "$< \cdot, \cdot >$" denotes the dot product operator.

Rule	**Production (Orthogonal)**	**Production (Oblique)**
start	$\langle if \rangle$	$\langle if \rangle$
if	$if\ \langle condition \rangle\ then\ \langle action \rangle\ else\ \langle action \rangle$	$if\ \langle condition \rangle\ then\ \langle action \rangle\ else\ \langle action \rangle$
condition	$\langle comp\_op \rangle(x, \langle const_x \rangle) \mid$ $\langle comp\_op \rangle(v, \langle const_v \rangle) \mid$ $\langle comp\_op \rangle(\theta, \langle const_\theta \rangle) \mid$ $\langle comp\_op \rangle(\omega, \langle const_\omega \rangle)$	$lt((< \langle const \rangle, input >, \langle const \rangle)$
action	$leaf \mid \langle if \rangle$	$leaf \mid \langle if \rangle$
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	
$const$		[-1, 1) with step 0.001

initial Δ_e), the score on the last Δ_e episodes is computed. Then, it is compared to the previous mean (i.e., the one computed on the previous set of Δ_e episodes, i.e., ($[2\Delta_e, \Delta_e[$). If the current mean is lower, then we can assume that the performance of the agent has (almost) converged to its optimal value (guaranteed by the fact that we are using a dynamic learning rate of $\frac{1}{v}$, where v is the number of visits to each action of each leaf [139]), and stop the evaluation.

Simplification of the DTs

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 the test 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 common. The iteration stops when the tree does not contain such nodes.

Table 3.2: Grammars used for the evolution of DTs in the MountainCar-v0 environment. "|" indicates the possible choices for a given rule. "comp_op" means "comparison operator". "lt" and "gt" stand for the "less than" and "greater than" operators. "$< \cdot, \cdot >$" denotes the dot product operator.

Rule	Production (Orthogonal)	Production (Oblique)
start	$\langle if \rangle$	$\langle if \rangle$
if	$if\ \langle condition \rangle\ then\ \langle action \rangle\ else\ \langle action \rangle$	$if\ \langle condition \rangle\ then\ \langle action \rangle\ else\ \langle action \rangle$
condition	$\langle comp\_op \rangle(x, \langle const_x \rangle) \mid \langle comp\_op \rangle(v, \langle const_v \rangle)$	$lt((< \langle const \rangle, input >, \langle const \rangle))$
action	$leaf \mid \langle if \rangle$	$leaf \mid \langle if \rangle$
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	
$const$		[-1, 1) with step 0.001

Table 3.3: 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. "$< \cdot, \cdot >$" denotes the dot product operator.

Rule	Production
start	$\langle if \rangle$
if	$if\ \langle condition \rangle\ then\ \langle action \rangle\ else\ \langle action \rangle$
condition	$lt((< \langle const \rangle, input_i >, 0)$
action	$leaf \mid \langle if \rangle$
$const$	$[-1, 1]$ with step 10^{-3}

Table 3.4: Hyperparameters used to evolve DTs in the CartPole-v1 environment. $\mathcal{Q}_0$ is the interval used for the initialization of the $\mathcal{Q}$ values through random sampling. $\mathcal{U}$ denotes the Uniform distribution.

Param.	**Value (Orthogonal)**	**Value (Oblique)**
n_p	200	200
n_g	100	50
s_i	1024	100
p_x	0	0
p_m	1	1
p_c	0.1	0.1
s_t	2	2
ε	0.05	0.05
$\mathcal{Q}_0$	$\mathcal{U}(-1, 1)$	$\mathcal{U}(-1, 1)$
α	0.001	0.001
n_e	10	10
γ	0.9	0.9

Table 3.5: Hyperparameters used to evolve DTs in the MountainCar-v0 environment. $\mathcal{Q}_0$ is the interval used for the initialization of the $\mathcal{Q}$ values through random sampling. $\mathcal{U}$ denotes the Uniform distribution.

Param.	**Value (Orthogonal)**	**Value (Oblique)**
n_p	200	200
n_g	1000	2000
s_i	1024	100
p_x	0	0.1
p_m	1	1
p_c	0.05	0.05
s_t	2	2
ε	0.05	0.05
$\mathcal{Q}_0$	$\mathcal{U}(-1, 1)$	$\mathcal{U}(-1, 1)$
α	0.001	0.001
n_e	10	10
γ	0.9	0.9

Table 3.6: Hyperparameters used to evolve DTs in the LunarLander-v2 environment. $\mathcal{Q}_0$ is the interval used for the initialization of the $\mathcal{Q}$ values through random sampling. k stands for the k-th sample seen by the agent, v stands for the v-th visit made to a leaf-action pair. "ES" refers to "early stopping".

Param.	**Value (Oblique)**
n_p	100
n_g	100
s_i	100
p_x	0.1
p_m	1
s_t	2
ε_0	1
ε_k	$0.99^k \varepsilon_0$
$\mathcal{Q}_0$	0
α	$1/v$
n_e	1000 with ES
Δ_e	30

3.3 Results

The results obtained in the three environments are shown in Figures 3.2 to 3.7 and in Table 3.7.

CartPole-v1

In the CartPole-v1 environment, both the orthogonal and the oblique grammar reach very high scores, with the oblique one hitting the maximum more often than the orthogonal. However, the oblique grammar is not able to solve the task in all the runs, as shown in Table 3.7.

Moreover, Figure 3.5 shows that the oblique grammar was also able to achieve lower $\mathcal{M}$. This means that, according to the metric $\mathcal{M}$, the oblique DTs obtained should be easier to interpret than the obtained orthogonal DTs.

Figure 3.8 shows the fitness trends of our approach in this environment, with both grammars. It can be observed that the oblique grammar converges faster than the orthogonal one. In fact, in most cases, the maximum fitness reaches the maximum value (i.e., 500) in less than 20 generations.

The difference in performance between the orthogonal and oblique DTs may suggest

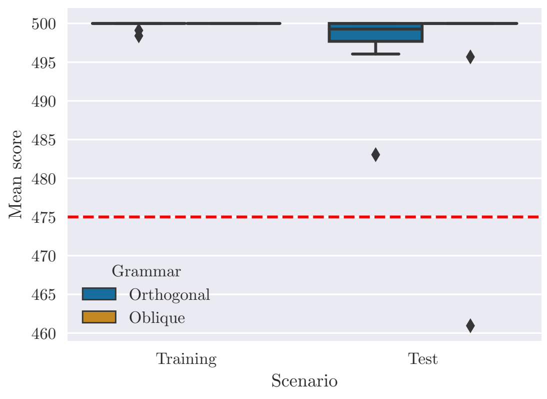


Figure 3.2: Boxplot of training and test scores for the CartPole-v1 environment. The red dashed line denotes the lower bound for solving the task.

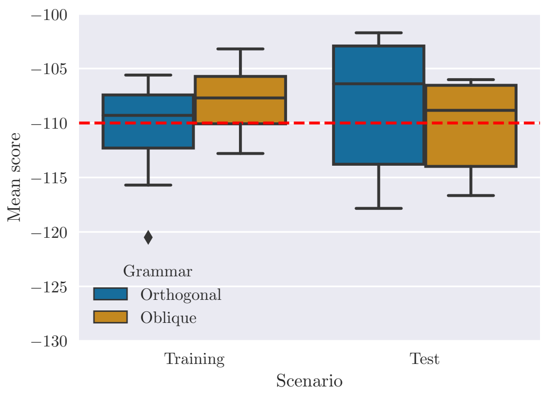


Figure 3.3: Boxplot of training and test scores for the MountainCar-v0 environment. The red dashed line denotes the lower bound for solving the task.

that oblique DTs are inherently more robust than orthogonal DTs in this task. To confirm this hypothesis, we tested the best individuals obtained in each of the 10 runs on a longer version of the CartPole-v1 environment to measure the stability of the solutions (as in Lyapunov stability). The results obtained in this environment, which has a duration of 10^4 steps (and thus the maximum score is 10^4) are shown in Figure 3.11. Moreover, Figure 3.12 shows the distance from the point of equilibrium (i.e, $\mathbf{0}$) at each step of the simulation, using the two best policies obtained by using the orthogonal and the oblique grammar. From the figure, it is clear that the best tree obtained with the oblique grammar remains closer to the point of equilibrium, whereas the orthogonal tree can significantly move away from it, supporting the hypothesis that oblique DTs are more stable than orthogonal DTs in this task.

Furthermore, we tested the robustness of the agents w.r.t. noise on the inputs. Figure 3.13 shows 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 noise with σ in the order of twice the sampling step used for the constants in the grammar. On the other hand, the oblique tree proved to be significantly more robust, being able to cope with noises that have a σ about 50 times bigger than the sampling step used for the constants.

The best DTs obtained in this environment are shown in Figures 3.16 and 3.17.

Finally, in Table 3.8, we compare the best solutions (obtained with this method) with state-of-the-art solutions found in the literature. The complexities computed for the neural-network-based approaches are approximations, i.e., not all the details of the methods are taken into account (only the architecture of the model), resulting in complexities

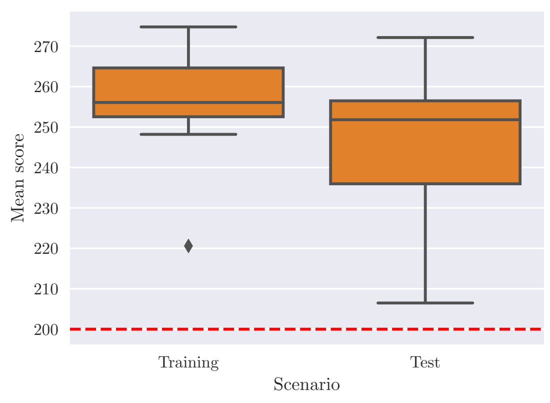


Figure 3.4: Boxplot of training and test scores for the LunarLander-v2 environment. The red dashed line denotes the lower bound for solving the task.

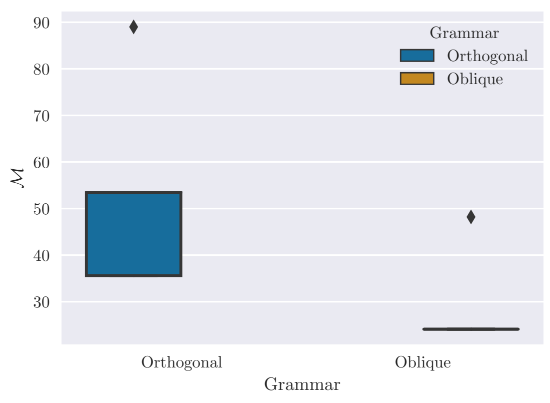


Figure 3.5: Boxplot of the values for $\mathcal{M}$ for the DTs evolved in the CartPole-v1 environment.

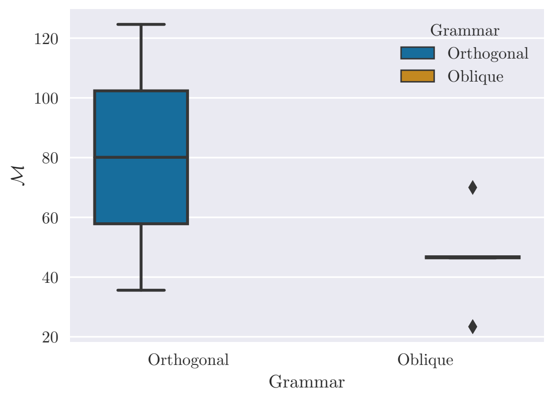


Figure 3.6: Boxplot of the values for $\mathcal{M}$ for the DTs evolved in the MountainCar-v0 environment.

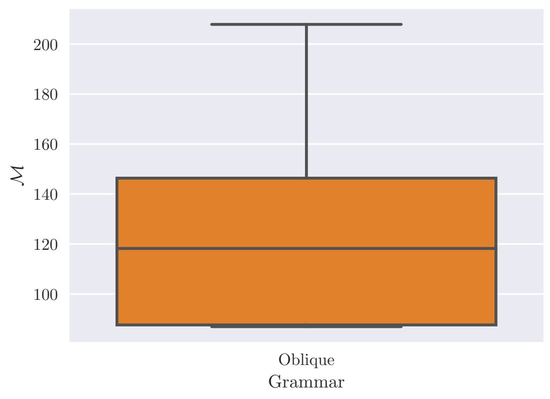


Figure 3.7: Boxplot of the values for $\mathcal{M}$ for the DTs evolved in the LunarLander-v2 environment.

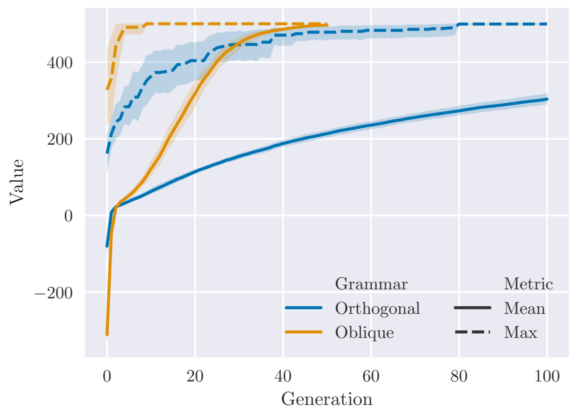


Figure 3.8: Fitness trends for the CartPole-v1 environment. The shaded area represents the 95% confidence interval (computed on 10 runs).

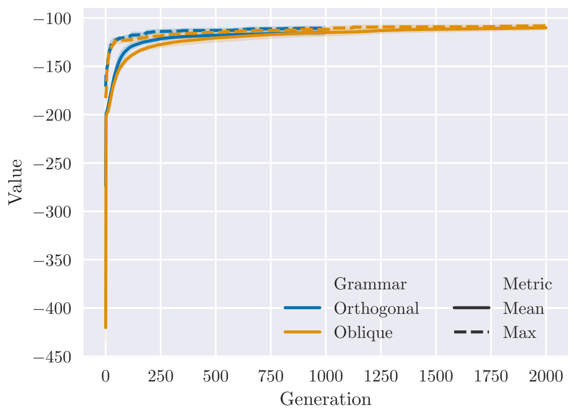


Figure 3.9: Fitness trends for the MountainCar-v0 environment. The shaded area represents the 95% confidence interval (computed on 10 runs).

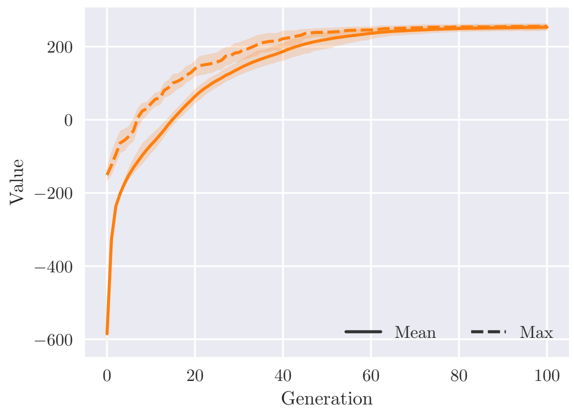


Figure 3.10: Fitness trends for the LunarLander-v2 environment. The shaded area represents the 95% confidence interval (computed on 10 runs).

Table 3.7: Descriptive statistics (mean and std. dev. over the best solutions found in 10 independent runs) on the three OpenAI environments considered in the experimentation. The results obtained with the orthogonal grammar are not shown for the LunarLander-v2 environment since we were not able to find a set of hyperparameters that led to solutions for this task.

Environment	**Type**	**Mean training score**	**Std. dev. training score**	**Mean testing score**	**Std. dev. testing score**	**% of runs solved**
CartPole-v1	Orthogonal	499.75	0.55	497.34	5.19	100%
	Oblique	500.00	0.00	495.66	12.27	90%
MountainCar-v0	Orthogonal	-110.62	4.57	-108.16	6.20	70%
	Oblique	-107.90	3.09	-110.31	4.39	50%
LunarLander-v2	Oblique	255.58	14.58	213.09	18.73	100%

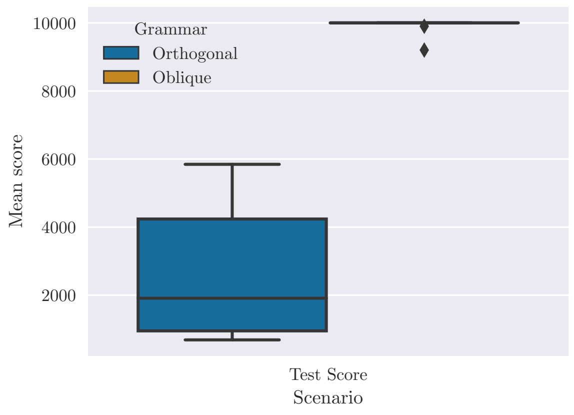


Figure 3.11: Boxplot of the scores obtained in the 10^4-steps-long version of CartPole-v1.

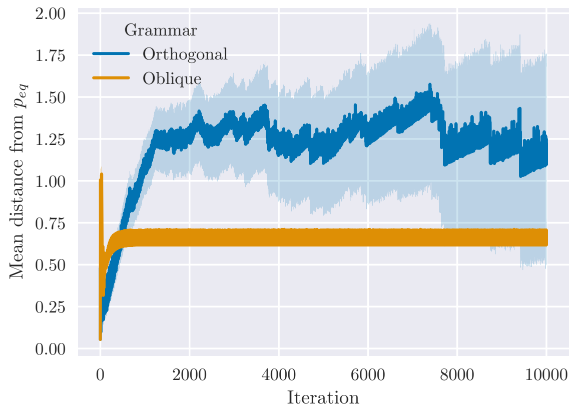


Figure 3.12: Distance of the cart-pole system from the point of equilibrium (i.e., $\mathbf{0}$). The shaded area represents the 95% confidence interval.

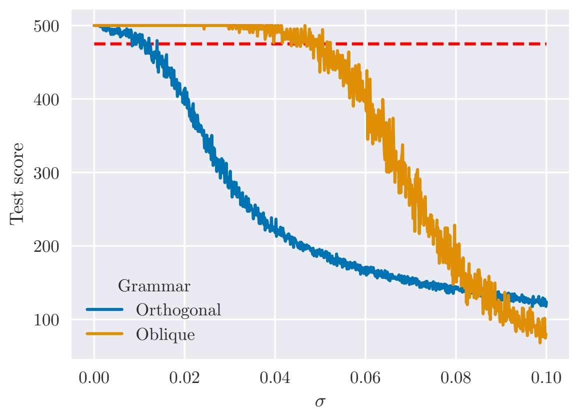


Figure 3.13: Plot of the Test Score w.r.t. different noise levels σ in CartPole-v1.

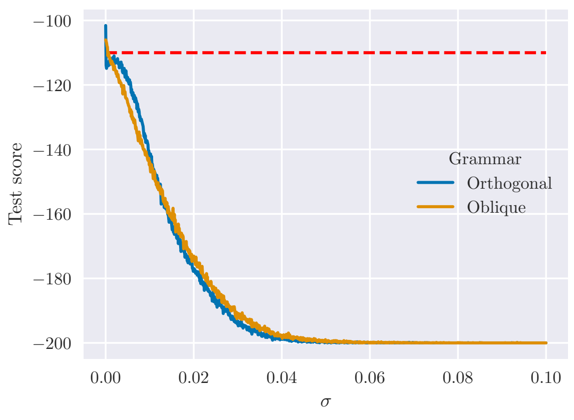


Figure 3.14: Plot of the Test Score w.r.t. different noise levels σ in MountainCar-v0.

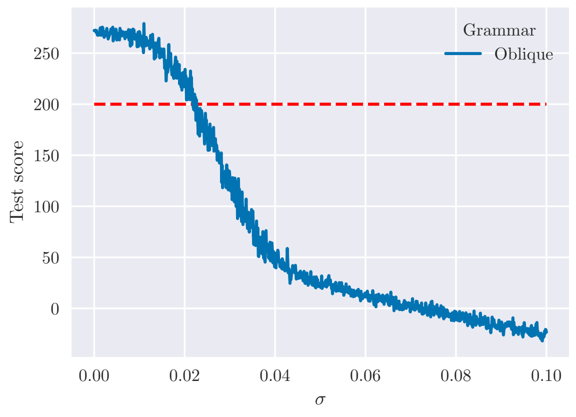


Figure 3.15: Plot of the Test Score w.r.t. different noise levels σ in LunarLander-v2.

that are slightly smaller than the real values. For the purpose of this comparison, these minor differences are negligible. The other solutions are available in the repository of the project[1].

From Table 3.8, it emerges that the proposed method is able to find solutions that match state-of-the-art performance while having a significantly better degree of interpretability.

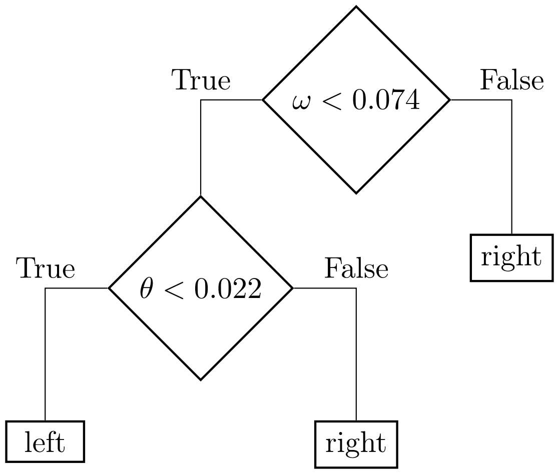


Figure 3.16: Best orthogonal decision tree (w.r.t. score) evolved in the CartPole-v1.

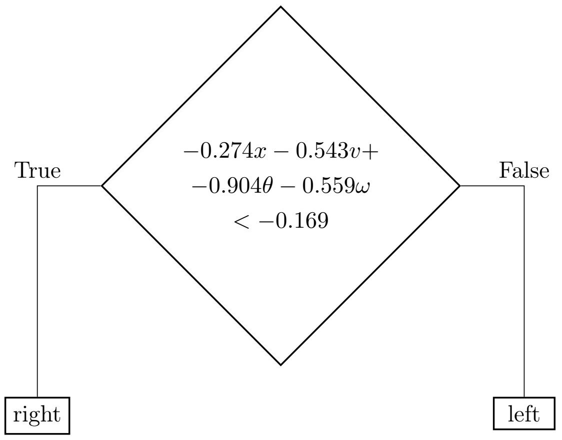


Figure 3.17: Best oblique decision tree (w.r.t. score) evolved in the CartPole-v1 environment.

MountainCar

In this task, the orthogonal grammar performs significantly better, solving the task more often and with better scores than the oblique grammar, as shown in Figure 3.3 and Table 3.7. This suggests that this task is harder to solve with oblique grammars. While this may seem counter-intuitive, since oblique trees are a generalization of orthogonal trees, it may be because the oblique grammar used in these experiments makes it difficult to obtain an orthogonal DT. On the other hand, also in this case, the oblique grammar seems to yield more interpretable DTs according to the metric $\mathcal{M}$.

The fitness trends for the evolutionary process are shown in Figure 3.9.

In Figure 3.14, we compare the robustness to input noise for both versions. In this case, both approaches proved to be not so robust to noise. Surprisingly, it can be observed that the orthogonal tree was not even robust to input noise that had $\sigma < \min_i(step_i)$ where $step_i$ is the sampling step for the constants of the i-th variable.

[1]https://gitlab.com/leocus/ge_q_dts

Table 3.8: Comparison of the solutions obtained by using the proposed approach with respect to the state-of-the-art. The results from [80] are averaged over ten independent runs. The results from [129] regard the discretized tree shown in Figure 3 - right in [129], tested on the same episodes used for the evaluation of our solutions.
(*): Results obtained by using a revised version of the tree obtained through personal communication with the first author of the study. (**): The tree from (*) has been simplified by using the technique used in this work.

Method	**Score**	$\mathcal{M}$
Deep Q Network [80]	327.30	1157.20
Tree-Backup(λ) [80]	494.70	1157.20
Importance-Sampling [80]	498.70	1157.20
Qπ [80]	489.90	1157.20
Retrace(λ) [80]	461.10	1157.20
Watkins's Q(λ) [80]	484.30	1157.20
Peng & Williams's Q(λ) [80]	496.70	1157.20
General Q(λ) [80]	499.90	1157.20
Qualitatively measured policy discrepancy [80]	**500.00**	1157.20
Bayesian Deep Reinforcement Learning [154]	113.52	8090.40
Kronecker-Factored Approximate Curvature [10]	321.00	70786.20
Differentiable Decision Trees [129]	388.76	89.20
Differentiable Decision Trees [129] (*)	**500.00**	106.80
Differentiable Decision Trees [129] (**)	**500.00**	**53.40**
Ours – Orthogonal	**500.00**	**35.60**
Ours – Oblique	**500.00**	**24.10**

Table 3.9 shows a comparison of the results obtained by the best trees obtained (Figures 3.18 and 3.19) w.r.t. the state of the art from the OpenAI Gym Leaderboard[2]. From the table, it can be seen that the best DT outperforms all the approaches from the leaderboard, setting a new state of the art, outperforming even Deep Neural Networks. However, the best DT obtained is not able to reach the lowest $\mathcal{M}$. In fact, the (possibly hand-crafted) closed-form policy from the leaderboard has higher interpretability according to $\mathcal{M}$.

Table 3.9: Comparison of the mean (testing) score of the solutions obtained by using the proposed approach versus the state of the art on MountainCar-v0.

Source	**Method**	**Score**	$\mathcal{M}$
Zhiqing Xiao[3]	Closed-form policy	**-102.61**	**54.7**
Keavnn[4]	Soft Q Networks [68]	-104.58	31079.2
Harshit Singh[5]	Deep Q Network	-108.85	984160.3
Colin M[6]	Double Deep Q Network	-107.83	46681.6
Amit[7]	Tabular SARSA	-105.99	381.5
Dhebar et al. [28]	Nonlinear DT (Open loop)	-128.87	66.8
Ours	Orthogonal DT	**-101.72**	106.80
Ours	Oblique DT	-106.02	**46.80**

LunarLander

Here, the oblique grammar was able to solve the task in 100% of the cases.

The boxplots of the scores and of the complexity of interpretability are shown in Figures 3.4 and 3.7. These figures show that there is a small performance drop when transferring from training to testing episodes. However, in all the cases, the agents' scores are well above the solving threshold.

The average fitness trend for these evolutionary processes is shown in Figure 3.10.

A comparison of the two best solutions (w.r.t score and interpretability) and the state of the art is shown in Table 3.10. Even though these DTs do not achieve (in absolute) the

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

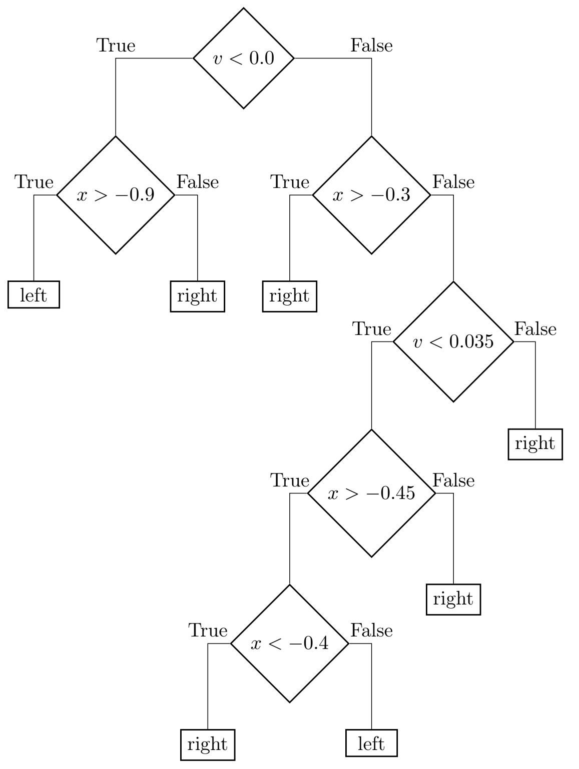
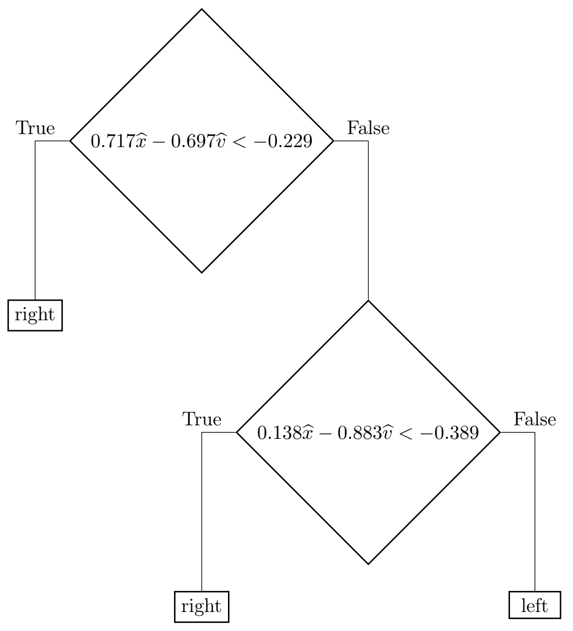


Figure 3.18: Best orthogonal decision tree (w.r.t. score) evolved in the MountainCar-v0 environment.



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

Figure 3.19: Best oblique decision tree (w.r.t. score) evolved in the MountainCar-v0 environment.

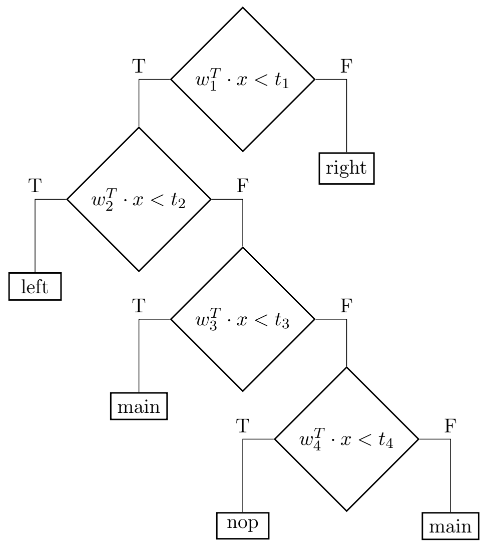


Figure 3.20: Best oblique decision tree (w.r.t. score) evolved in the LunarLander-v2 environment.

best performance and the best $\mathcal{M}$, they represent the best compromise between the two metrics. However, the best solution w.r.t. score 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 3.20

Table 3.10: Comparison of the proposed solution with respect to the state of the art on the LunarLander-v2 environment. The results from [98] are averaged on 5 runs. The results from [153] and [77] are averaged on 10 runs.

Source	**Method**	**Score**	$\mathcal{M}$
Keavnn[8]	Soft Actor Critic	217.92	210733.2
Liu[9]	Soft Q Network	217.09	647691.1
Ash Bellet[10]	Deep Q Network	225.79	1295307.1
Sanket Thakur[11]	Deep Q Network	200.65	259285.8
Mahmood[12]	Deep Q Network	200.3	237079.7
Daniel Barbosa[13]	Proximal Policy Opt.	201.47	1673
XinlyYu[14]	Deep Q Network	**278.23**	518153
Ruslan[15]	Dueling Deep Q N.	200.22	30878.1
Ollie Graham[16]	Deep Q Network	201.46	30878.1
Nikhil Barhate[17]	Actor Critic	254.58	4337.3
Udacity[18]	Deep Q Network	201.46	30878.1
Sigve Rokenes[19]	Deep Q Network	266	$1.21 \cdot 10^8$
Peng et al. [98]	Advantage-weighting	229 ± 2	518153
Xu et al. [153]	Value-difference	248.2 ± 21	632620.2
Malagon et al. [77]	Shallow NN	258.8	**77.6**
Silva et al. [129]	Rule List	-78.4	89
Dhebar et al. [28]	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 ± 18.72	123.34 ± 39

3.3.1 Comparison with other interpretable methods

In this section, we compare the interpretable solutions found in the literature, comparing them to the presented method.

CartPole

Differentiable Decision Trees Silva et al. [129] 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 structure of the tree and the actions taken by using PPO [128] and backpropagation. The DT is shown in 3.21.

Moreover, since the performance of this solution are not satisfactory, by communicating with the first author we were able to obtain an improved version. This solution is shown in Figure 3.22.

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 the best tree found with the method presented in this chapter. In fact, while our best tree has a maximum depth of 2 nodes, their DT has a maximum depth of 3 nodes. This increase in complexity is reflected by the difference in the $\mathcal{M}$ measure.

Since one may hypothesize that this increase in complexity may lead to more robustness, a comparison of the robustness to noise of the trees shown in Figures 3.16 and 3.17

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

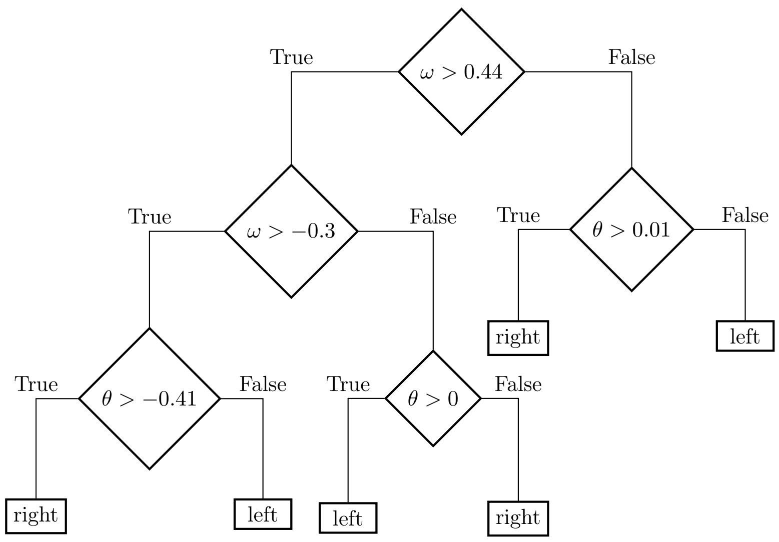


Figure 3.21: DT proposed in [129].

and the two from [129] is shown in Figure 3.26. The figure shows that the orthogonal trees obtained by Silva et al. have a robustness that is comparable to that of the best DTs obtained by means of the approach presented here. This suggests that orthogonal trees may be intrinsically less robust than oblique ones for this task.

MountainCar

Zhiqing Xiao The system proposed by this entry[20] consists of 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 (not considering the solutions proposed in this chapter) on this task while also having the best degree of interpretability (according to our modification of the metric proposed in [149]). The policy is the following:

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

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

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

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

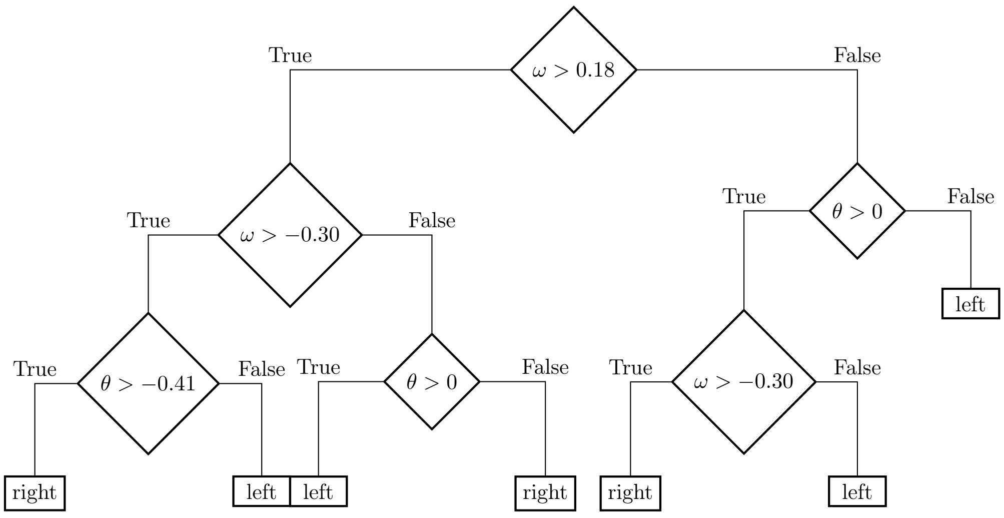


Figure 3.22: DT shared by the first author of [129]

While this policy has a smaller $\mathcal{M}$ than the best DT obtained in this chapter, it may be a bit harder to interpret. We hypothesize that this is due to the fact that the $\mathcal{M}$ metric has been proposed to evaluate the interpretability of general mathematical formulae, while in this case it is applied to the interpretation of *hyperplanes*, whose interpretability may be different. While hyperplanes are defined by mathematical formulae, the interpretability of a hyperplane may also depend on the number of non-linear operations that are used to determine the hyperplane.

Amit This entry[21] 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 3.9 shows that, even if this approach is transparent and easily interpretable, its interpretability is higher than the DTs obtained in this chapter. 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 an impact on the policy, discarding irrelevant details.

Dhebar et al. Dhebar et al., in [28], propose an approach to IRL 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.

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

In this case, we only had access to the open-loop policy for the MountainCar-v0 environment, which is shown in Figure 3.23.

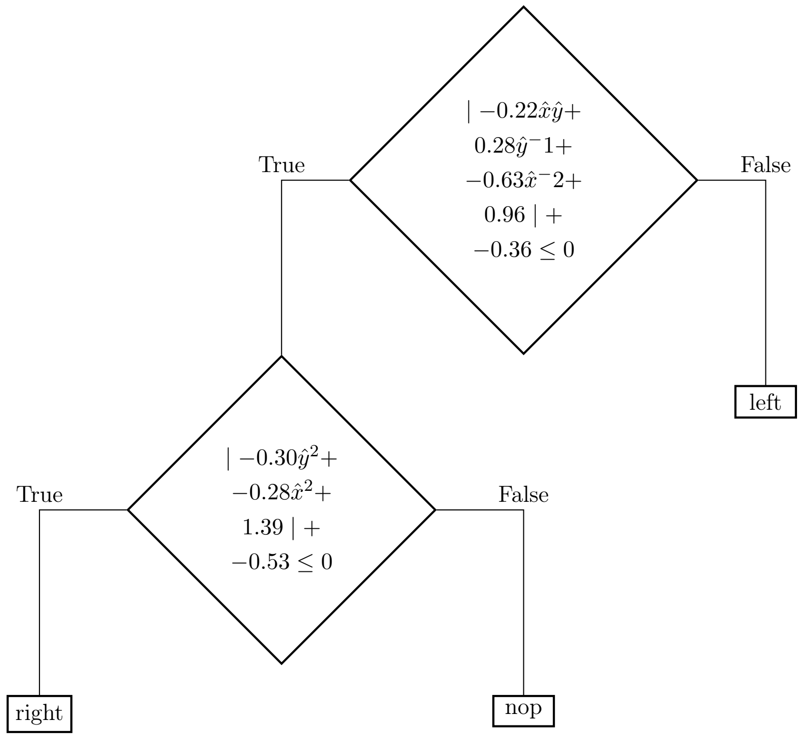


Figure 3.23: Tree representation of the solution proposed in [28] for the MountainCar-v0 environment. The variables with a hat are normalized by using this way: $1 + \frac{x - x_{min}}{x_{max} - x_{min}}$.

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 nonlinearity 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 [28].

Finally, we perform a comparison of the robustness to input noise with the solutions provided by “Zhiqing Xiao” and the one provided by Dhebar et al. Figure 3.27 shows how performance varies 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.

LunarLander-v2

Silva et al. In [129] 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 3.24 shows the solution obtained. However, as shown in Table 3.10, it does not achieve satisfactory performance. This is due to the fact that, while the differentiable tree is able to achieve good performance (even though not solving the task), its discretization modifies the final distribution of the actions.

Malagon et al. In [77] the authors use the Univariate Marginal Distribution Algorithm to evolve a neural network without hidden layers in the LunarLander-v2 domain. 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 [66] and the metric $\mathcal{M}$.

Dhebar et al. In [28] 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, and we only had access to the 3-level-deep NLDT.

The tree is shown in Figure 3.25. 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, shown in Figure 3.28. However, for this comparison, we could not include the results from Malagon et al. since the weights were not publicly accessible.

3.3.2 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 DTs with fixed leaves (also optimized by GE). Moreover, we perform statistical tests to test whether the differences are statistically significant by fixing a threshold for the p-value of $\alpha = 0.05$. To perform these experiments, we used the same parameters shown in Tables 3.4 to 3.6. The numerical results are shown in Table 3.11

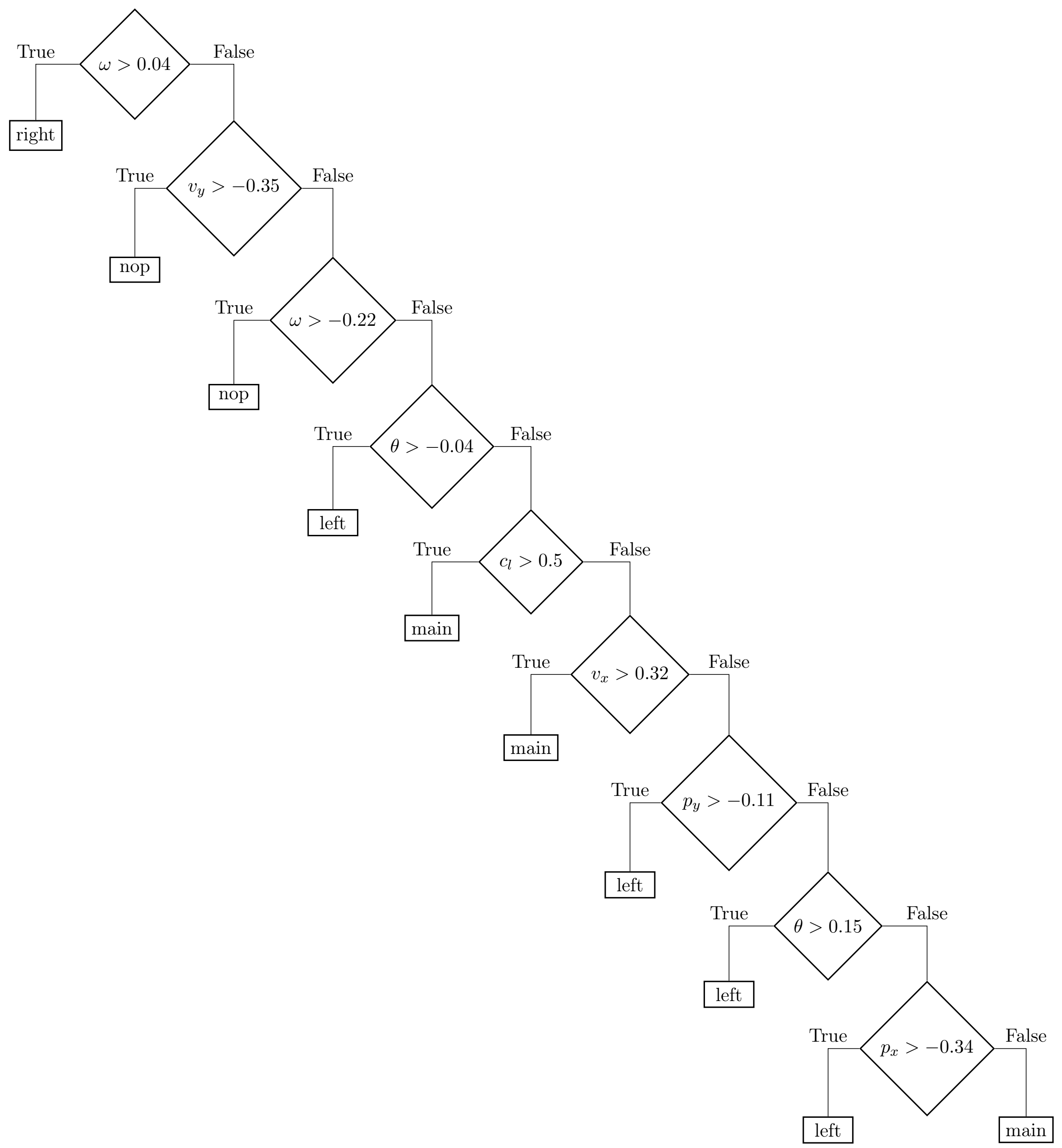


Figure 3.24: Tree representation of the solution proposed in [129] for the LunarLander-v2 environment.

CartPole

For the orthogonal case, we observe that while in most cases evolution is able to evolve agents that achieve a perfect training score, they have poor generalization capabilities. In our opinion, this is akin to overfitting. In fact, in this case, the agents did not understand

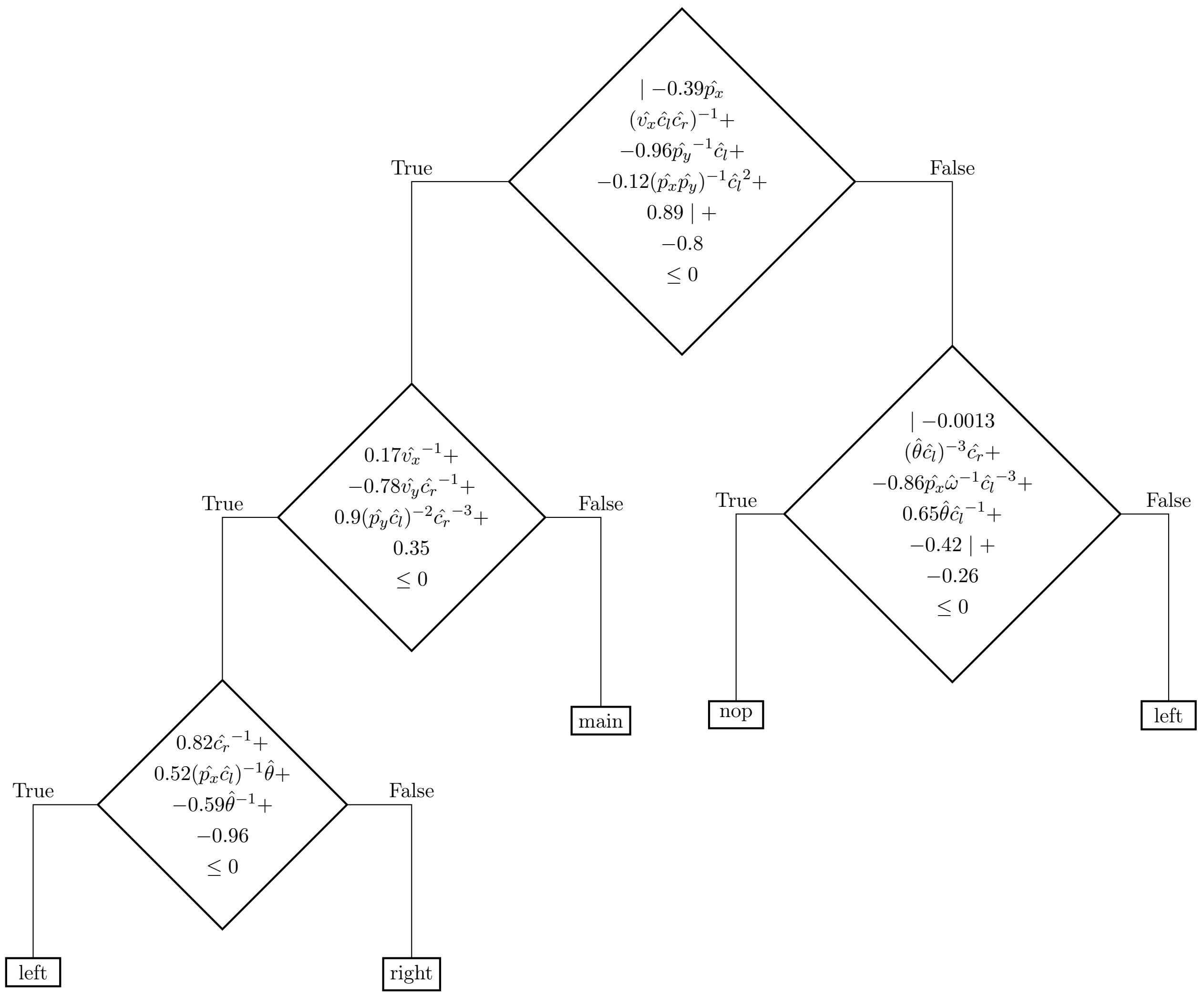

$| -0.39\hat{p_x}$
$(\hat{v_x}\hat{c_l}\hat{c_r})^{-1}+$
$-0.96\hat{p_y}^{-1}\hat{c_l}+$
$-0.12(\hat{p_x}\hat{p_y})^{-1}\hat{c_l}^2+$
$0.89 | +$
-0.8
≤ 0
True
False
$0.17\hat{v_x}^{-1}+$
$-0.78\hat{v_y}\hat{c_r}^{-1}+$
$0.9(\hat{p_y}\hat{c_l})^{-2}\hat{c_r}^{-3}+$
0.35
≤ 0
True
False
main
$| -0.0013$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

Figure 3.25: Tree representation of the solution proposed in [28] for the LunarLander-v2 environment. The variables with a hat are normalized in the following way: $1 + \frac{x - x_{min}}{x_{max} - x_{min}}$.

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 comes from the same distribution) with threshold $\alpha = 0.05$.

For the oblique grammar, instead, the results seem quite similar to those obtained when using $\mathcal{Q}$-Learning. In this case, we get a p-value of 0.73. Thus, we cannot reject the null hypothesis with threshold $\alpha = 0.05$. For this reason, we will assume that they come from the same distribution.

This suggests that, since oblique trees seem to be both more robust to noise and more stable than orthogonal trees, good oblique policies for this environment can be obtained

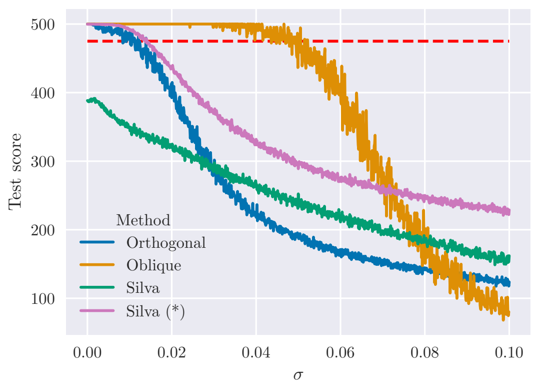


Figure 3.26: Comparison of the robustness w.r.t. the solutions from [129]. “Silva (*)” refers to the DT shown in Figure 3.22.

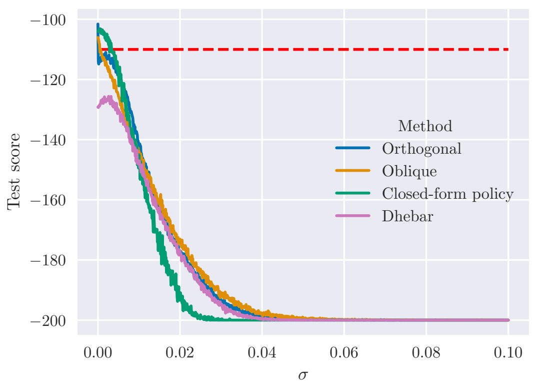


Figure 3.27: Comparison of the robustness to input noise between our solutions and the interpretable ones on the MountainCar-v0 environment.

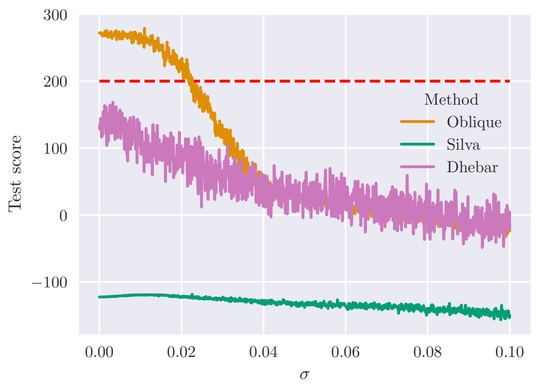


Figure 3.28: Comparison of the robustness to input noise w.r.t. other interpretable solutions on the LunarLander-v2 environment.

Table 3.11: Descriptive statistics (mean and std. dev. over the best solutions found in 10 independent runs) on the three OpenAI environments considered in the experimentation using only Grammatical Evolution, without $\mathcal{Q}$-Learning.

Environment	**Type**	**Mean training score**	**Std. dev. training score**	**Mean testing score**	**Std. dev. testing score**	**% of runs solved**
CartPole-v1	Orthogonal	500.00	0.00	470.94	26.98	50.00%
	Oblique	500.00	0.00	497.73	7.18	100.00%
MountainCar-v0	Orthogonal	-109.36	7.80	-112.67	7.24	50.00%
	Oblique	-101.23	2.51	-108.35	1.75	80.00%
LunarLander-v2	Oblique	175.02	125.71	147.20	131.15	60.00%

even without $\mathcal{Q}$-learning.

MountainCar

In both the orthogonal and the oblique cases, performance seems to be similar to that of the combination of GE and $\mathcal{Q}$-Learning. A Two-tailed Mann-Whitney U-Test confirms that in both cases we cannot reject the null hypothesis.

Thus, in this environment, it seems that $\mathcal{Q}$-Learning does not add any value. However, this may also be due to the fact that n_e is quite low for this environment, and it is pretty hard to explore. Hence, with higher n_e, the conclusions may be different.

LunarLander

We expect that in this task, since it is harder than the previous two (and n_e is higher), GE performs worse than our approach.

According to our expectations, this approach can solve the task only in 60% of the cases. Moreover, the hypothesis is confirmed by a two-tailed Mann-Whitney U-Test, which 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.

3.3.3 Interpretation of the solutions

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

CartPole

Orthogonal tree The tree shown in Figure 3.16 is extremely easy to interpret. In fact, this agent moves the cart to the left if

$$\omega < 0.074 \wedge \theta < 0.022 \tag{3.5}$$

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 “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:

$$-0.274x_k - 0.543v_k - 0.904\theta_k - 0.559\omega_k < -0.169 \tag{3.6}$$

where k refers to the current timestep. To simplify the process, we write Equation 3.6 as the following:

$$-ax_k - bv_k - c\theta_k - d\omega_k < t \tag{3.7}$$

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:

$$x_n \approx -\frac{t}{a} \tag{3.8}$$

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 3.7 as follows:

$$-x_k - \frac{b}{a}v_k - \frac{c}{a}\theta_k - \frac{d}{a}\omega_k < \frac{t}{a} = -x_n \tag{3.9}$$

We can then perform other steps and obtain:

$$-x_k - b'v_k - c'\theta_k - d'\omega_k < -x_n \Rightarrow \tag{3.10}$$

$$-b'v_k - c'\theta_k - d'\omega_k < -x_n + x_k \Rightarrow \tag{3.11}$$

$$-b'v_k - c'\theta_k - d'\omega_k < -x_n + x_{n-1} - x_{n-1} + ... + x_k = \sum_{j=n}^{k+1} -x_j + x_{j-1} \tag{3.12}$$

Then, by noting that

$$\frac{x_k - x_{k-1}}{\tau} = v_k \tag{3.13}$$

we can rewrite Equation 3.12 as:

$$-b'v_k - c'\theta_k - d'\omega_k < -\sum_{j=k+1}^{n} v_j\tau \tag{3.14}$$

$$-c'\theta_k - d'\omega_k < -\sum_{j=k}^{n} g_j v_j\tau \tag{3.15}$$

where

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

Now, by observing that

$$\frac{\theta_k - \theta_{k-1}}{\tau} = \omega \tag{3.16}$$

we obtain

$$-c'\theta_k - d'\frac{\theta_k - \theta_{k-1}}{\tau} < -\sum_{j=k+1}^{n} g_j v_j\tau \tag{3.17}$$

$$-(d' + \tau c')\theta_k + d'\theta_{k-1} < -\tau^2 \sum_{j=k+1}^{n} g_j v_j \tag{3.18}$$

Finally, note 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 velocities become small ($\underset{k}{max} \mid v_k \mid < 0.55$) because the pole is balanced, we can write that:

$$\mid \sum_{j=k+1}^{n} g_j v_j \mid \lessapprox \frac{b'}{\tau} \cdot 1.5 + 49 \cdot 1.5 + 450 \cdot 0.55 = 420 \tag{3.19}$$

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

Then, we obtain

$$-(d' + \tau c')\theta_k + d'\theta_{k-1} < \tau b' v_k \tag{3.20}$$

$$c\theta_k > -(bv_k + d\omega_k) \tag{3.21}$$

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[22]:

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

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.

MountainCar

Orthogonal tree The orthogonal tree (Figure 3.18) is easy to interpret. In fact, if we look at the leaves, we see that the agent accelerates to the left only in two cases: $(v < 0 \land x > -0.9) \lor (v \in [0, 0.035) \land x \in [-0.4, -0.3])$. This means that the agent 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 $v \leq 7.5799 \cdot 10^{-2} \cdot x + 6.6955 \land v \leq 1.1516 \cdot 10^{-2} \cdot x + 5.495 \cdot 10^{-3}$. This corresponds to the decision regions shown in Figure 3.29.

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.

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

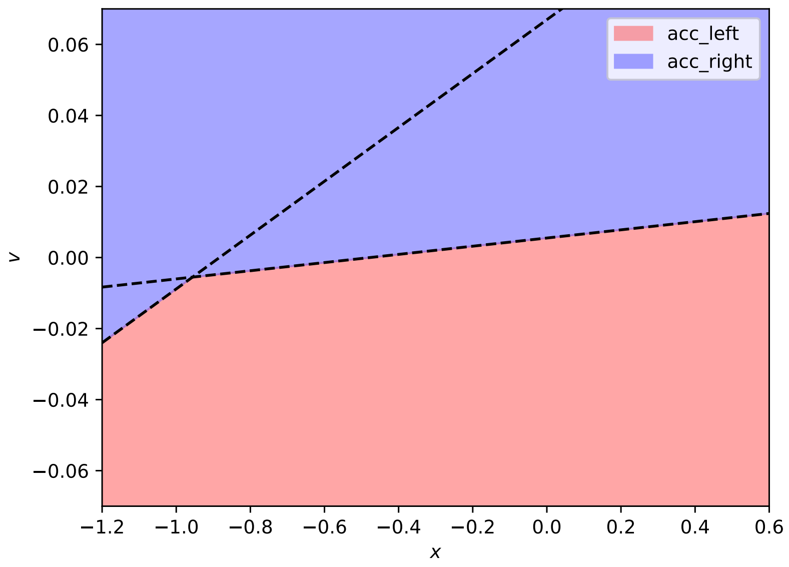


Figure 3.29: Decision regions for the best oblique tree evolved in the MountainCar-v0 environment.

LunarLander

In this case, since the oblique tree (Figure 3.20) has 4 conditions and 8 unknowns, it is a bit harder to interpret.

The weights $w_1, \ldots, w_4$ are the following:

$$w_1 = 0.401, -0.104, +0.495, -0.055, -0.69, -0.845, -0.2, -0.597$$
$$w_2 = 0.448, -0.366, +0.431, -0.462, -0.693, -0.821, +0.461, -0.132$$
$$w_3 = -0.101, +0.133, -0.791, +0.653, -0.207, +0.731, +0.068, +0.525$$
$$w_4 = 0.12, -0.044, -0.772, -0.136, -0.169, +0.821, -0.573, -0.251$$

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

$$ap_x - bp_y + cv_x - dv_y - e\theta - f\omega - gc_l - hc_r \geq 0 \tag{3.22}$$

where $a, b, ..., h$ replace the constants of w_1

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 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 3.22 as follows:

$$ap_x + cv_x - bp_y - dv_y - e\theta - f\omega \geq 0 \tag{3.23}$$

By merging some terms we obtain:

$$a(p_x + v_x c') - b(p_y - v_y d') - e(\theta - \omega f') \geq 0 \tag{3.24}$$

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

$$ap_x^{k+1} - bp_y^{k+1} \geq e\theta^{k+1} \tag{3.25}$$

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 for 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:

$$ap_x - bp_y + cv_x - dv_y - e\theta - f\omega < 0 \tag{3.26}$$

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

$$a(p_x + v_x c') - b(p_y + v_y d') - e(\theta + \omega f') < 0 \tag{3.27}$$

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

$$ap_x^{k+1} - bp_y^{k+1} < e\theta^{k+1} \tag{3.28}$$

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:

$$-ap_x + bp_y - cv_x + dv_y - e\theta + f\omega < 0 \tag{3.29}$$

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

$$-a(p_x + v_x) + b(p_y + v_y) - (c - a)v_x + (d - b)v_y - e\theta + f\omega < 0 \tag{3.30}$$

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:

$$-ap_x^{k+1} + bp_y^{k+1} - c'v_x + d'v_y - e\theta + f\frac{\theta - \theta^{k-1}}{\tau} < 0 \tag{3.31}$$

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

$$-\tau ap_x^{k+1} + \tau bp_y^{k+1} - \tau c'v_x + \tau d'v_y - \tau e\theta + f(\theta - \theta^{k-1}) < 0 \tag{3.32}$$

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:

$$-\tau c'v_x + \tau d'v_y + (f - \tau e)\theta - f\theta^{k-1} < 0 \tag{3.33}$$

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

$$-c'v_x + d'v_y + (f - \tau e)\omega + \tau e\theta^{k-1} < 0 \tag{3.34}$$

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

$$\omega < \frac{1}{f - \tau e}(c'v_x - d'v_y - \tau e\theta^{k-1}) \tag{3.35}$$

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

$$\omega < c''v_x \tag{3.36}$$

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):

$$ap_x - bp_y - cv_x - dv_y - e\theta + f\omega < 0 \tag{3.37}$$

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

$$\omega < \frac{c}{f}v_x + \frac{d}{f}v_y \tag{3.38}$$

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

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

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 of 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.

While some solutions may seem hard to interpret (i.e., oblique decision trees), it is important to see 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.

Moreover, it's important to note that the simplification mechanism simplifies the interpretation as it allows removing unnecessary nodes, significantly reducing the size of the trees.

Finally, it is important to note that, while the approach presented in this paper is competitive w.r.t. the state of the art on scores, its computational cost (in terms of episodes) is often significantly higher.

3.4 Conclusions

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

The results show that the proposed approach is able to generate decision trees that are comparable to or even better than the non-interpretable state-of-the-art (from the performance point of view) while having significantly better interpretability.

Moreover, we compare the solutions obtained to the state-of-the-art from the point of view of 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., adapted to machine learning models).

Since it is important for practical applications, we also compare 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. 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., adapted to machine learning models). In fact, while we found that this metric is good ad differentiating between huge and small models, we also saw that it is not able to estimate interpretability with a fine granularity. For instance, as shown in Section 3.3, this metric tends to say that a shallow cascade of small, intuitive non-arithmetical operations such as orthogonal splits are less interpretable than fewer larger combinations of non-linear splits, each of which describes a complex hyperplane. Finally, we demonstrate that the produced agents can be interpreted, practically showing the advantage of interpretable models w.r.t. black boxes.

Chapter 4

Interpretable Reinforcement Learning with continuous action spaces

Based on: Custode, Leonardo Lucio, and Giovanni Iacca. “A co-evolutionary approach to interpretable reinforcement learning in environments with continuous action spaces.” 2021 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, 2021.

4.1 Introduction

While several approaches have been proposed to perform RL with interpretable models in discrete action spaces, how to use them in the case of continuous action spaces is still an open question. In fact, to our knowledge, none of the interpretable methodologies has been applied to RL problems with continuous action spaces. The only exception is represented by the approach presented in [129], which does allow to perform RL with continuous actions, but it has a drawback. In fact, since it uses differentiable decision trees, these need to be transformed into “traditional” decision trees in order to be interpreted properly. To this end, the trees must be transformed by converting the “soft” splits (i.e., sigmoids) into “hard” splits (i.e., binary conditions). This conversion process, as shown in their paper, may introduce a significant loss in performance.

In this chapter, we extend the approach proposed in Chapter 3 by allowing it to produce agents that can act in continuous action spaces. Here, we address the problem of “choosing” a value from a continuous action space by dividing the problem into two parts. The first one is identical to Chapter 3, and consists in obtaining a tree that has to choose an action from a discrete action space. The second part consists in “translating” the discrete action to a continuous one. We do so by adopting a cooperative co-evolutionary approach [103]. Consequently, we evolve *simultaneously* two different populations: a population of decision trees and a population of sets of actions. We evolve the action sets with an estimation of distribution algorithm, which allows us to perform continuous optimization.

The rest of the chapter is structured as follows. The next section makes a summary of the background of this paper. Section 4.3 describes the proposed method. In Section 4.4 we will present the experimental results and, finally, in Section 4.5 we will draw the conclusions of this chapter.

4.2 Background

Our method uses the following methodologies: cooperative co-evolution, grammatical evolution, Q-Learning, and the UMDA$_c^G$ estimation of distribution algorithm. We are not going to describe again grammatical evolution and Q-Learning as they have already been presented in Chapter 3.

4.2.1 Cooperative co-evolution

Cooperative co-evolution [103] is an evolutionary approach to function optimization that allows for the decomposition of the structure of the problem into sub-problems. It works by combining multiple evolutionary processes, one for each sub-component of the problem. In a cooperative co-evolutionary setting, all the populations adapt to each other to optimize the fitness of the whole solution. Of note, each evolutionary process concurs in the optimization of the fitness of the whole solution without any “knowledge” about the other evolutionary processes, i.e., all the populations evolve *independently*.

4.2.2 $UMDA_c^G$

The univariate marginal distribution algorithm for continuous domains [63] (UMDA$_c$) is an estimation of distribution algorithm [64] (EDA). It is the continuous extension of the UMDA algorithm [88]. This algorithm uses the concept of rank-based evolutionary algorithms, i.e., evolutionary algorithms in which the fitness is not meaningful when used alone. Instead, it is helpful to *sort* the individuals by performance.

The algorithm works as follows. At each generation, n individuals are sampled from the current joint probability density distribution $f(\boldsymbol{x};\boldsymbol{\theta}) = \prod_{i=1}^{N} f_i(x_i;\theta_i)$, where N is the number of parameters to optimize. Then, the best k individuals are selected to produce the following generation. The distribution for the next generation of individuals is computed as follows. First, a hypothesis test is performed to get the best-fitting density distribution, and then a maximum likelihood estimate is used to compute the parameters of said distribution.

By constraining the distributions to be normal, the hypothesis testing phase can be avoided and the distribution's parameters can be directly computed as:

$$\hat{\mu_i} = \frac{1}{k}\sum_{i=1}^{k} x_i;\ \hat{\sigma_i} = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(x_i - \hat{\mu_i})^2}$$

This variant is called UMDA$_c^G$.

4.3 Method

4.3.1 Discrete-action RL with decision trees

In Chapter 3, we presented a method to perform discrete-action RL with decision trees, which combines GE and Q-learning. Our method evolves the inner structure of the decision tree by using GE and then, in the fitness evaluation phase, trains the leaves of the tree by means of Q-learning. This approach uses GE to search for a "space-decomposition" function, which decomposes the state space of the problem into groups of *semantically* similar states; while it uses Q-learning to search for state-action functions, which map (groups of) states to actions.

4.3.2 Continuous-action RL with decision trees

In order to obtain agents that can cope with continuous-action RL problems, here we use a co-evolutionary approach similar to the one proposed in [103]. Our approach consists in evolving two distinct populations. The first population is composed of decision trees. The second population, instead, contains sets of continuous actions that the agent will use. These two populations are needed to evolve the two components of the agent shown in Fig. 4.1.

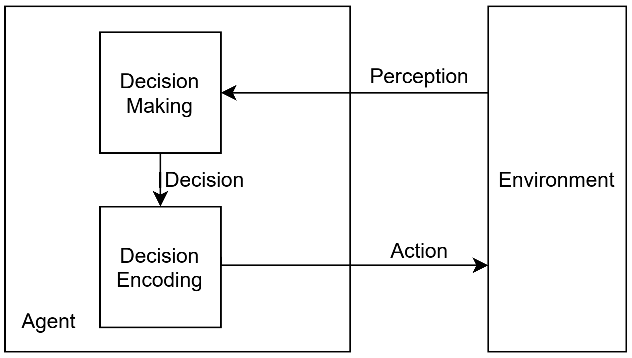


Figure 4.1: Graphical representation of the agents being evolved.

The decision trees encode discrete policies (as done in Chapter 3) that correspond to the "Decision Making" module of Fig. 4.1. Furthermore, each set of actions contains $n_a \cdot s_a$ floating-point numbers, where n_a corresponds to the size of the "pool" of actions and s_a is the dimensionality of each action (e.g., the number of motors that the agent must control). The translation from discrete action to continuous action is performed in the "Decision Encoding" module of Fig. 4.1, where the discrete decision computed by the "Decision Making" module is used as the index for the evolved list of actions.

In Fig. 4.2, the evaluation phase for the two populations is presented. As we can see, we perform a pre-defined number of random pairings n_{pairs} between trees and action sets. Subsequently, each pair is evaluated through the fitness function. After the evaluation phase, each individual's fitness (in both populations) is set to the mean score obtained by all the pairs that contained such individual. If an individual does not appear in the first n_{pairs} pairs, an extra pair is added by selecting a random individual from the other population. The pseudo-code for the evaluation phase is shown in Algorithm 4. Note that, to implement co-evolution, we use an *ask-tell* implementation for both GE and UMDA$_c^G$, i.e., an implementation that returns the whole population when the *ask* method is called and updates the population when the *tell* method (which takes as input the fitness values) is called.

To evolve the population of decision trees, we use the same approach presented in Chapter 3: in particular, we use a grammar that allows us to encode decision trees as a series of `if-then-else` in Python, and Grammatical Evolution to evolve such trees. When evaluating the decision tree, the action for each leaf is not fixed. Instead, the leaves perform Q-learning to learn the proper mapping between states and actions.

On the other hand, to evolve the sets of actions, a continuous optimization algorithm is used. Even though this method is general and can be used with other optimization algorithms, in our experiments we use the UMDA$_c^G$ algorithm [64]. The advantage given by UMDA$_c^G$ over other methods relies, in this case, on its selection mechanism: since at each generation the subsequent distribution depends on the best individuals of the current generation, in this co-evolutionary setting the population seems to *stagnate* faster than other methods, e.g., CMA-ES [45]. While a quick stagnation is usually a problem, in this case it may be necessary. In fact, while in the initial phase it is good to have an exploration of the action space, in the next generations it might be useful to keep the actions fixed (i.e., the action-sets do not vary significantly) so that the evolutionary process that evolves the decision trees can "rely" on these actions. In other words, a small mutation in the genotype of a decision tree will not cause a destructive change in its behavior, thus preserving a "locality" principle for the mapping between genotypes and phenotypes.

Algorithm 4: Pseudo-code for the evaluation of the two populations.

```
   input: ge: ask-tell implementation of GE;
          umda: ask-tell implementation of $UMDA_c^G$;
          $n_{pairs}$: minimum number of pairs to evaluate;
   // Create pairs of individuals from the two populations
1  trees ← ge.ask() ;                                  // get current population
2  actions ← umda.ask() ;                                     // get current pop.
3  pairs ← make_pairs(trees, actions, $n_{pairs}$);
   // Evaluate the fitness of the pairs
   // Q-learning is performed in the fitness evaluation function
4  fitnesses ← compute_fitnesses(pairs);
   // Compute the fitness of each individual
5  tree_fitnesses ← empty_list(trees.size);
6  act_fitnesses ← empty_list(actions.size);
7  foreach tree ∈ trees do
8      indices ← where_is(tree, pairs);
9      t_index ← get_index(tree, trees);
10     tree_fitnesses[t_index] ← mean(fitnesses[indices]);
11 end
12 foreach set ∈ actions do
13     indices ← where_is(set, pairs);
14     a_index ← get_index(set, actions);
15     act_fitnesses[a_index] ← mean(fitnesses[indices]);
16 end
17 ge.tell(tree_fitnesses);
18 umda.tell(tree_fitnesses);
```

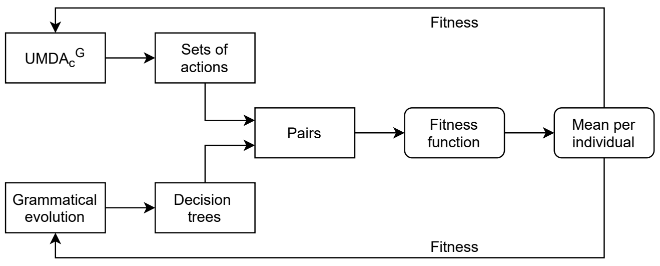


Figure 4.2: Evaluation phase for the two populations

4.3.3 Experimental setup

To validate the effectiveness of the proposed approach, we test it in two well-known continuous-action reinforcement learning benchmarks available in the OpenAI Gym framework [12]:

- MountainCarContinuous-v0,
- LunarLanderContinuous-v2.

Both environments are the continuous-action counterpart of the environments used in Chapter 3. In MountainCarContinuous-v0, the agent has to choose an acceleration value (in $[-1, 1]$) at each step. The agent receives a penalty for accelerating, which is proportional to the magnitude of the acceleration; and a reward of 100 for reaching the target position. The environment is considered solved when the agent receives a mean return higher than 90 points over 100 episodes.

In LunarLanderContinuous-v2, instead, the agent has to compute two different values at each step (both in $[-1, 1]$), where the first value controls the main engine and the second one controls a side engine that can push either to the left or to the right. The penalties, also in this case, are proportional to the magnitude of the horizontal and vertical accelerations, and the rewards are equal to the ones for LunarLander-v2. The environment is considered solved when the mean return is higher than 200 points.

For the MountainCarContinuous-v0 environment, we use a grammar that can generate orthogonal decision trees (i.e., each hyperplane is orthogonal to the axis of the variable used in the condition), shown in Table 4.1. On the other hand, for the LunarLanderContinuous-v2 environment, we use a grammar that can generate oblique decision trees (i.e., linear combinations of the variables used as conditions), also shown in Table 4.1. The reason behind the use of oblique decision trees for the LunarLander-v2 environment relies on the results shown in Chapter 3 and [129], which suggest that solving this task by using orthogonal decision trees may be hard.

The parameters for the Grammatical Evolution, Q-learning, and UMDA$_c^G$ algorithms (equal for all the environments) are shown in Table 4.2.

We set $n_{pairs} = 200$. For the UMDA$_c^G$ algorithm, we used as population size a number of individuals such that each tree is evaluated once, while each set of actions is evaluated more than once. The rationale behind this choice is the following: apart from the initial generations, the sets of actions will slightly differ from each other. So, if the policy encoded by the tree is effective, it should work well even with a non-perfect set of actions. For this reason, we can evaluate each tree once, while we have to evaluate each set of actions more than once. This trick allows us to assess the goodness of the set of actions in several

Table 4.1: Grammar used for the production of decision trees. The different choices for each rule are delimited by the "|" symbol. n denotes the number of input variables given by the environment. The "lt" operator stands for "less than".

Environment	**Rule**	**Production**
MountainCar	root	$\langle if \rangle$
	if	$if\ \langle split \rangle\ then\ \langle action \rangle\ else\ \langle action \rangle$
	split	$lt(var_i, \langle const_i \rangle),\ i \in [0, n]$
	action	$leaf \mid \langle if \rangle$
	$const_0$	$x \in [-1.2, 0.6]$ with step 0.1
	$const_1$	$x \in [-0.07, 0.07]$ with step 0.01
LunarLander	root	$\langle if \rangle$
	if	$if\ \langle split \rangle\ then\ \langle action \rangle\ else\ \langle action \rangle$
	split	$lt(\sum_{i=0}^{n} var_i \cdot \langle const \rangle, 0),\ i \in [0, n]$
	action	$leaf \mid \langle if \rangle$
	const	$x \in [-1, 1]$ with step 10^{-3}

policies. So, having a number of action sets much smaller than the number of trees allows us to evolve the second population (i.e., the action sets) almost "for free", meaning that the number of individuals evaluated is approximately equal to the number of individuals evaluated in the discrete-action case Chapter 3.

To statistically assess the repeatability of our results, we perform 10 independent runs for each setting[1]. We choose to perform 10 runs because, given the results shown in the next subsection, these yield a relative error on the empirical mean of $r = 0.006$ for the MountainCarContinuous-v0 environment and $r = 0.019$ for the LunarLander-v2 environment, which (given the characteristics of these two tasks) is acceptable to evaluate if they are solved or not. These uncertainties have been computed by using the method proposed in [20].

All the experiments have been performed on an HPC, using 4GB of RAM and 20 cores per experiment (with parallel fitness evaluations) with a time limit of 6 hours.

4.4 Results

In Table 4.3, we show the descriptive statistics of the obtained trees, in terms of both score and complexity. The thresholds for solving the two tasks are 90 and 200 for the MountainCarContinuous-v0 and LunarLanderContinuous-v2 environments, respectively.

[1]`https://gitlab.com/leocus/ge_q_dts_carl`

Table 4.2: Hyperparameters for the algorithms in all the environments. $\mathcal{N}$ denotes the normal distribution and $\mathcal{U}$ denotes the uniform distribution.

Algorithm	**Parameter**	**Value**
Grammatical Evolution	Population size	200
	Generations	40
	Time limit	6 h
	Genotype length	100
	Selection	Tournament selection
	Tournament size	3
	Crossover probability	0
	Mutation probability	1
	p_{gene}	0.05 (MountainCarContinuous-v0) 0.1 (LunarLanderContinuous-v2)
Q-learning	Algorithm	ε-greedy with ε-decay
	ε_0	1
	Decay multiplier	0.99
	Learning rate	$1/k$, k: leaf visits counter.
	Number of episodes	1000 with early stopping
	Early stopping period	30
	Initialization	$\mathcal{U}(-1, 1)$ (MountainCarContinuous-v0) 0 (LunarLanderContinuous-v2)
UMDA$_c^G$	Initialization strategy	$\mathcal{N}(0, 1)$
	Population size	100
	Selection size	10
	Number of actions	8

Table 4.3: Scores of the best agents evolved in the two settings over 10 independent evolutionary runs in 100 episodes.

Metric	**Environment**	**Mean**	**Std**	**Best**	**Worst**
Score	MountainCar	98.12	0.78	98.77	96.41
	LunarLander	239.34	6.40	252.54	231.21
$\mathcal{M}$	MountainCar	39.16	16.36	17.8	71.2
	LunarLander	154.55	63.78	63.2	252.2

As we can observe, the proposed method is able to solve the task with good performance in 100% of the cases in both environments.

The best individuals found in the both environments (w.r.t. score) are shown in Fig.

4.3 and 4.4, respectively. We show in Figures 4.5 and 4.6 a visual representation of the sets of actions evolved through the course of the evolutionary process in the two cases. This observation validates the stagnation hypothesis made in Section 4.3, i.e., that most actions "converge" towards a fixed value in the first part of the evolutionary process.

A comparison of the proposed solutions (both in score and complexity) w.r.t. the results found in the literature and in the OpenAI Gym's leaderboard[23] can be seen in Table 4.4 and 4.5 for the MountainCarContinuous-v0 and the LunarLanderContinuous-v2 environments, respectively. Note that, for the based methods based on neural networks, an approximation of $\mathcal{M}$ based only on the architecture of the network is computed, without taking into account the other details of the network. While this may not lead to exact results, the difference is negligible for our comparison purposes.

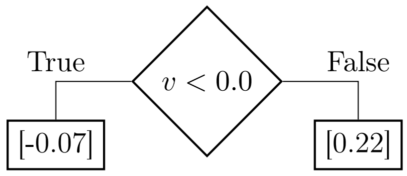


Figure 4.3: Best agent evolved in the MountainCarContinuous-v0 environment.

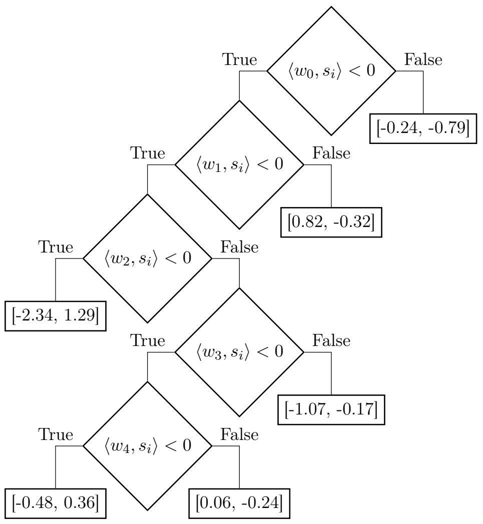

True
$\langle w_0, s_i \rangle < 0$
False
[-0.24, -0.79]
True
$\langle w_1, s_i \rangle < 0$
False
[0.82, -0.32]
True
$\langle w_2, s_i \rangle < 0$
False
[-2.34, 1.29]
True
$\langle w_3, s_i \rangle < 0$
False
[-1.07, -0.17]
True
$\langle w_4, s_i \rangle < 0$
False
[-0.48, 0.36]
[0.06, -0.24]

Figure 4.4: Best agent evolved in the LunarLander-v2 environment. $\langle a, b \rangle$ denotes the dot product between a and b.

[2]`https://github.com/openai/gym/wiki/Leaderboard`

[3]Only the results for which $\mathcal{M}$ could be computed are shown, i.e., the ones that were publicly available.

Table 4.4: Comparison of the results obtained by using our approach and the results from the literature and the OpenAI Gym leaderboard on the MountainCarContinuous-v0 environment.

Source	**Method**	**Score**	$\mathcal{M}$
Zhiqing Xiao	Closed-form policy	93.36	22.3
Ashioto	Actor-critic	92.45	2161.2
Nextgrid.ai	Soft actor-critic	92.38	248833.8
Keavnn	Soft actor-critic	92.75	31881.8
Liventsev et al. [69]	Neural program synthesis	91.57	71.2
Oller et al. [91]	Random Weighted Guessing	96.1	209.7
Ours	Decision tree	**98.77**	**17.8**

Table 4.5: Comparison of the results obtained by using our approach and the results from the literature and the OpenAI Gym leaderboard on the LunarLanderContinuous-v2 environment.

Source	**Method**	**Score**	$\mathcal{M}$
Keavnn	Soft actor-critic	**276.15**	20936
liu	Soft actor-critic	241.92	60615.9
Nextgrid.ai	Proximal policy optimization	239.59	36634.5
Jootten	Advantage actor-critic	240	22369.4
Goecks et al. [43]	Cycle-of-learning	261.8	236702
Goecks et al. [43]	Cycle-of-learning	253.24	236702
Goecks et al. [43]	Cycle-of-learning	245.24	236702
Ours	Decision tree	252.54	**94.7**

4.4.1 Discussion

MountainCarContinuous-v0

We observe that our solution dominates (as in Pareto front's domination) all the other solutions since it achieves a better score and lower $\mathcal{M}$ than all the other solutions. Interestingly, our approach is also able to outperform (in both score and interpretability) the closed-form policy that has been manually derived by Zhiqing Xiao.

LunarLanderContinuous-v2

Though our approach did not achieve the best score across all models in this case, it attained the 4th highest score, solving the task with a considerable margin (the task is considered solved when the score becomes greater than 200). However, its complexity is dramatically lower than all the other approaches: sacrificing approximately 8.5% of the score (w.r.t. the best solution by Keavnn) for an average complexity that is 3 orders of magnitude lower seems an extremely good trade-off.

4.4.2 Interpretation of the policies

In this subsection, we attempt to interpret the best policies found by the proposed method for the two environments, to understand how they work.

MountainCarContinuous-v0

The best agent evolved for this environment can be seen in Fig. 4.3. In this case, the agent is extremely easy to interpret: if velocity is negative, the agent builds potential energy by accelerating to the left. Otherwise, it exploits the kinetic energy by accelerating to the right.

LunarLanderContinuous-v2

In this case, the tree (shown in Fig. 4.4) is harder to interpret, due to its oblique splits. The weights for the splits are the following:

$w_0 = [0.228, 0.336, 0.485, 0.603, -0.421, -0.905, -0.615, -0.704]$;
$w_1 = [0.442, -0.175, 0.411, -0.482, -0.578, -0.735, -0.993, 0.390]$;
$w_2 = [0.950, -0.493, -0.213, 0.308, -0.661, 0.444, 0.945, 0.199]$;
$w_3 = [0.534, -0.406, 0.534, 0.704, -0.024, -0.378, 0.106, -0.759]$;
$w_4 = [0.281, 0.571, 0.999, -0.982, 0.214, 0.847, -0.927, 0.301]$.

So, to interpret the behavior of the agent, we have to analyze the behavior of the splits in different conditions. This environment gives as input to the agent 8 variables: p_x, the position on the x-axis; p_y, the position on the y-axis; v_x, the velocity on the x-axis; v_y, the velocity on the y-axis; θ, the angle w.r.t. the x-axis; ω, the angular velocity w.r.t. the y axis; c_l, the contact sensor input for the left leg; c_r, the contact sensor input for the right leg. The actions work as follows:

- The first action corresponds to the acceleration on the y axis. When it is less than 0.5, no acceleration is given. On the other hand, when it is in $[0.5, 1]$ an acceleration is supplied with power ranging from 50% to 100%.

- The second action corresponds to the acceleration on the x axis. When it is less than -0.5, the engine accelerates towards the left (negative x), and when it is greater than 0.5, the engine accelerates towards the right (positive x).

Since c_l and c_r can only assume a value of either 0 or 1, they can be seen as a variation of the threshold for the split, i.e., the condition becomes:

$$a \cdot p_x + b \cdot p_y + c \cdot v_x + d \cdot v_y + e \cdot \theta + f \cdot \omega < -g \cdot c_l - h \cdot c_r \tag{4.1}$$

where $a \dots h$ are the components of the vector w_k.

For simplicity, we set $-g \cdot c_l - h \cdot c_r = t$, such that the conditions that we want to analyze take the following form:

$$a \cdot p_x + b \cdot p_y + c \cdot v_x + d \cdot v_y + e \cdot \theta + f \cdot \omega < t \tag{4.2}$$

where t is a dynamic threshold (i.e., it changes when contact with the ground happens).

Starting from the root of the tree (i.e., the uppermost node) shown in Fig. 4.4, let us analyze each condition and its actions.

First condition: By writing the first condition as done in Eq. 4.2, and grouping the terms in x, y, and θ, we obtain:

$$a(p_x + \frac{c}{a}v_x) + b(p_y + \frac{d}{b}v_y) - e(\theta + \frac{f}{e}\omega) < t \tag{4.3}$$

Now, by noting that $\frac{c}{a} \approx \frac{d}{b} \approx \frac{f}{e} \approx 2$, we can approximate Eq. 4.3 as:

$$a(p_x + 2v_x) + b(p_y + 2v_y) < e(\theta + 2\omega) + t \tag{4.4}$$

This policy can be interpreted as follows: if the weighted sum of the two components of the expected position in the next two time steps (if the agent does not fire any engine, and supposing that friction is negligible) is smaller than the expected angle, then check the other conditions, otherwise accelerate to the left. For instance, when the expected angle is negative and the predicted positions are positive, the agent accelerates the lander to the left, to avoid overturning.

Second condition: This condition can be formulated as:

$$a(p_x + \frac{c}{a}v_x) - b(p_y + \frac{d}{b}v_y) - e(\theta + \frac{f}{e}\omega) < t \tag{4.5}$$

which can be refactored as:

$$a(p_x + \frac{c}{a}v_x) - e(\theta + \frac{f}{e}\omega) < b(p_y + \frac{d}{b}v_y) + t \tag{4.6}$$

The condition can be interpreted as follows: if the weighted sum of the predicted next position and predicted next angle is smaller than the predicted position on the y-axis, then check another condition, otherwise fire the main engine. Basically, this condition checks whether the lander is unbalanced towards the left. In fact, since the agent knows (from the previous condition) that the lander is not falling on the right, when this condition is false (i.e., it is not going to fall on the left) the agent fires the main engine to soften the landing.

Third condition: This condition cannot be analyzed as the previous ones. For this reason, we analyze the mean value (and standard deviation) of each term and obtain:

$$\begin{aligned} a \cdot \overline{p_x} \approx -10^{-2} \pm 8.5 \cdot 10^{-2}; b \cdot \overline{p_y} \approx 1.5 \cdot 10^{-1} \pm 2.4 \cdot 10^{-1} \\ c \cdot \overline{v_x} \approx -2 \cdot 10^{-3} \pm 4 \cdot 10^{-2}; d \cdot \overline{v_y} \approx -4 \cdot 10^{-2} \pm 5 \cdot 10^{-2} \\ e \cdot \overline{\theta} \approx 3 \cdot 10^{-2} \pm 8 \cdot 10^{-2}; f \cdot \overline{\omega} \approx 1.5 \cdot 10^{-2} \pm 5 \cdot 10^{-2} \\ g \cdot \overline{c_l} \approx 4 \cdot 10^{-1} \pm 5 \cdot 10^{-1}; h \cdot \overline{c_r} \approx 9 \cdot 10^{-2} \pm 10^{-1} \end{aligned} \tag{4.7}$$

this means that we can ignore all the variables except for p_y and c_l (at the expense of a negligible loss in performance). Then, this condition is simply:

$$-b \cdot p_y + g \cdot c_l < 0 \tag{4.8}$$

that, with $\frac{g}{b} \approx 2$, can be rewritten as:

$$p_y > \frac{g}{b} c_l \tag{4.9}$$

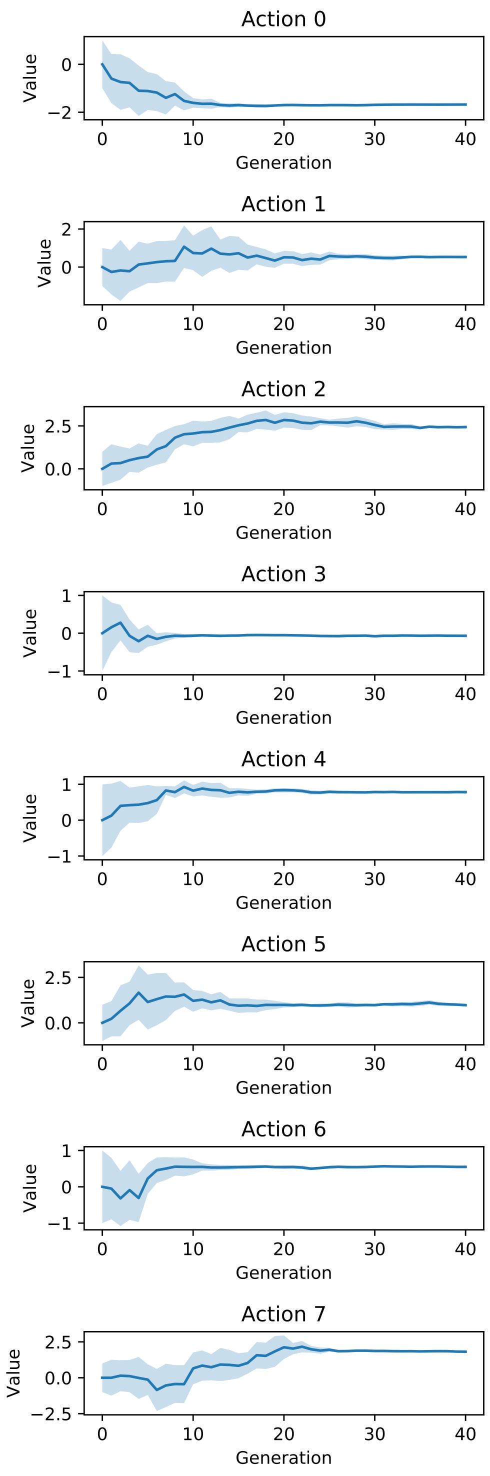
Eq. 4.9 can be interpreted as follows. From the previous conditions, we know that the lander is unbalanced towards the left. This condition checks that there is no contact with the ground ($\max p_y = 1.5$). When there is contact with the ground, the second term is approximately 2, and then the condition evaluates to false. If there is no contact with the ground, then the agent accelerates the lander to the right to balance it. Otherwise, it keeps all the engines off (to minimize energy consumption).

Finally, in the last two conditions that are checked by following the right branch of the third condition, all the actions that can be executed correspond to switching off both engines.

4.5 Conclusions

In this chapter, we propose a method that allows us to obtain interpretable agents for reinforcement learning tasks with continuous action spaces. We do so by combining co-evolutionary approaches with reinforcement learning techniques such as Q-learning.

The proposed method proved to be able to solve with satisfactory performance two well-known benchmarks. In these two tasks, the agents have performances that are comparable (or even better) w.r.t. the state of the art (including non-interpretable approaches), while having substantially smaller complexity.



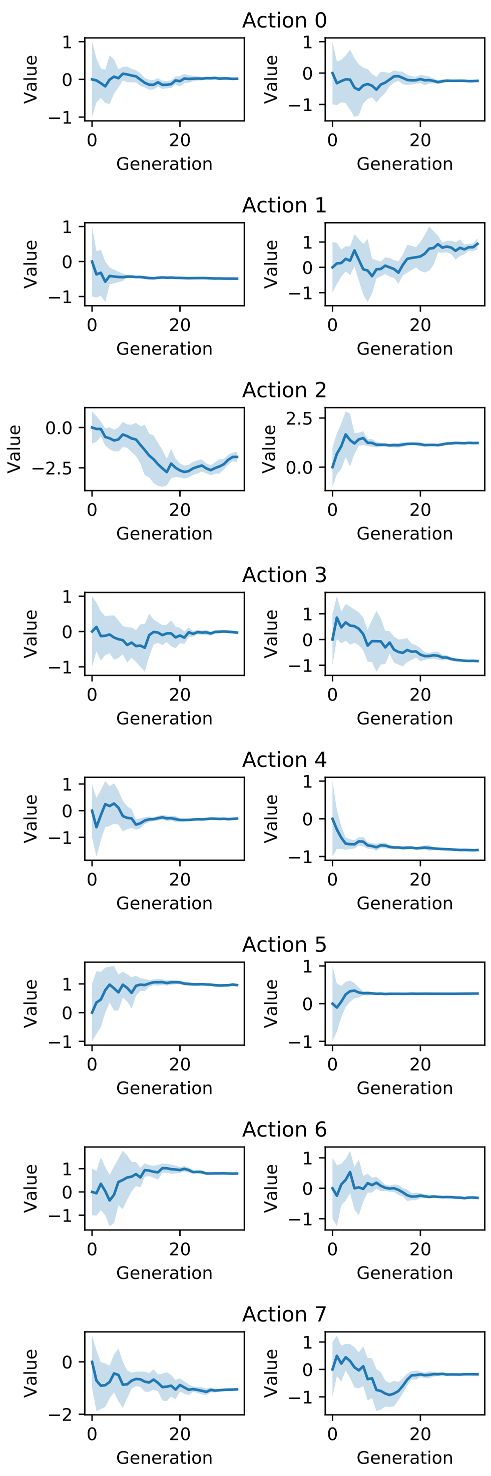


Figure 4.5: Evolution of action values (solid: mean, shaded: std. dev.) in the MountainCarContinuous-v0 environment.

Figure 4.6: Evolution of action values (solid: mean, shaded: std. dev.) in the LunarLanderContinuous-v2 environment.

However, this approach has some limitations: for example, the agents are not able to perform actions that are proportional to input variables (e.g., to implement proportional controllers).

Future work includes: the development of more sophisticated grammars that allow the agents to select which variables to address in each condition, the use of more sophisticated optimization techniques to tackle harder problems, the replacement of static actions (in the "Decision Encoding" module) with linear models, to obtain more advanced control systems.

Chapter 5

Modular Interpretable Systems

Based on: Custode, Leonardo Lucio, and Giovanni Iacca. "Interpretable pipelines with evolutionary optimized modules for reinforcement learning tasks with visual inputs." Proceedings of the Genetic and Evolutionary Computation Conference Companion. 2022.

5.1 Introduction

The recent literature has proposed some seminal approaches for performing RL with interpretable models [129, 28], e.g., based on evolutionary computation [23, 22] (See Chapters 3 and 4). So far, these interpretable reinforcement learning (IRL) approaches have been mostly tested on relatively simple control RL tasks, such as those of the OpenAI gym benchmark [13], on which they have obtained fairly good results. However, these methods are not expected to work well in RL tasks with raw input data, as in these contexts each variable alone (e.g., a pixel) may not be meaningful enough to take decisions.

In this chapter, we introduce the concept of *interpretable pipelines* for tackling RL tasks with visual inputs. An interpretable pipeline is a multi-agent system where each agent is an interpretable model with well-defined responsibilities, which communicates with the other agents in the pipeline. We optimize such pipelines by means of a co-evolutionary approach, in which different evolutionary algorithms (EAs) run in parallel, each of which optimizes a single agent. We test our approach on three different Atari games, where we observe that the proposed method is able to achieve satisfactory performance in deterministic settings (i.e., without frame-skipping). On the other hand, our approach is not able to achieve satisfactory performance in environments with stochastic frame-skipping (yielding higher uncertainty about the future), which provides some hints for future work.

This chapter is structured as follows. The next section makes a brief overview of the related work. Section 5.2 explains the methodology used in our experiments. In Section

5.3 we present the results and, finally, in Section 5.4 we draw the conclusions of this work.

5.2 Method

To evolve interpretable pipelines for image-based RL tasks, we build on some of the aforementioned previous works from the literature [23, 22, 142]. In detail, our proposed system is an interpretable pipeline composed of two parts:

- a vision module, that is meant to process the input to extract a pre-defined number of features;
- a decision module, whose purpose is to decide which action to take, based on the features extracted by the vision module.

It is important to note that in this chapter, features represent high-level visual information (position of relevant objects), while decisions are actions taken by a decision tree (playing one of the Atari games considered in our experimentation). However, the proposed pipelines can be extended to tackle classification problems.

A graphical representation of this kind of pipeline is shown in Figure 5.1. In the following subsections, we will first explain the details of the two kinds of modules, and then we will describe our co-evolutionary approach.

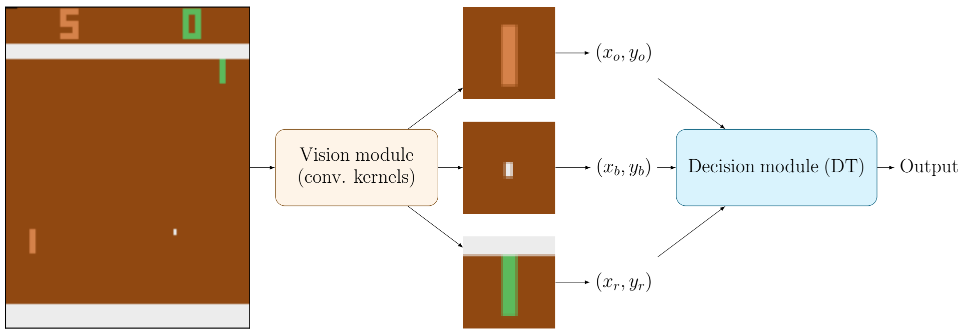


Figure 5.1: An example of the proposed pipelines (note that the frames are taken from the Pong environment).

5.2.1 Vision module

In [142], the authors used a simplified self-attention module that ranks the patches of the image by importance. Then, the coordinates of the k most important patches are given in input to an LSTM network [49] that computes the decision to take, where k is a predetermined parameter. Similarly, here we use a vision module, whose purpose is to find the k most important patches in the image, returning their coordinates.

However, in order to have better interpretability, rather than a self-attention module, in our vision module we employ k convolutional kernels, each of which is supposed to detect a single entity of interest in the image.

Moreover, using k distinct kernels instead of a single self-attention module allows us to have a fixed-order constraint on the input: e.g., given two kernels k_A and k_B which respectively handle the detection of two distinct objects A and B, we are guaranteed that the decision module will receive in input the position of the two entities always in the same order. In contrast, when using a self-attention module as done in [142], we do not have such guarantees, and thus the decision module has to handle such cases of inputs in unpredictable order, thus requiring a more complex decision logic.

The pseudo-code that describes how the vision module works is shown in Algorithm 5. In the algorithm $R : \mathbb{R}^{h' \times w'}$ is the output of the convolution between the image and the kernel, which results in a matrix of size $h' \times w'$, where $h' = h_{img} - 2\lfloor \frac{h_k}{2} \rfloor$ and $w' = w_{img} - 2\lfloor \frac{w_k}{2} \rfloor$, and h_{img} is the height of the image, h_k is the height of the kernel, w_{img} is the width of the image, w_k is the width of the kernel.

Algorithm 5: Computation performed by the vision module

```
Data: an image, $\mathbf{X}$
Data: a list of kernels, $\mathbf{k}$
Result: a list of coordinates, $\mathbf{c}$
1 $\mathbf{c} \leftarrow []$;
2 for $k \in \boldsymbol{k}$ do
3     $R \leftarrow convolution(\mathbf{X}, k)$;                // apply kernel $k$ to $\mathbf{X}$
4     $(x^*, y^*) \leftarrow \underset{(x,y)}{argmax}(R[y, x])$;   // $(x, y)$ size of $\mathbf{X}$
5     $\mathbf{c} \leftarrow \mathbf{c} + [(x^*, y^*)]$;          // concatenate $\mathbf{c}$
6 end
7 return $\mathbf{c}$;
```

5.2.2 Decision module

The goal of the decision module is to perform "reasoning" on the coordinates of the most important patches of the input images and to take a decision on top of them.

To keep the interpretability of the pipelines high, we use an automatically synthesized decision tree as a decision module. This decision tree takes as input the list of coordinates computed by the vision module, thus it does not use the raw data of the whole image.

This module then returns a discrete action that can be directly sent to the environment, see the examples shown in Figure 5.6 and the corresponding analysis reported in Section 5.3.1.

5.2.3 Co-evolutionary process

To optimize the vision and decision modules adopted within the proposed pipelines, we employ a co-evolutionary approach [102]. In particular, we combine Covariance Matrix Adaptation Evolution Strategies (CMA-ES) [45] with Genetic Programming (GP) [58], as shown in Figure 5.2. We use CMA-ES to evolve the parameters of the vision module, i.e., the weights of each kernel module. CMA-ES has been chosen for being one of the most robust algorithms for derivative-free optimization. On the other hand, by using Genetic Programming (more specifically, strongly typed Genetic Programming [87]), we evolve decision trees, as described below.

While using pipelines there is the risk of having bad performance due to compounding errors, we expect the decision tree, i.e., the one that is at the end of the pipeline, to learn to be robust to the errors of the vision module. This is due to the fact that, since the two populations are evolved simultaneously, the fitness obtained by the DTs implicitly depends on their robustness to input noise.

Genetic Programming for evolving decision trees

To evolve decision trees, we use two types of nodes: condition nodes and leaf nodes. A condition is represented as a node with three child nodes: a comparison node and two nodes (either leaves or conditions).

A comparison node is composed of a node representing an operator (e.g., “less than”, “equal to”, and “greater than”) that has two child nodes, encoding two expressions.

An expression node can be either a constant, a variable, or an arithmetical operation between expression nodes.

Since the interpretability of a decision tree crucially depends on the complexity of the conditions (i.e., more complex hyperplanes are hard to interpret, e.g., the ones presented in [28]), not only do we use a constant to limit the size of the tree, but we also employ a different constant to limit the depth of the conditions.

By doing so, we can control better the interpretability of the tree by allowing, for instance, deeper trees with simple conditions.

Fitness evaluation

In order to evaluate the quality of the individuals from both populations, we pair each individual with all the individuals of the other population.

We evaluate the pair on e episodes, and we compute the average score (across episodes) for the pair $\bar{s}_{i,j}$, where i is the index of the individual from the population of vision modules and j is the index of the individual from the population of decision modules.

Then, we define the fitness of an individual as the maximum $\bar{s}$ that that individual obtained across all the pairings, i.e., $f_i = \underset{j}{max}(\bar{s}_{i,j})$ for vision modules and $f_j = \underset{i}{max}(\bar{s}_{i,j})$ for decision modules.

While using a different operator such as the mean (across pairings) seems more meaningful, from preliminary experiments we observed that using the mean leads to a stagnation of the co-evolutionary process. We hypothesize that this is due to the fact that, by using the mean, the fitnesses are affected by the randomness in the evaluation phase and by the qualities of all the individuals of the other population, so that an individual that obtains medium-low scores in all the pairings may have a higher fitness than an individual that works very well combined with a specific individual of the other population but works poorly with all the other individuals.

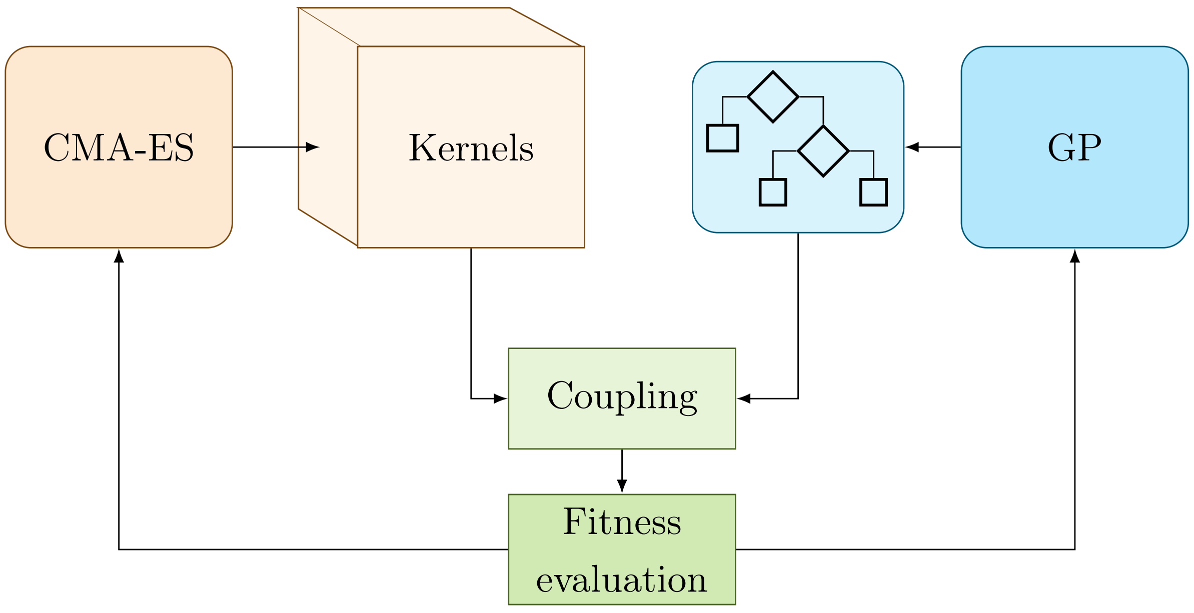


Figure 5.2: Scheme of the proposed co-evolutionary process.

Reducing the number of evaluations

The computational cost (in terms of the number of evaluations, where a single evaluation relates to an instance of the proposed pipelines) for the co-evolutionary process is $\mathcal{O}(n_p^{CMA} \cdot n_p^{GP} \cdot n_g \cdot n_e)$, where n_p^{CMA} is the population size for CMA-ES, n_p^{GP} is the population size for the Genetic Programming (assuming that each individual of one population is paired with all the individuals of the other population), n_g is the number of generations, and n_e is the number of episodes.

To reduce such cost, we propose a mechanism that evaluates the *behavior* (i.e., the outputs in response to given inputs) of each individual in the two populations, and avoids the evaluation of individuals whose behavior is too similar. At each generation, for both the vision and decision modules in the current populations of the co-evolutionary process, we give them in input a set of samples and we store their outputs. Then, for both populations (separately), we cluster these outputs by means of the DBSCAN algorithm [127]. More specifically, the samples used for clustering are calculated as follows.

- **Vision modules** – For these modules, we evaluate their behavior by giving them in input a set of n_{vm} images sampled randomly from an episode used at the beginning of the evolutionary process. Then, we use the concatenation of all the outputs of each module as input samples for the clustering process.
- **Decision modules** – For these modules, we evaluate their behavior by giving them in input n_{dm} randomly sampled coordinates from the vision modules' output space. Then, we perform a one-hot encoding of the decision modules' outputs and, by concatenating all the one-hot encoded outputs for each module, we obtain the input samples for the clustering process. The reason underlying the one-hot encoding is related to the "meaning" of the input samples for the clustering process. In fact, if we used the raw outputs of the decision modules as input samples for clustering, we would give an implicit "proximity" meaning such that an action $a_i \in \mathcal{A} = \{a_0, \dots, a_i, a_{i+1}, a_{i+2}, \dots, a_A\}$ would be considered "closer" to action a_{i+1} than a_{i+2}, while the meaning of such actions may not have such a "similarity" principle. Instead, by performing a one-hot encoding, the distance between two different actions (performed by different decision modules given the same input) is always constant, thus removing the bias associated to the raw outputs.

Once clustering has been performed, we evaluate only the centroids of each cluster (and the individuals not assigned to any cluster) and we assign, for each individual of the cluster, the same fitness.

The reason underlying the choice of the DBSCAN algorithm is due to the fact that this algorithm is based on the concept of *density*, so that we can intuitively set the thresholds for considering two points as "close". Moreover, this algorithm does not require specifying a pre-defined number of clusters, so this means that if we find a cluster, its points are close enough to be considered similar. Finally, we do not use constant parameters for the distance threshold ε. Instead, we use an initial ε_0, which, at each step, is multiplied by a scaling constant $\gamma \in [0, 1)$. This scaling mechanism gives us the following properties. At the beginning of the evolutionary process, since the diversity is high, we perform a coarse-grained clustering of the individuals so that we can significantly speed up the initial generations. On later generations, the thresholds become increasingly smaller, so that even not-so-diverse individuals are evaluated separately: this, in turn, leads to a greater number of simulations, which allows us to discriminate individuals in a more fine-grained fashion. Finally, when the ε_i parameter (for the i-th generation) tends to zero, we avoid evaluating only those individuals that are behaviorally *identical*. However, since in the final generation the evolutionary process is expected to converge, the number of evaluations decreases again.

5.2.4 Experimental setup

We test our approach in three environments from the Atari Learning Environment implemented in OpenAI Gym [9, 75, 13]. In particular, we use the Pong-v4, Bowling-v4, and Boxing-v4 environments, hereinafter simply referred to as Pong, Bowling, and Boxing, respectively. Table 5.1 shows the parameters used for the evolution. The parameters have been determined empirically, based on the knowledge gained in preliminary experiments. The parameter setting is the same for the three environments. The only difference is in the number of available actions for the decision module, that is 4, 4, and 6 respectively for Pong, Bowling, and Boxing. It is important to note that the number of episodes is quite low. This is due to the fact that in the fitness evaluation phase, there is no learning involved (differently from Chapters 3 and 4), and thus the episodes are only used to evaluate the pipelines on average. While a higher number of episodes would certainly increase the precision of the estimate of the performance of the pipeline in unseen episodes, it would significantly increase the computational cost of the search process. All the runs were performed using an HPC that allocated 70 CPUs and 4GB of RAM for each run, with a time limit of 6 hours. For each environment, we consider both the settings with and without stochastic frame-skipping, the first one being harder than the second one. Stochastic frame-skipping is harder because it is akin to a non-uniform sampling (while environment without frame-skipping or with deterministic frame-skipping do a uniform sampling). In fact, the time elapsed between two sensory inputs of the agents can vary, and this introduces two problems: (1) it is harder for the agent to compute time-dependent properties of the objects, such as velocities and accelerations; and (2) the agent cannot exactly know what is the outcome of an action, since it does not know how long the action will be applied to the environment (i.e., the environment has "sticky actions"). For each environment and setting, we perform 5 runs. This number of runs is enough to ensure statistical significance since, given the results shown in Table 5.2, the confidence interval (95%) is low enough to validate our conclusions (see the next section for the results). The confidence interval has been computed as $CI_{95\%} = \frac{t_{\{5-1,0.025\}}\sigma}{\sqrt{5}}$, where t is the critical value from the Student's t distribution, and σ is the standard deviation.

5.2.5 Image pre-processing

Before feeding an image to the vision module, we perform the following pre-processing steps:

1. We remove the topmost 35 pixels: these pixels correspond to the part of the image that describes the "status" of the game, thus we remove it to avoid that the evolved pipelines use this information to take decisions.

2. We resize the image to 96×96: this operation speeds up the vision modules' computations, without losing information about the entities present in the game.

3. We normalize the input in $[0, 1]$ by performing a min-max normalization.

5.2.6 Environments

All the environments share the same type of observations, which consist in 210×160 RGB images, where each pixel is encoded with three 8-bit integers (one for each channel).

Pong

The Pong environment (and its counterpart without frame-skipping, PongNoFrameskip) is a game in which there are two agents (here intended as "players", not to be confused with the agents that compose our pipelines), who play the Pong game. Each agent controls a racket (which can move on the y axis), and the goal of each agent is to send the ball farther than the opponent's x position.
Reward: Each point scored by the playing agent (i.e., the green racket) gives a reward of 1 point. On the other hand, each point scored by the opponent agent gives a reward of -1 point. In all the other cases, the reward given to the agent is 0.
Actions: The action space consists of 4 actions: two of them are NOP actions (i.e., they do not move the racket), one moves the racket upwards, and another moves the racket downwards.

Termination criterion: The simulation ends when either the agent or the opponent score 21 points or after 10^5 elapsed timesteps.

Bowling

The Bowling environment (and its counterpart BowlingNoFrameskip) consists in a bowling game, where the agent has to throw a ball to hit some pins. Thus, differently from Pong, this is a single-player game.
Reward: After each round (i.e., throwing twice the balls, or one in case of a strike), the agent receives a reward equal to the number of pins that have been hit. Moreover, strikes and spares provide extra points (also given at the end of the round). In all the other cases, the reward given to the agent is 0.
Actions: The agent can perform 4 actions: a NOP action, an action that moves the ball upwards, one that moves the ball downwards, and an action that allows the agent to throw the ball.
Termination criterion: The simulation ends after 10 rounds or after 10^5 timesteps. However, since an agent that does not know how to throw the ball will make the simulation become extremely time-consuming, we reduced the number of maximum timesteps to 5×10^3 during the evolutionary process. Then, to obtain the results shown in Section 5.3, we test the best pipelines evolved on the full task.

Boxing

In the Boxing (and its counterpart BoxingNoFrameskip) environment, there are two agents that compete: the white boxer (played by the agent) and the black boxer (which is the opponent).
Reward: When the agent hits the opponent, it can receive either 1 or 2 points, depending on the distance between the agents (hitting the opponent from a closer position gives more points). On the other hand, when the agent is hit by the opponent it receives a reward that is the opposite of the one previously described. In all the other timesteps, the reward given to the agent is 0.
Actions: The environment provides 18 actions, composed of: NOP, movements in the 4 cardinal directions, punching, and combinations of movements and punching (including diagonals). However, here for simplicity, we reduce the set of actions to the non-composite actions, i.e.: NOP, movements in the 4 directions, and punching.
Termination criterion: The simulation ends when either the agent or the opponent scores 100 points or after 10^5 elapsed timesteps.

Table 5.1: Hyperparameters used in the experimentation.

Parameter	**Value**
CMA-ES Population size (n_p^{CMA})	50
CMA-ES Initial mean	0
CMA-ES Initial σ	0.1
GP Population size (n_p^{GP})	50
GP Crossover probability	0
GP Mutation probability	1
GP Tournament size	10
GP Elitism	Yes (1 elite)
DBSCAN Number of samples (vision) (n_{vm})	100
DBSCAN Number of samples (decision) (n_{dm})	100
Number of generations (n_g)	100
Maximum depth of decision tree	4
Maximum depth of condition	2
Number of convolutional kernels (k)	2
Size of convolutional kernels	5×5×3
Size of the images	96×96
Number of episodes (n_e)	3
Time limit	6 hours
Number of runs	5

5.3 Results

The results obtained from the 5 runs for each environment and setting are shown in Table 5.2. The results shown in the table have been obtained by testing the best-evolved pipelines on 100 unseen episodes. We observe that our approach performs well in environments without frame-skipping, but it performs poorly in settings with frame-skipping. While state-of-the-art approaches are able to achieve very good performance even in cases with frame-skipping, it is important to point out that these approaches are not interpretable, thus they do not provide any information about their inner processes. On the other hand, while our approaches do not perform well when trained in setups with frame-skipping, they are completely transparent, potentially allowing an adaptation to domains with frame-skipping. In fact, the non-interpretable approaches have $\mathcal{M}'$ scores in the order of 10^7, while ours are in the order of 10^2. This difference can be further appreciated in Figure 5.4. These results encourage future research in IRL, as adding more complexity to the pipelines would still yield a significant gain in interpretability w.r.t. the current non-interpretable state-of-the-art.

Figure 5.3 shows a comparison of the distribution of the scores of the best pipelines evolved in 5 runs of each environment and setting (normalized w.r.t. the minimum and maximum possible scores of the environments). Once again we observe that, in all the cases, the settings without frame-skipping achieve very good performance (i.e., they are closer to one), while in the cases where frame-skipping is applied, our algorithm is not able to achieve good performance.

Table 5.2: Summary of the results of the best pipelines evolved in 5 runs of each setting for each environment. Each pipeline has been tested on 100 unseen episodes. "FS" stands for the setting with frame-skipping, "NoFS" indicates the setting without frame-skipping, "SoTA" indicates the results from the (non-interpretable) state-of-the-art. Please note that all the papers from the literature report results only in the FS setting.

Env.	**Setup**	**Mean**	**Std.**	**Best**	**Reference**
Pong	FS (ours)	-6.97	8.49	7.42	
	NoFS (ours)	21.00	0.00	21.00	
	FS (SoTA)	-	-	21.00	[150, 121, 124]
Bowling	FS (ours)	189.68	5.06	196.53	
	NoFS (ours)	220.20	18.00	240.00	
	FS (SoTA)	-	-	260.00	[124, 31]
Boxing	FS (ours)	48.59	19.05	75.37	
	NoFS (ours)	92.78	3.09	98.00	
	FS (SoTA)	-	-	100.00	[9, 124, 50, 38, 4, 34, 126]

The fitness trends, as well the cumulative number of evaluations across generations (mean, represented as solid line, ± std. dev., represented as shaded area, across 5 runs) are shown for each environment and setting in Figure 5.5. We observe that the clustering mechanisms allows us to save a significant amount of evaluations, increasing the efficiency of the co-evolutionary process (approximately saving 41%±12% evaluations).

5.3.1 Analysis of the best pipelines evolved

We conclude our presentation of the results with an analysis of the best-evolved pipelines obtained by means of the proposed method. Note that, since the settings without frame-skipping produced better results, we will limit our analysis to these settings.

The best pipelines obtained are shown graphically in Figure 5.6. In order to improve the readability, the decision modules have been manually simplified, deleting the conditions that always evaluate to the same truth value.

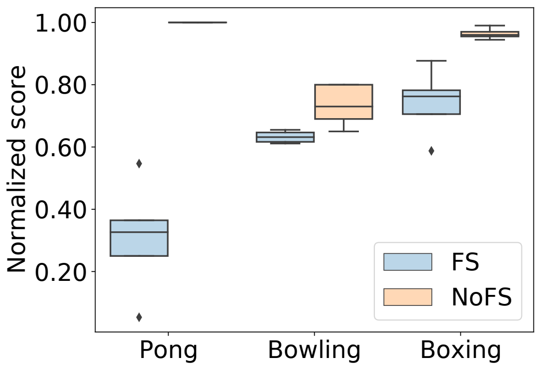


Figure 5.3: Distribution of the normalized scores of the best pipelines evolved in 5 runs of each setting for each environment, normalized w.r.t. the minimum and maximum scores allowed by each environment.

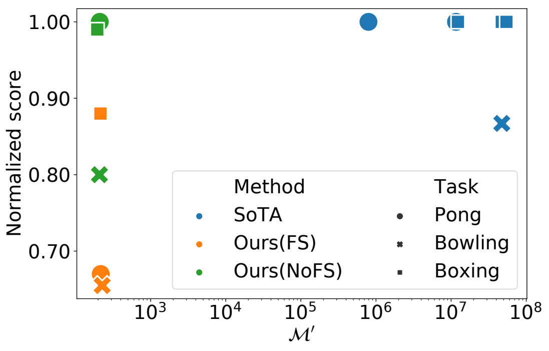
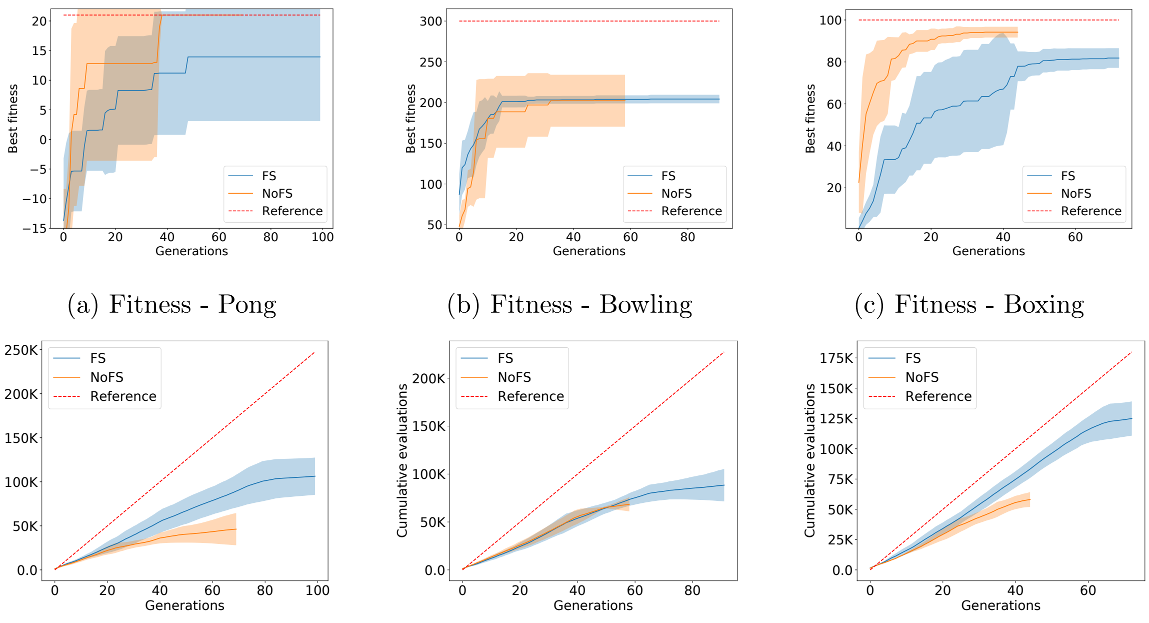


Figure 5.4: Comparison in terms of average normalized score across 100 episodes and $\mathcal{M}'$ between the best-evolved pipelines and the state-of-the-art (SoTA) methods from the references reported in Table 5.2.



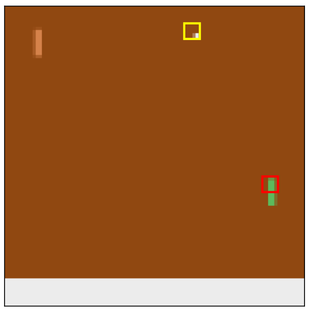
(a) Fitness - Pong (b) Fitness - Bowling (c) Fitness - Boxing

(d) Evaluations - Pong (e) Evaluations - Bowling (f) Evaluations - Boxing

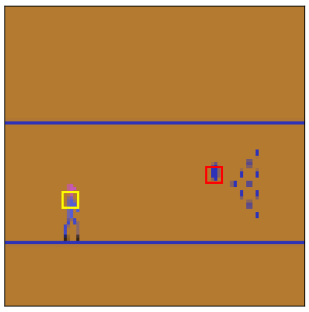
Figure 5.5: Fitness and number of evaluations (mean ± std. dev. across 5 runs) over time. For the fitness, the reference is the best possible fitness allowed by the environment. For the number of evaluations, the reference is the number of evaluations needed if the clustering mechanism described in Section 5.2.3 is not applied.

Pong

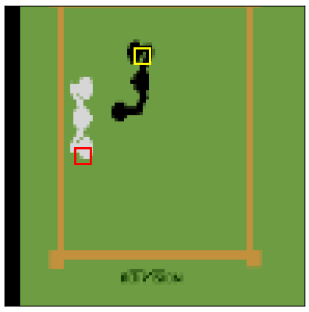
As shown in Figure 5.6a, the vision module of the best pipeline evolved for Pong detects the two most important entities: The racket and the ball, giving in output their coordinates:



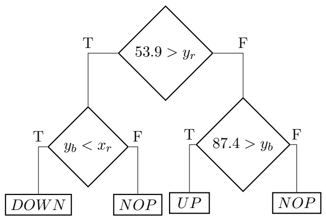
(a) Vision module - Pong



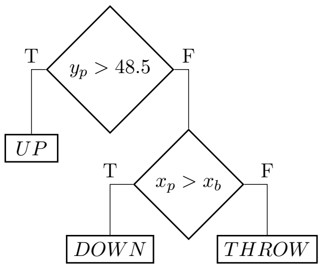
(b) Vision module - Bowling



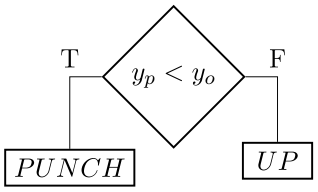
(c) Vision module - Boxing



(d) Decision module - Pong



(e) Decision module - Bowling



(f) Decision module - Boxing

Figure 5.6: Best pipelines evolved for the three environments (without frame-skipping). On the top, we show the entities discovered by the vision modules, while on the bottom we show the corresponding decision modules.

x_r, y_r, x_b, y_b, where the subscript r refers to the player's racket and the subscript b refers to the ball.

The policy of the decision-making module for this environment, as shown in Figure 5.6d, works as follows. Firstly, it checks whether the racket is on the upper part of the screen. If so, it checks whether the y-coordinate of the ball is less than the x-coordinate of the racket. This condition can be simplified: the horizontal position of the racket is constant ($x_r = 82$). So, this condition is equivalent to $y_b < 82$, which means that the ball is not near the bottom wall (white part in Figure 5.6a). Then, if this condition evaluates to true, the decision module decides to go downwards, otherwise, it does not perform any action. On the other hand, when the racket is on the lower part of the screen, the decision module checks $87.4 > y_b$, i.e., if the ball is not near to the bottom wall. If so, it decides to go upwards, otherwise, it does not perform any action.

Bowling

The entities recognized by the vision module are shown in Figure 5.6b, and they are: the person throwing the ball and the ball. Thus, the coordinates that the vision module sends to the decision module are: x_p, y_p, x_b, y_b, where the subscript p refers to the person, and the subscript b refers to the ball.

The decision module (Figure 5.6e) performs the following decision-making process. First of all, it checks the vertical coordinate of the person and, if it is too low on the screen ($yp > 48.5$, note that the top-left screen has coordinates $(0, 0)$), it moves the person upwards, so that it is positioned correctly to perform a good shot. Then, after the person is correctly positioned on the bowling alley, it checks another condition to understand whether the ball has been thrown or not ($x_p > x_b$). In fact, if the ball has not been thrown, it stands behind the person. Thus, if the ball is moving toward the pins, it moves the position of the ball downwards. Otherwise, if the ball is still in the hand of the person, it makes the person throw the ball.

Boxing

In Figure 5.6c, we show the two entities recognized by the vision module: the punch of the player, and the position of the right arm of the opponent. Thus, the vision module returns their coordinates: x_p, y_p, x_o, y_o, where the subscript p refers to the player and the subscript o refers to the opponent. Moreover, by testing the vision module, we have observed that the kernel does not work perfectly, i.e., it happens that the kernel that should recognize the position of the player recognizes a part of the opponent. Nevertheless, this pipeline manages to achieve a near-optimal average score (on unseen episodes) of 98, i.e., 1% away from the maximum.

The decision module, shown in 5.6f, works as follows. If the player is positioned upper than the opponent (again, note that the top-left corner has coordinates $(0, 0)$), it tries to punch the opponent, otherwise, it goes upwards, to reach the opponent.

Interestingly, such a simple policy (encoded by the decision module) allows us to understand some properties of the pipeline and the game itself. First, the game can be played with very good results by using only two actions (note that the environment provides 18 actions for the player). Moreover, the other actions are not needed because the opponent chases the player. For this reason, the decision module finds more advantageous to use a semi-defensive strategy, i.e., always punch if the opponent is reachable by the punches, and chase it only when it is upwards.

5.4 Conclusions

In this chapter, we propose a novel methodology (based on a divide-et-impera paradigm) for evolving interpretable systems for RL tasks with visual inputs. In particular, our approach is based on pipelines characterized by a separation of concerns between a vision module (which uses convolutional kernels) and a decision module (based on a decision tree). Our results show that our approach is able to learn how to effectively play three Atari games in simplified settings (i.e., without frame-skipping). However, when applying frame-skipping to the environments, our approach is not able to achieve satisfactory performance.

Moreover, observing the behaviors of the pipelines evolved confirmed the hypothesis made in Section 5.2.3. In fact, we observed that even though the coordinates computed by the vision modules were pretty noisy, the resulting policies were still able to achieve good performance.

Future work should introduce ways to address the uncertainty in non-deterministic settings (i.e., with frame-skipping), in order to make this approach more robust to noise in the environment and achieve performances comparable to those of the state-of-the-art algorithms developed for these settings. In this sense, two possibilities would be to incorporate in our approach some mechanisms used in evolutionary optimization in the presence of noise [3], or using fuzzy [52] or probabilistic [62] decision trees.

Chapter 6

Approaches for Multi-Agent Interpretable Reinforcement Learning

Based on: Crespi, Marco, Leonardo Lucio Custode, and Giovanni Iacca. “Towards Interpretable Policies in Multi-agent Reinforcement Learning Tasks.” Bioinspired Optimization Methods and Their Applications: 10th International Conference, BIOMA 2022, Maribor, Slovenia, November 17–18, 2022, Proceedings. Cham: Springer International Publishing, 2022.

6.1 Introduction

In recent years, the application of Deep Learning (DL) to the field of Multi-Agent Reinforcement Learning (MARL) led to the achievement of significant results in the field. While DL allows training powerful multi-agent systems (MASs), it has some drawbacks. First of all, to exploit state-of-the-art deep reinforcement learning (RL) algorithms, one often has to employ centralized approaches for training [94], which limits the scalability of the system, i.e., no agents can be added after the MAS has been trained. Moreover, deep RL methods suffer from an even worse drawback: the lack of *interpretability*. In fact, in safety-critical or high-stakes contexts, DL approaches cannot be employed as they are not fully predictable [116, 119, 117] and, thus, they may exhibit unexpected behaviors in edge cases. While interpretability in RL is an important concern, in MARL it is even more important. In fact, in contrast to traditional RL setups where safety can be assessed by inspecting the trained agent, in MARL not only do we need to analyze each agent, but we also need to understand their *collective behavior*.

In this chapter, extend the approach presented in Chapter 3 for training an interpretable MAS. More specifically, we extend the setup proposed in Chapter 3 by using a *cooperative co-evolutionary algorithm* [103] in which each evolutionary process addresses the evolution of an agent of the MAS. As a baseline, we also provide the results obtained when a single policy is trained for all the agents in the MAS. We evaluate our approach on the Battlefield task from MAgent [160] (implemented in the PettingZoo library [143]). The teams evolved with our approach are able to obtain satisfactory performance, eliminating the whole opponent team in up to 98% of the cases.

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

The rest of the chapter is structured as follows. In the next section, we describe the proposed method. Then, in Section 6.3 we present the numerical results. Finally, we draw the conclusions in Section 6.4.

6.2 Method

The goal of our work is to produce interpretable agents that are capable of cooperating to solve a given task. To do that, we evolve populations of interpretable agents in the form of decision trees. To evolve these decision trees, we use the same approach described in Chapter 3. It is important to note that our method employs a cooperative co-evolutionary process [103], where each population optimizes the structure of the tree for a particular agent of the environment.

6.2.1 Creation of the teams

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

Each agent-local population has n_p individuals, such that n_p different teams are created. Each i-th team is formed by the corresponding i-th individuals (one per each agent-local population), where i is an index $\in [0, n_p - 1]$. This approach guarantees that each individual from each agent-local population is evaluated exactly once. Note that the selection operator, when applied, shuffles the array of the individuals. This means that an individual from an agent-local population is generally not always evaluated with the same individuals taken from the other agent-local populations.

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

6.2.2 Fitness evaluation

Once a team is created, it undergoes n_e episodes of simulation of the task. In the simulation phase, the agents perform IQL (with a dynamic ε-greedy exploration approach) to learn the function that maps the leaves to actions. By using IQL, each agent does not have to take into account the choices made by the other agents, as these are modeled as part of the environment. Moreover, given a sufficient number of episodes for the evaluation, the continuous learning of all the agents results in a co-adaptation. After the simulation phase, the seventh decile of the returns (i.e., the cumulative reward for each episode) received by an agent is used as fitness. The choice of the seventh decile lies on the fact that our fitness function is meant to describe the quality of a genotype as the quality of the state-space decomposition function [21], which can only be measured when the performance of the agent converges. While also the mean, the maximum, and the median have been considered as aggregation functions to compute the fitness, they have been discarded for the following reasons. Since the agents initially use a high ε for the exploration, the initial returns have a significant impact on the mean, thus they do not reflect the true quality of the genotype. Using the median would also present problems: on the one hand, the median would discard all the episodes in which the cooperation between the agents was fruitful enough to receive high returns; on the other hand, since we expect the returns to grow towards the end of the simulation phase, using the median would mean that we take into account the performance of a not-fully-trained agent. Finally, if we used the maximum to aggregate the returns, we would give too much importance to spurious good performance that may occur in the simulation (e.g., returns obtained just by randomly effective behaviors), without taking into account the performance of the trained version of the agent. In a preliminary experimental phase, the seventh decile represented a good trade-off between the median and the maximum, reflecting more closely the performance of the agents. The fitness evaluation process is described in Figure 6.1.

6.2.3 Operators

Besides the selection, mutation, crossover, and replacement operators used in Chapter 3, we also make use of an “adoption” mechanism, which allows individuals to be transferred from one population to the other.

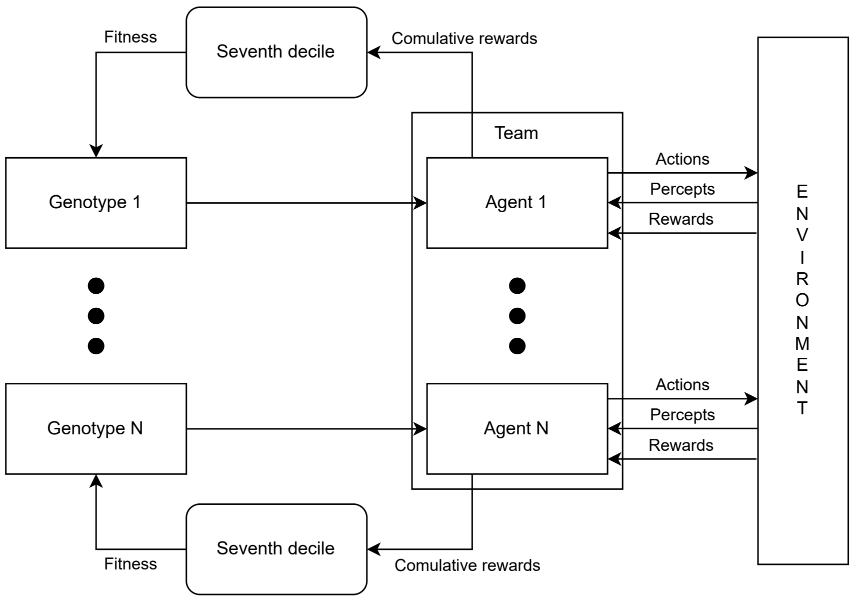


Figure 6.1: Block diagram of the fitness evaluation process.

The adoption of an individual happens at the end of each generation. An agent-local population is randomly chosen and the individual with the highest fitness is selected. At this point, the selected individual is copied into the other agent-local populations, replacing a randomly selected individual from the offspring. The adopted individual's parents are then assigned to the replaced individual's parents. The reason why we use this adoption mechanism lies in the reward system of the specific BattleField environment (see Section 6.2.4). As mentioned by the authors of PettingZoo: "Agents are rewarded for their individual performance, and not for the performance of their neighbors, so coordination is difficult"[1]. This means that only agents capable of hitting or killing enemies (this will become clearer in the next section) obtain a high fitness and the adoption mechanism allows sharing "knowledge" across agent-local populations.

6.2.4 Experimental setup

Environment

We simulate a multi-agent environment by using the PettingZoo library [143]. More specifically, we use the Battlefield environment from the MAgent [160] suite. A screenshot of the environment is shown in Figure 6.2.

In this task, there are two teams: the red team and the blue team. As the name of the environment suggests, the goal of each team is to defeat the other team by killing all

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

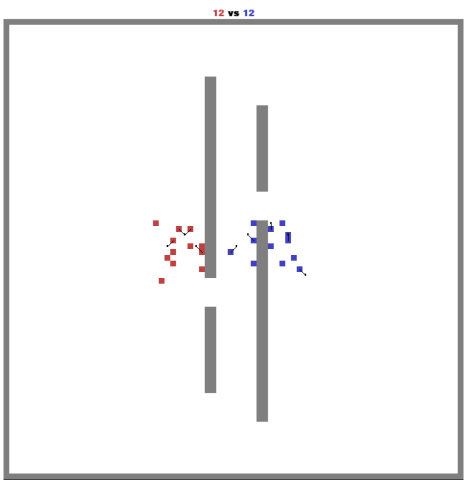


Figure 6.2: A screenshot from the Battlefield environment.

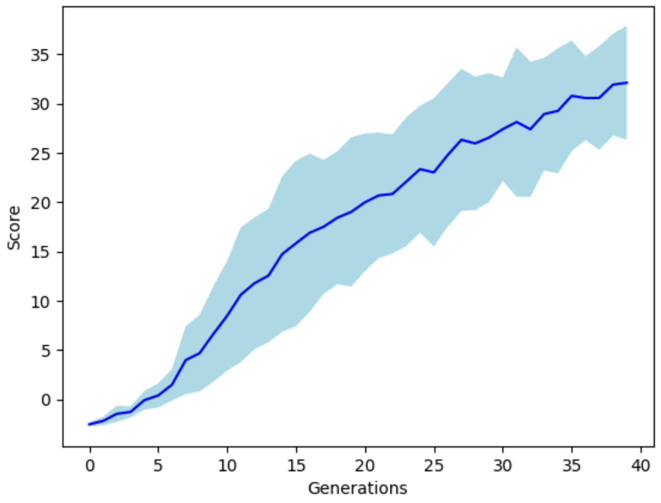


Figure 6.3: Maximum return (blue line: average, shaded area: std. dev.) at each generation.

of its members. The environment is an 80×80 grid. To win the battle, the agents have to learn to collaborate with their team in order to eliminate the enemies, and to move through the map to overcome walls and obstacles.

Each agent has a perceptive field of 13×13 squares and can either move or attack at each turn. The agents' perception is composed of: local presence/absence of an obstacle in a square; local presence/absence of a teammate/enemy in a square; health points (hp) of the teammate/enemy in a square; global density of teammates/enemies. A square represents a 7×7 quadrant of the environment. Note that each agent's local perception area corresponds to a circle with a radius of six squares around that agent. Moreover, to simplify the learning phase (and the interpretability of the agents evolved), we perform a pre-processing of these features, based on domain knowledge, in order to obtain higher-level features that are then fed as inputs to the decision tree. The selected features, extracted from the raw observations, are reported in Table 6.1. The "Abbreviation" column shows the abbreviation that we will use throughout the text to refer to a specific feature.

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

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

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

Table 6.1: Extracted features, their abbreviation and their domain.

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

Table 6.2: Actions that the agent can perform.

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

the 8 adjacent squares.

Table 6.2 shows the action that can be performed by the agent. As in Table 6.1, the "Abbreviation" column shows how we refer to the actions in the remainder of the text. The rewards obtained by the environment are the following: 5 points if the agent kills an opponent; -0.005 points for each timestep (a time penalty, thus the quicker the team wins, the higher the reward); -0.1 for attacking (to make the agent attack only when necessary); 0.9 when the agent hits an opponent (to give a quicker feedback to the agent, without having to wait for killing an agent to obtain a positive reward that encourages hitting enemies); -0.1 if the agent dies. At each timestep, the agent receives a combination of these rewards based on the events that happened in the last timestep. For instance, if an agent attacks and hits an enemy, it obtains a total reward of $r = 0.9 - 0.1 - 0.005$.

While there is no reward for collaboration, we decided not to alter the reward function to encourage it, to preserve the original configuration of the environment. Note that we evolve only one of the two teams (the blue one), while the other team (the red one) uses a random behavior for all the agents. This choice has been made in order to provide a non-biased baseline policy, i.e., to prevent the evolved policies from overfitting to a specific handmade policy for the red team. Furthermore, we decided not to competitively co-evolve the policies for both teams (blue and red) to reduce the complexity of the evolutionary process, and focus on the interpretability of the evolved policy for the blue team. For each fitness evaluation, n_e episodes are simulated, each of 500 timesteps.

Parameters

The parameters used for GE and Q-learning are shown in Table 6.3. To ensure that the Q function tends to the optimal one, we employ a learning rate of $\alpha = \frac{1}{v}$, where v is the number of visits made to the state-action pair [139]. The grammar for the GE algorithm is shown in Table 6.4. Note that we constrain the grammar to evolve orthogonal decision trees, i.e., decision trees whose conditions are in the form $x < c$, where x is a variable and c is a constant.

6.3 Results

We perform 10 independent evolutionary runs to evolve the policy of each agent in the blue team. Figure 6.3 shows the average maximum return (across the 10 runs) during the evolutionary process generation. The shaded area indicates the standard deviation across runs. We should note that while the average trend did not reach yet a plateau after the considered number of generations, we had to limit the total duration of our runs due to constraints on the available computational resources. On average, one full run of our approach takes approximately 30 hours on a 16-core machine with parallelization at the level of team evaluation. Given the cost of each run, we constrain the duration of each run, which reduces the probability of matching state-of-the-art scores [17].

Since the goal of the task is the elimination of the opponent team, we use two metrics to analyze the results in a post-hoc test phase (i.e., after the evolutionary process): the number of opponents killed, and the agents' returns over 100 unseen episodes. Table 6.5 shows the results of this test phase. For each of the 10 evolutionary runs, we report the

Table 6.3: Hyperparameters used for the two algorithms (Grammatical Evolution and Q-learning) used in the experimentation.

Algorithm	Parameter	Value
Grammatical Evolution	n_p	60
	n_g	40
	p_x	0.4
	p_m	0.8
	p_c	0.05
	s_i	500
	s_t	3
Q-learning	α	$1/v$
	ε_0	1
	n_e	400
	ε_k	$0.99^k\varepsilon_0$

Table 6.4: Grammar used to evolve the decision trees. "|" denotes the possibility to choose between different productions; "dt" indicates the start symbol.

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

statistics obtained with a team composed of the best agents (one for each population) evolved in that run over unseen episodes. The "Team kills" row shows the descriptive statistics of the number of enemies killed in each episode. Note that a team is formed by 12 agents therefore in a single episode the number of enemies killed is limited between 0 and 12. The "Agents' returns" row shows the descriptive statistics of the average returns of all the agents in the team. The "Completed" column shows the percentage of episodes in which the team was able to eliminate the entire opponent team.

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

6.3.1 Interpretation

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

Figure 6.4 shows the decision tree of one of the agents evolved in one of the evolutionary runs presented before. For space reasons, we cannot present all the evolved agents from each run. However, similar considerations apply also to the other evolved agents in the various runs.

By reading the decision tree in the figure, we can describe how the agent moves in

Table 6.5: Summary of the test results (co-evolutionary approach).

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

the environment. In the following, please remember that we evolve only the blue agents' behavior, and that these agents always start on the right side of the environment (see Figure 6.2). To facilitate the description of the evolved policy, we added an id to each node in the decision tree.

The selected agent moves up to the left (id 24) until the local density of enemies below (id 6) or to the right (id 18) reaches a certain threshold. In both cases, the agent changes the direction and moves toward the enemies (ids 11 and 21). It also moves to the right (id 25) if there is a high global density of enemies on its right (id 22). This means that this agent moves to the top left of the map and intercepts the enemies it finds in that area. Another interesting behavior of this agent lies in the fact that it tries not to be on the front lines. In fact, if there is a high density of allies in the same quadrant (id 14) it tends to move to the right (id 17), therefore from the direction from which its team started. This agent appears to behave like a "wing": it moves above the enemies and tries to eliminate the ones that try to move in the space between it and the allies below.

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

Other interesting behaviors emerge from the observation of the teams in the environment. A common behavior of the agents starting closer to the opponent team is to go through the gap in the walls to reach the enemies. There are also more complex behaviors. In some runs it is possible to see some agents moving to the top of the environment, passing the walls from above, and then descending to hit the enemies they encounter. Much rarer is the reverse behavior, where agents pass the walls from below and then move up.

6.3.2 Comparison with a non-co-evolutionary approach

To provide a baseline for the proposed co-evolutionary approach, we also performed experiments in the same environment using a single phenotype for the entire team, i.e., by cloning the phenotype and assigning it to each member of the team. The parameters used in these experiments are the same as Table 6.3. Note that in this case each agent in the team shares the same decision tree structure, but each one develops its own leaves by using IQL.

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

Table 6.6 shows the test results obtained by the agents evolved in each of 10 runs over 100 unseen episodes. We can compare these results with the ones obtained in the co-evolutionary setup (shown in Table 6.5) by using the number of kills at test time. In this regard, we observe that there is a large difference in performance between the two setups, with the co-evolutionary setup largely outperforming the non-co-evolutionary

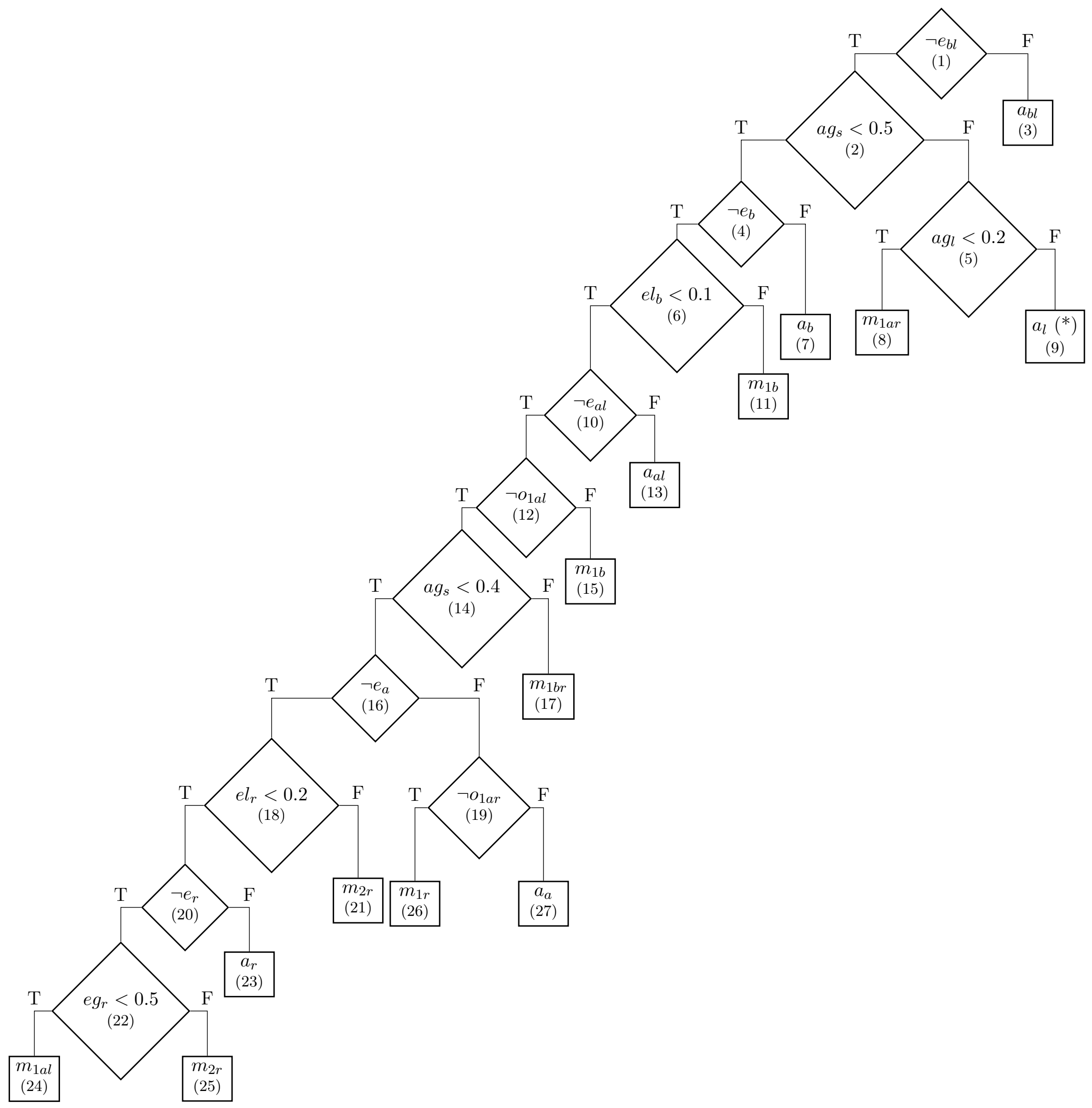


Figure 6.4: Decision tree of the selected agent. The "(*)" notation indicates that the leaf has been visited a number of times that is not sufficient to train it, thus it can be seen as a random action. The numbers in parentheses are the identifiers of the nodes.

approach. This indicates that, even though the agents have similar goals in both setups, the co-evolutionary setup can indeed find much better solutions. This may be due to the fact that the adoption mechanism used in the co-evolutionary approach allows for a quicker spreading of high-performing genotypes in the populations.

Another observation concerns the completion percentage: by looking at it, it appears that the performance of the non-co-evolutionary approach has higher variance across runs. This suggests that the performance of this setup is heavily impacted by the initialization, with only a few occasional runs achieving a satisfactory completion percentage. A possible improvement of the non-co-evolutionary setup would be to include an ad hoc method, e.g.

based on domain knowledge, to provide a smarter initialization.

Table 6.6: Summary of the test results (non-co-evolutionary approach).

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

6.4 Conclusions

In this chapter, we proposed a co-evolutionary approach to interpretable RL in MARL settings. We do so by extending the approach proposed in Chapter 3 in two ways: a non-co-evolutionary approach and a co-evolutionary approach. The experimental results show that the co-evolutionary approach is able to achieve significantly higher performance while keeping the same computational cost.

Future work includes: 1) evaluating the proposed approach on different tasks; 2) introducing the possibility of communication between agents (both symbolic [35] and sub-symbolic [73]); 3) designing more efficient methodologies for training interpretable MARL systems, including, for instance, using other RL algorithms (different from Q-learning), and comparing them with existing methods, as well as handmade problem-specific policies;

4) performing a sensitivity analysis for the proposed method; and 5) designing more efficient approaches that allow to evolve policies with lower computational cost.

Chapter 7

Applications

In this chapter, we will show two potential applications for the methods described in the past chapters. Both of them are focused on COVID-19, and aim at containing the damages of the pandemic from two different points of view:

- Reducing the number of infections by means of social restrictions
- Maximizing the efficiency of hospitals by automatically assessing the severity of patients' lungs

More specifically, in Section 7.1 we will show how the methodology proposed in this thesis can be used to obtain policies that help to decide the type of restrictions to contain a pandemic. Moreover, in Section 7.2, we will also show how the methodology can be used to augment black-box models to improve their performance and give insights into their inner functioning.

7.1 IRL for containing pandemics

Based on: Custode, Leonardo Lucio, and Giovanni Iacca. "Interpretable AI for policy-making in pandemics." Proceedings of the Genetic and Evolutionary Computation Conference Companion. 2022.

7.1.1 Introduction

COVID-19 has changed the way of living of all the people on Earth. In fact, since its outbreak, several countries adopted safety measures such as social distancing, lockdowns, and closing of certain types of economic activities and others. These safety measures were taken to slow down the spreading of the pandemic in the world population. However, some safety measures may significantly impact the economy of a country. For this reason, it is important to find a trade-off between the spreading of the pandemic and economic losses.

To this end, previous works focused on the use of simulators to estimate the virus propagation and economic losses given a policy [56, 145]. In this setting, a policy decides the safety measures to adopt in a given scenario. However, while these works have been proven to be able to find policies that have better trade-offs between economic losses and pandemic spread than the ones used by governments, they do not employ interpretable AI methods. Thus, even though such models do perform very well, their applicability is limited due to the lack of understandability. More specifically, the main drawbacks of non-interpretable approaches for this task are: a) the fact that the trained models act as "oracles" and, thus, a policymaker cannot understand the rationale underlying a decision; and b) the fact that such models cannot be easily inspected and, thus, their behavior cannot be formally specified.

In this chapter, we propose an approach that produces interpretable models for policy-making in pandemics. Our approach uses the methodology presented in Chapter 3 to produce DTs that are very simple and objectively interpretable. Moreover, our solutions exhibit better performance w.r.t. non-interpretable state-of-the-art models. Since our evolved policies are both more effective and more interpretable than existing black-box models, they are extremely suitable for creating policies to control the pandemic while avoiding unnecessary economic damage.

This chapter is structured as follows. The next section makes a short review of the state of the art. Section 7.1.3 describes the method used to evolve interpretable DTs. In Section 7.1.4 we show the experimental results and, in Section 7.1.5, we interpret the trees obtained. Finally, in Section 7.1.6 we draw the conclusions and suggest future work.

7.1.2 Related work

In [56], the authors propose a simulator of pandemics that can simulate a population at a fine granularity, modeling the activities each agent can perform, as well as various types of economic activities. The goal of this simulator is to test policies with the objective of minimizing simultaneously the spreading of the pandemic and the economic damages. The authors test various handcrafted policies, policies used from a few countries, and a deep-reinforcement-learning-based policy. The results show that the deep-RL-based policy is able to outperform both the handmade approaches and the governmental ones.

In [145], the authors propose a simulator to estimate the effects on the management of closing economic activities on both the spreading of the pandemic and the economic losses. The proposed simulator is tailored to the U.S. economy, simulating all 51 states and a central entity that manages subsidies.

The main difference between the approaches proposed above is that in [56] the simulator acts at a very small scale, but with a high level of detail, being able to simulate aspects of everyday life, while in [145] the simulator aims to perform at a very large scale with a lower level of detail.

In [84], the authors use a surrogate-assisted evolutionary process to optimize an agent that has to apply restrictions to avoid the spreading of the pandemic. They make use of two neural networks: one that acts as the policy, and the other that estimates the quality of a policy, trained on real data. While the performance of this approach is promising, the use of black-box policies such as neural network makes the real-world application of such systems difficult [116].

7.1.3 Method

Simulator

Since our goal is to evolve general policies for minimizing the spread of the pandemic and, at the same time, reducing economic losses, we employ the simulator proposed in [56]. In fact, this simulator allows us to test rules that are applicable in every country (i.e., not only tailored to the U.S. as in [145]) and that does not make use of economic subsidies.

This simulator allows us to simulate the pandemic of a small city, with the following properties:

- Population: 1000 people;
- Number of hospitals: 1, with a capacity of 10 people;
- Number of houses: 300;
- Number of grocery stores: 4;
- Number of offices: 5;
- Number of schools: 10;
- Compliance rate: 0.99.

The compliance rate is the probability that an individual, at each time step, will comply with the restrictions. This allows us to simulate an imperfect scenario, closer to the real world. Moreover, the state given to the policy is not exact. In fact, to resemble more closely a real-world scenario, the measurements are noisy (e.g., not all the exposed people may require a swab, and thus the number of infected may be higher than measured).

Furthermore, it is important to note that we are trying to simulate and mitigate the pandemic from the onset, which may prove difficult in the real world. Finally, the parameters of the SEIR model have been tuned in such a way that the epidemic resembles COVID-19.

State The simulator, at each step, provides the following features:

- i_g: Number of infected people since the beginning of the simulation;
- r_g: Number of recovered people since the beginning of the simulation;
- c_g: Number of patients in critical conditions since the beginning of the simulation;
- d_g: Number of dead people since the beginning of the simulation;
- h_g: Number of people that did not contract the virus from the beginning of the simulation until the current simulation day;
- i_d: Number of daily infected;
- r_d: Number of daily recovered;
- c_d: Number of patients that are in critical conditions in the current simulation day;
- d_d: Number of dead people in the current simulation day;
- h_d: Number of people that did not contract the virus in the current day[1];
- l: The current level of ongoing restrictions;
- h: A Boolean variable indicating whether the capacity of the hospitals is saturated.

All the variables are normalized by using a min-max normalization before being fed to the policy-making agent. Note that the simulator returns a noisy version of the estimates of the variables described above, to simulate a real-world scenario in which not all the results of the tests are known.

Actions The agent can choose an action between a pool of 5 actions:

1. Stage 0: No restrictions
2. Stage 1: Stay at home if sick, gathering limits:
    - Low-risk people: 50 people

[1]Please note that the simulator always returns $h_d = h_g$. However, for consistency with the experiments reported in [56], in our experiments we kept both values as inputs to the policy-making agent.

- High-risk people: 25 people

3. Stage 2: Limitations of stage 1 + wear masks + light social distancing, schools, and hair salons are closed, gathering limits:
    - Low-risk people: 25
    - High-risk people: 10
4. Stage 3: Limitations of stage 2 + moderate social distancing + no gatherings
5. Stage 4: Limitations of stage 3 + heavy social distancing + office and retail stores closed

A more detailed list of actions is described in [56].

Rewards At each step, the agent receives a reward of:

$$r = -0.4 \cdot max(\frac{c_d - C}{C}, 0) - 0.1 \cdot \frac{l^{1.5}}{5^{1.5}}, \tag{7.1}$$

where C is the capacity of the hospitals.

Thus, the reward function aims to trade off the number of patients in critical conditions with the stringency level of the restrictions. Moreover, this function indirectly induces the agent to minimize also the closing of stores, limiting the economic losses.

Evolution of decision trees

We employ the same setup used in Table 3.4, except for the learning rate, which was set to $\alpha = 10^{-3}$.

Fitness evaluation The fitness of each phenotype is determined by using the corresponding policy to perform decisions in the `PandemicSimulator`[2] environment.

7.1.4 Results

Table 7.2 shows the scores obtained by the best agents obtained in each independent run. Here, we differentiate between *training* and *testing* returns. Training returns are defined as the returns obtained by the agent in the training process, while testing refers to the returns obtained by the agent after the evolutionary process, i.e., when learning is disabled.

[2]https://github.com/SonyAI/PandemicSimulator

Table 7.1: Grammar used to produce the decision trees.

Rule	**Production**
dt	$\langle if \rangle$
if	$if\ \langle condition \rangle\ \ then\ \langle action \rangle\ \ else\ \langle action \rangle$
condition	$input\_var\ \langle comp\_op \rangle\ \ \langle const_{input\_var} \rangle$
action	$leaf \mid \langle if \rangle$
comp_op	$lt \mid gt$
$const_{1..10}$	[0, 1) with step 0.1
$const_{11,12}$	[0, 1) with step 0.5

Figure 7.1, instead, compares the results obtained with the best DT produced by our method (i.e., the one corresponding to seed 3 in Table 7.2, which obtained the highest test mean return) to the policies reported in [56]. In particular, the policies used for the comparison are the following:

1. S0-4-0: Starts with stage 0, then, once 10 have been infected, it switches to stage 4 and, after 30 days, it returns to stage 0.
2. S0-4-0FI: Similar to S0-4-0, but the return from stage 4 to stage 0 is performed gradually, by reducing the restrictions by 1 stage every 5 days.
3. S0-4-0GI: Similar to S0-4-0, but the intermediate stages from stage 4 to stage 0 last 10 days instead of 5.
4. S0: Always applies stage 0 restrictions.
5. S1: Always applies stage 1 restrictions.
6. S2: Always applies stage 2 restrictions.
7. S3: Always applies stage 3 restrictions.
8. S4: Always applies stage 4 restrictions.

From Figure 7.1, we can see that our best DT outperforms all the hand-crafted policies under comparison (note that the negative reward is to be maximized). The statistical difference between the best decision tree evolved and the hand-crafted policies have been confirmed by a two-sided Wilcoxon test with $\alpha = 0.05$.

As for the deep-RL-based policy presented in [56], while we could not test it due to the fact that the model is not publicly available (and neither its numerical results are), we estimate, from the plots reported in the original paper, a cumulative reward of about -5, which is substantially lower than the cumulative reward obtained by our best DT.

Finally, in Figure 7.2 we compare the policy obtained by means of our best DT with the ones implemented by the Italian and Swedish governments (for which we use the implementation provided in [56]). These policies act as follows:

1. ITA (approximation of the restrictions adopted by the Italian government): Increase the restrictions gradually from stage 0 to stage 4 and then gradually returns to stage 2.

2. SWE (approximation of restrictions adopted by the Swedish government): Applies stage 0 regulations for 3 days and then stage 1 restrictions for the duration of the simulation.

Also in this case, we observe that the performance is significantly better than the compared policies. Similarly, also in this case, a Wilcoxon test with $\alpha = 0.05$ confirms the statistical significance of the results.

Moreover, we observe that the number of people infected, by using our proposed policy, is significantly smaller w.r.t. the other approaches. This fact, combined with the very high rewards obtained by our policy, suggests that the policy minimizes the economic losses by trying to stop the pandemic at the beginning and then making the restrictions less stringent.

In the next section, we analyze the policy to understand the reasons underlying its significantly higher performance.

Table 7.2: Results obtained by running our method in 10 i.i.d. experimental runs.

Train mean return	**Test mean return**	$\mathcal{M}$
-1.82 ± 0.29	-1.67 ± 0.47	40.94 ± 14.66

7.1.5 Interpretation

The best DT obtained is shown in Figure 7.3. While the size of the DT is quite limited, its performance is extremely satisfactory. We can see that the DT makes use of two conditions to compute its output.

The first condition (the root) checks whether the number of never-infected people (h_d) is greater than 90%. If so, then it checks the second condition. Otherwise, it applies stage 2 restrictions, regardless of the other variables. We may hypothesize that, in the latter case, the policy does that because if the number of infected people (from the beginning

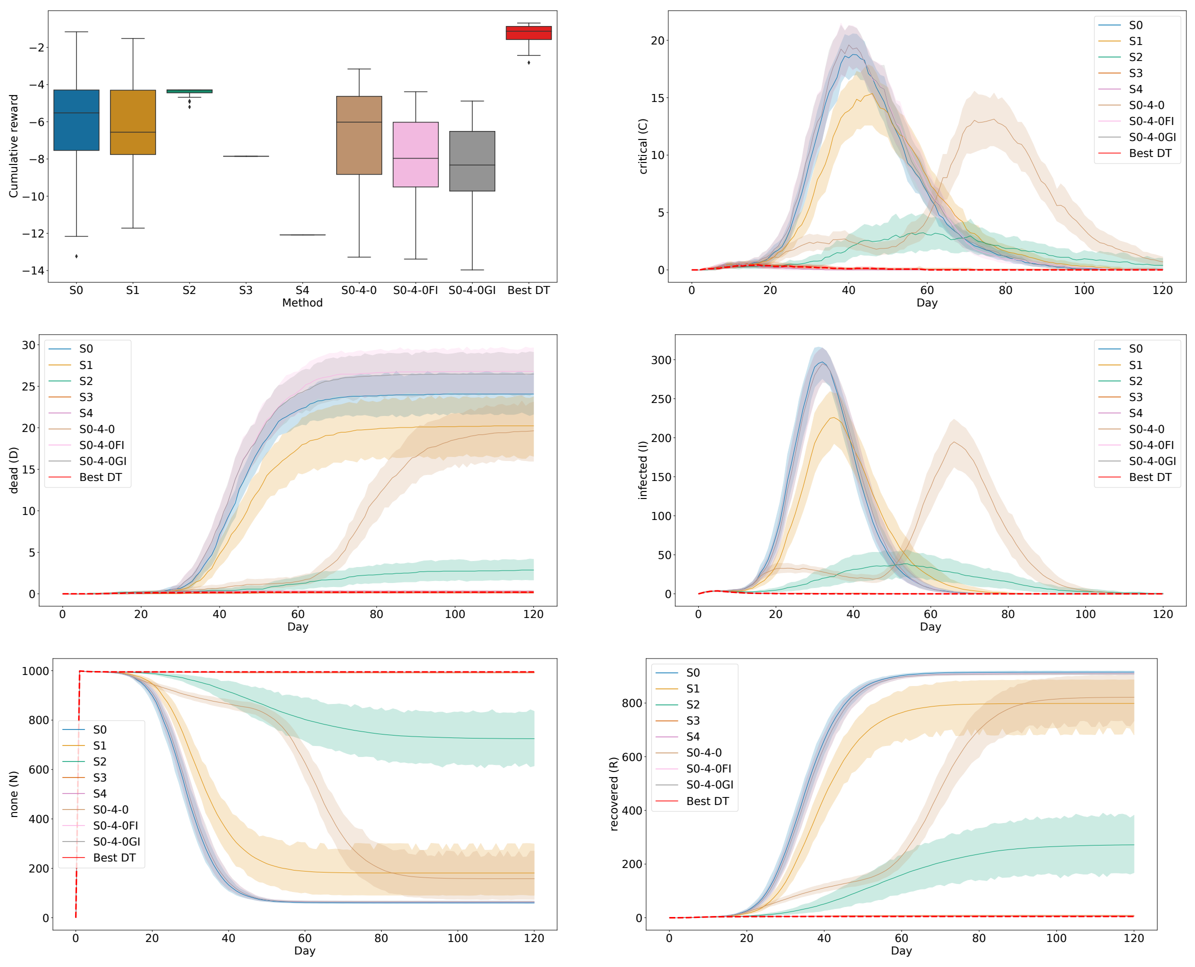


Figure 7.1: Comparison of the performance of the best DT evolved w.r.t. handcrafted policies proposed in [56].

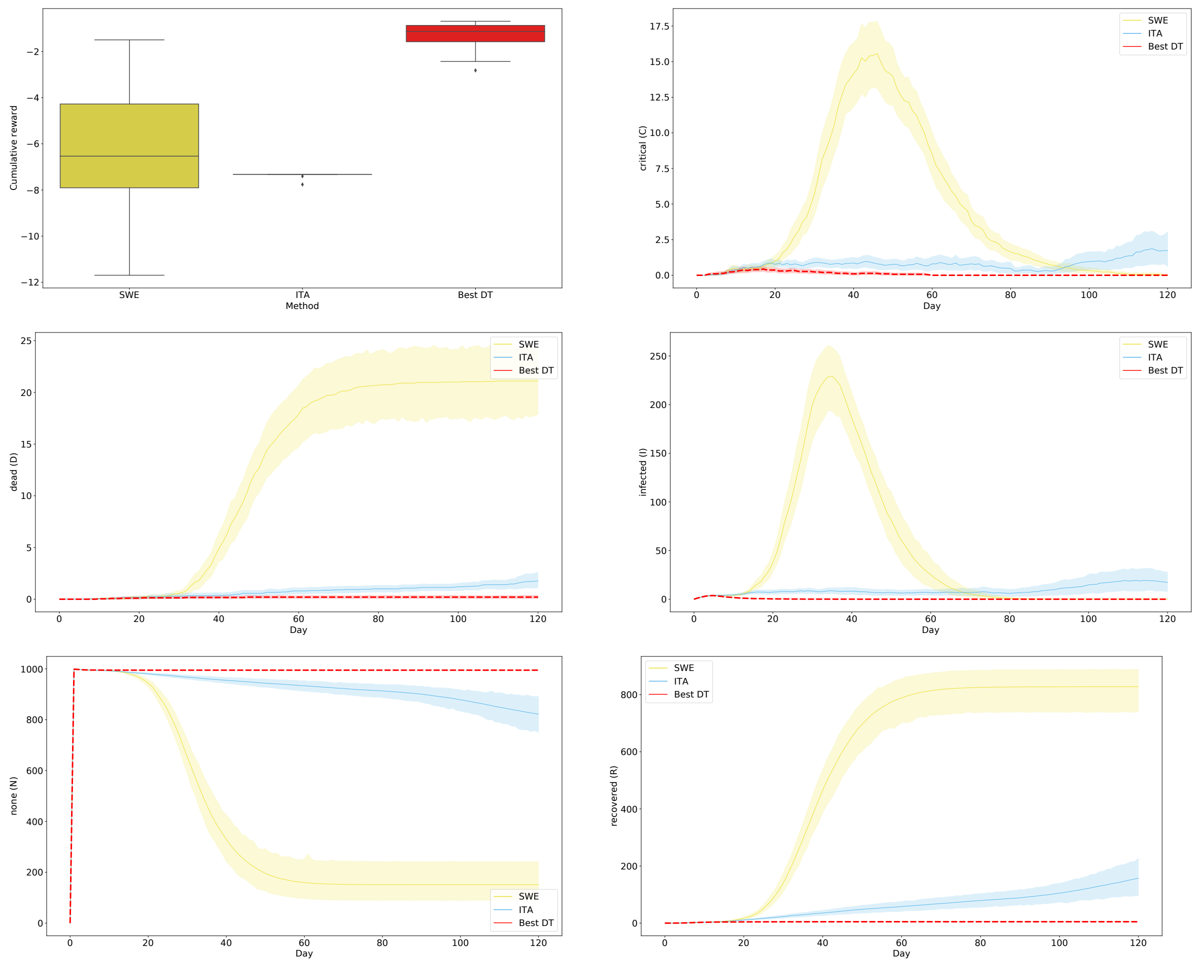


Figure 7.2: Comparison of the performance of the best DT evolved w.r.t. policies adopted by the Swedish (SWE) and Italian (ITA) governments.

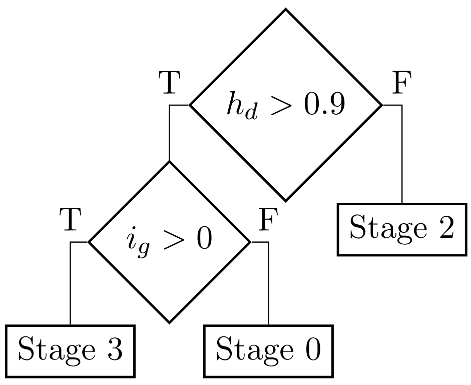


Figure 7.3: Best decision tree evolved (on the test score).

of the pandemic) is greater than the 10%, then the virus may spread very quickly if the restrictions are not appropriate.

The second condition, instead, checks the number of people infected so far (i_g). If the known number of infected is greater than zero, it applies stage 3 restrictions. Otherwise, it applies no restrictions.

In essence, the combination of the two branches of the root aims to induce a "strong" response to the initial wave of the pandemic, which is then slightly relaxed after the number of total cases increases. Thus, the overall strategy learned by the agent is to stop the pandemic at the beginning, keeping slightly less stringent restrictions after the initial wave, to avoid new waves.

It is important to note that the DT obtained depends highly on the scenario simulated. In fact, we expect that changing the parameters of the epidemic or the initial state (e.g., having significantly more infected people) can significantly change the outcome of the optimization process, both in terms of DT and in terms of score.

7.1.6 Conclusions

In this section, we leveraged interpretable AI methodologies to train interpretable policies for managing a pandemic.

Despite the widely-thought trade-off between interpretability and performance, the obtained policies proved to be significantly better than handcrafted policies, deep learning policies, and government policies.

It is important to note that the results shown above have been tested in a simulated scenario and, thus, their applicability to real-world scenarios must be assessed.

A limitation of the current work is the simulation scenario adopted: here, we simulate a population of 1000 people. The assessment of the scalability of our results is left as future work.

Other future works include: testing the proposed methodology on different simulators with different objectives, and adopting more realistic scenarios w.r.t. economical aspects (e.g., subsidies) or epidemiological aspects (e.g., the possibility of the emergence of new variants).

7.2 Neuro-Symbolic Analysis of COVID-19 Patients' Data

Based on: Custode, Leonardo Lucio, et al. "Multi-objective automatic analysis of lung ultrasound data from COVID-19 patients by means of deep learning and decision trees." Applied Soft Computing 133 (2023): 109926.

7.2.1 Introduction

Since the outbreak of the coronavirus disease 2019 (COVID-19) pandemic, the use of lung ultrasound (LUS) has been globally and fastly spreading. Indeed, the main advantages of LUS (portability, cost-effectiveness, real-time imaging, and safety) compared to other imaging technologies such as, e.g., Computed Tomography (CT), allowed LUS to be widely adopted to evaluate the state of lungs in patients affected by COVID-19 [135, 136, 101, 72, 90, 157, 152, 97, 30]. Moreover, LUS can be nowadays used for patients' monitoring and for the triage of symptomatic patients [135]. In particular, LUS is often exploited to detect COVID-19-associated interstitial pneumonia and follow its evolution [27, 136]. To perform this task, different imaging protocols have been proposed together with semi-quantitative scoring systems [1]. Indeed, even though quantitative approaches aiming at assessing the condition of lung parenchyma with ultrasound are emerging [83, 81, 27, 134, 86, 159], these strategies are not available for emergency contexts, due to their current preliminary state. Therefore, semi-quantitative scoring systems based on specific LUS imaging patterns (e.g., vertical and horizontal artifacts, or consolidations) have been extensively exploited during the pandemic [136].

Even though an LUS quantitative analysis cannot be performed with the currently available technologies, the use of artificial intelligence (AI) for the classification of LUS frames according to a semi-quantitative scoring system can be exploited to reduce subjectivity in the evaluation and to reduce the time required to perform the analysis [115, 18, 155, 40].

Here, we exploit a standardized imaging protocol based on 14 scanning areas and on a four-level scoring system, which allows the grading of the state of lungs [136]. A recent study demonstrated how this standardized protocol and scoring system have a prognostic value when evaluating the cumulative score (sum of scores obtained in the 14 scanning areas) at exam-level [100]. We acquire 1808 LUS videos from 100 COVID-19-positive patients, which consist of 366,301 frames in total. These frames are then fed to two DNNs [115] that were previously trained to perform automatic scoring and segmentation of LUS frames according to the above-mentioned four-level scoring system [136]. We successively use the scores given as output by two DNNs (respectively for segmentation and labeling) [115] to train and test a novel automatic approach, based on decision trees (DTs) automatically synthesized by evolutionary computation, aiming at passing from frame-based labeling to video-based labeling. Specifically, we compare the video-level scores given by our automatic approach with scores given by expert clinicians. Indeed, to perform their evaluation, clinicians associate a score to each video rather than to each frame. We then assess the performance of our aggregation approach (both at video level and exam level) by comparing the results obtained by the proposed method with the empirical aggregation technique previously reported in [82], which represents the current state-of-the-art. We hypothesize that, even though this existing technique achieves good performance, the fact that its decisions are obtained by aggregating the outputs of the DNNs by means of a simple threshold-based approach may be sub-optimal. To overcome this limitation, we instead use a fully data-driven DT-based approach, that is in principle more flexible and does not require empirical choices of thresholds. To summarize, the main contributions of this work are the following:

1. we propose a neuro-symbolic approach to the automatic scoring of COVID-19 patients by combining DNNs and interpretable DTs;
2. we compare single-objective and multi-objective evolutionary approaches to synthesize DTs optimized w.r.t. three different metrics of interest;
3. we interpret the evolved DTs to understand their decision policies;
4. we obtain decision support systems that have both higher prognostic agreement and less variance w.r.t. the approach previously proposed in the state-of-the-art work in the field [82].

This section is organized as follows. Next subsection describes the data and the method used for this study. Section 7.2.3 presents the experimental results. In Section 7.2.4, we analyze the DTs produced, and, finally, in Section 7.2.5, we draw the conclusions.

7.2.2 Method

We use the two models from [115] as feature extractors, whose outputs are aggregated and given in input to an evolved DT, which will then make a prediction of the score related to the video. A block diagram of the process is shown in Figure 7.4.

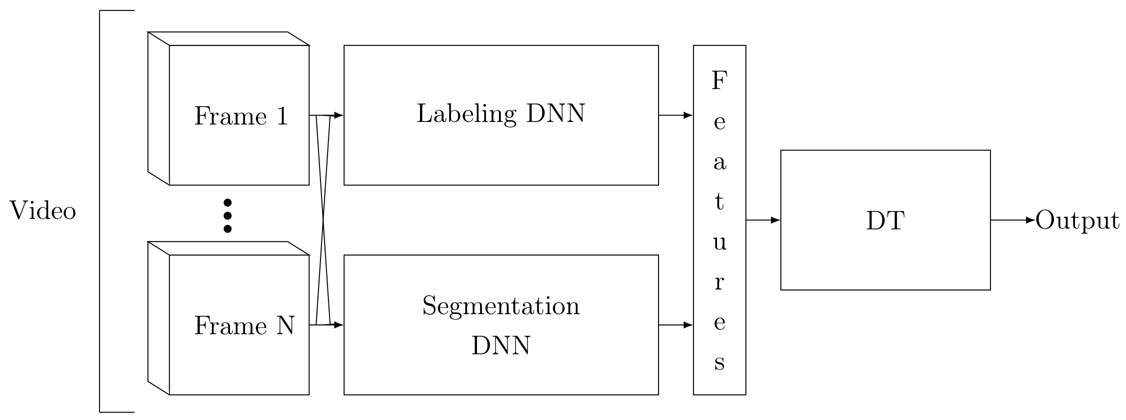


Figure 7.4: Block diagram of the prediction process.

Data

The investigated population consists of 100 patients diagnosed as COVID-19 positive by a reverse transcription polymerase chain reaction (RT-PCR) swab test. Of the 100 patients, 63 (35 male, 28 female; ages ranging from 26 to 92 years, and average age equal to 63.72 years) were examined within the Fondazione Policlinico San Matteo (Pavia, Italy), 19 (16 male, 3 female; ages ranging from 34 to 84 years, and average age equal to 63.95 years) within the Lodi General Hospital (Lodi, Italy), and 18 (8 male, 10 female; ages ranging from 23 to 95 years, and average age equal to 52.11 years) within the Fondazione Policlinico Universitario Agostino Gemelli (Rome, Italy). As a subgroup of patients was examined multiple times, on different dates, a total of 133 LUS exams were performed (94 at Pavia, 20 at Lodi, and 19 at Rome). A total of 1808 LUS videos were thus acquired (1,290 at Pavia, 276 at Lodi, 242 at Rome), which consist of 366,301 frames (292,943 at Pavia, 44,288 at Lodi, 29,070 at Rome).

The data from Pavia have been acquired using a convex probe with an Esaote MyLab Twice scanner, and an Esaote MyLab 50, setting an imaging depth from 8 to 12 cm (depending on the patient) and an imaging frequency from 5.0 to 6.6 MHz (depending on the scanner). The data from Lodi have been acquired using a convex probe with an Esaote Mylab Sigma scanner, and a MindRay TE7, setting an imaging depth from 8 to 12 cm (depending on the patient) and an imaging frequency from 3.5 to 5.5 MHz. The data from Rome have been acquired using a convex probe with an Esaote MyLab 50, an Esaote MyLab Alpha, and a Philips IU22, setting an imaging depth from 8 to 12 cm (depending on the patient), and an imaging frequency from 3.5 to 6.6 MHz (depending on the scanner).

This study was part of a protocol that has been registered (NCT04322487) and received approval from the Ethical Committee of the Fondazione Policlinico Universitario San

Matteo (protocol 20200063198), of Milano area 1, the Azienda Socio-Sanitaria Territoriale Fatebenefratelli-Sacco (protocol N0031981), of the Fondazione Policlinico Universitario Agostino Gemelli, Istituto di Ricovero e Cura a Carattere Scientifico (protocol 0015884/20 ID 3117). All patients gave informed consent.

The patients were examined by applying a standardized acquisition protocol based on 14 scanning areas [136]. This protocol is based on a four-level scoring system consisting in assigning a score that ranges from 0 to 3, depending on the observed LUS patterns, with score 0 indicating a healthy lung surface, and 1, 2, 3 an increasingly altered lung surface [136]. All the 1808 LUS videos were thus scored by LUS medical experts (in this case, the authors T.P., F.T., and A.S.). Each expert labeled the videos acquired by himself, i.e., T.P. labeled videos from Pavia, F.T. from Lodi, and A.S. from Rome. The distribution of scores assigned at the video level by the experts is shown in Figure 7.5.

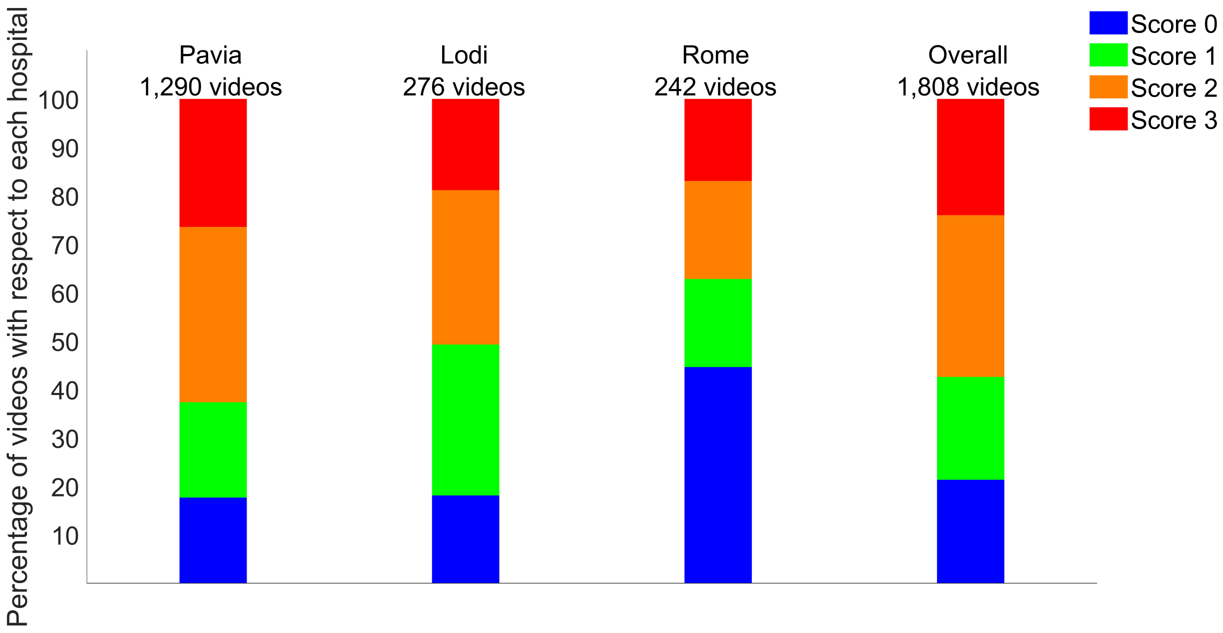


Figure 7.5: The distribution of scores assigned at video-level by the three clinical experts is shown. The percentage of scores 0, 1, 2, and 3 is shown for each hospital (Pavia, Lodi, and Rome) and for the entire dataset (overall). The total number of videos for each group is provided on top.

Inputs All the 1808 videos are fed to the two DNNs presented by Roy *et al.* [115], i.e., a labeling DNN derived from Spatial Transformer Networks and a segmentation DNN derived from U-Nets and DeepLab v3+. The former provides as output a score for each input frame, whereas the latter provided semantic segmentation and assigned one or multiple scores to each frame [115]. As the segmentation DNN can provide multiple scores for the same frame, we assign only the highest score predicted by this DNN to

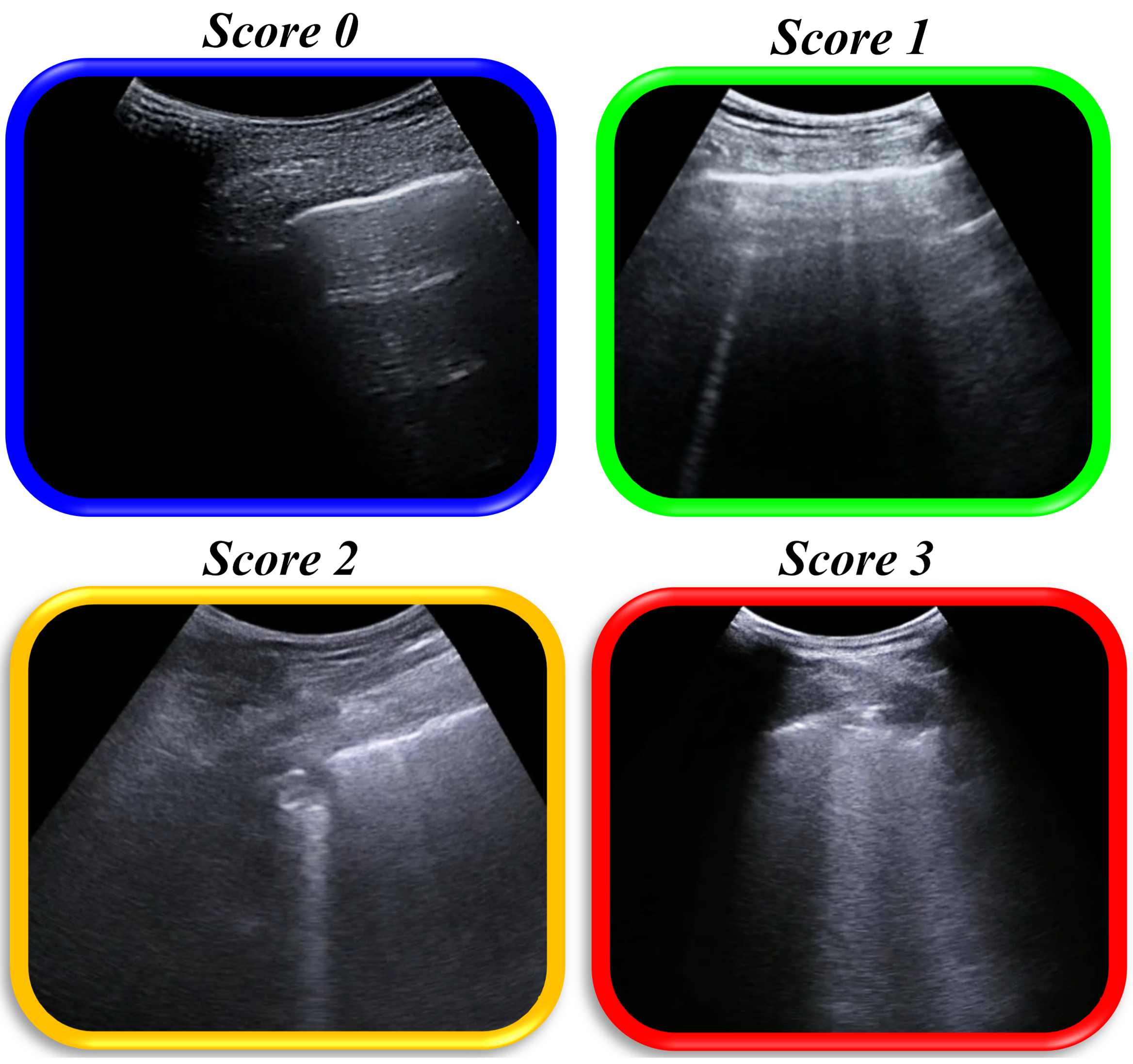


Figure 7.6: Examples of frames labeled as scores 0, 1, 2, and 3.

each considered frame (i.e., the worst-case score). Moreover, it is important to highlight how the segmentation DNN could provide no scores in output (if it does not find any relevant LUS pattern). Therefore, an extra score indicating the absence of LUS patterns (characteristic of the four scores) is considered when evaluating the output provided by segmentation DNN (we called it score -1). These two DNNs have been previously trained with the dataset presented by Roy *et al.* [115], which does not depend on the dataset exploited in the hereby work. Figure 7.6 shows examples of frames labeled as scores 0, 1, 2, and 3. Figure 7.7 shows the distribution of scores assigned at frame-level by the exploited two DNNs. Considering the entire dataset (see the overall distribution, Figure 7.7, right), there is a high percentage of scores 0 and 2 for both labeling and segmentation

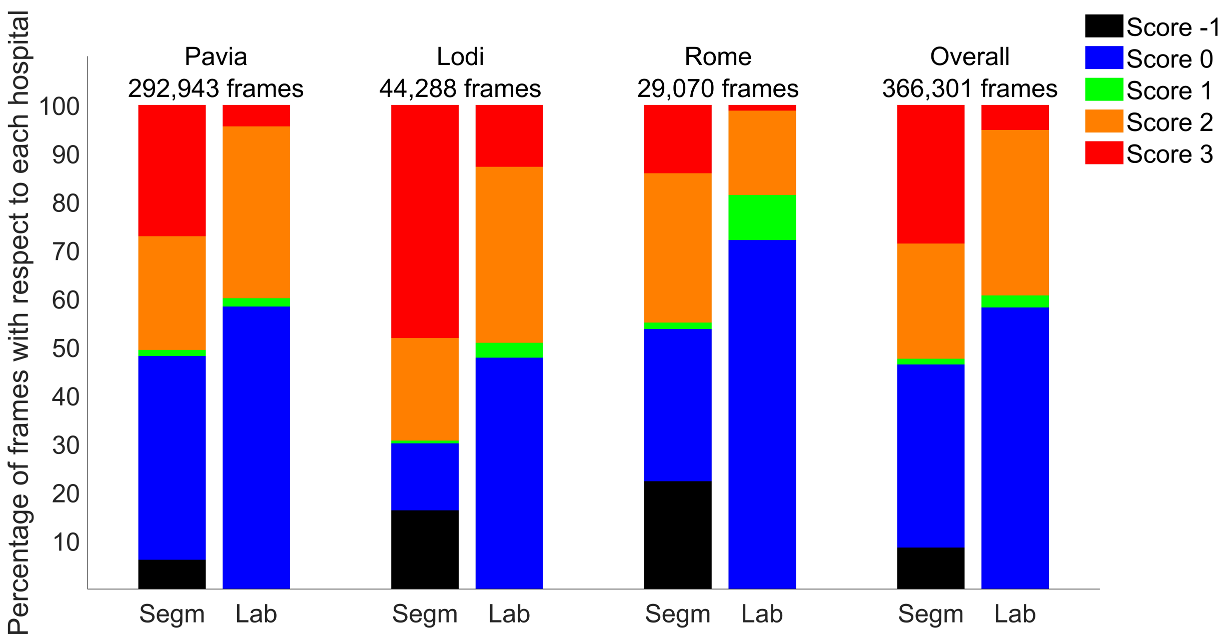


Figure 7.7: The distribution of scores assigned at frame-level by the labeling and segmentation DNNs presented in [115]), exploited in this work, is shown. The percentage of scores 0, 1, 2, and 3 are shown for each hospital (Pavia, Lodi, and Rome) and for the entire dataset (overall). The frame-level scores given by each architecture are shown separately. As the segmentation DNN can provide multiple scores for the same frame, we scored each frame with the worst-case (i.e., maximum) score predicted by the segmentation DNN. As the segmentation DNN could provide no output (if it does not find any relevant LUS pattern), an extra distribution represented as score -1 is observable in the left bars. The total number of frames for each group is provided on top.

DNNs, whereas score 1 is less frequently given as output. It is also observable how the percentage of score 3 is significantly higher when looking at the segmentation DNN.

Targets Given the frame-level labeling provided by the two DNNs, our target consists in finding an aggregation technique that allows us to pass from a frame-level score to a video-level score, which is the output needed by physicians to perform their clinical evaluation. Therefore, the goal of the proposed technique is to optimize three metrics of interest that compare the video-level scores obtained by our algorithm and the ones assigned by clinical experts (see Figure 7.5). These three metrics are: the video-level agreement, the exam-level agreement, and the prognostic-level agreement [82].

The video-level agreement consists of the percentage of videos that are correctly classified by the algorithm (i.e., the score assigned by the expert coincides with the score assigned by the algorithm) [82]. We also evaluate the video-level agreement when allowing a disagreement up to 1 point (e.g., if the algorithm classified a video as score 2 and the expert as score 1 or 3, the evaluation is correct) [82]. To distinguish these two video-level agreements, we denote the agreement characterized by an exact match between video-level scores as video-level agreement with a threshold (Th) equal to 0, whereas we refer to the video-level agreement allowing a disagreement up to 1 point as video-level agreement with Th equal to 1.

The exam-level agreement is instead computed by considering the cumulative score obtained by summing the video-level scores assigned to each of the 14 scanning areas [136, 82]. Specifically, we compute the exam-level agreement as the percentage of LUS exams (133 in total) having a cumulative score (ranging from 0 to $3 \times 14 = 42$) allowing a disagreement between algorithm and clinical experts of up to 2, 5, and 10 points (i.e., Th equal to 2, 5, and 10, respectively) [82]. To support the stratification between patients at high risk of clinical worsening and patients at low risk, we need to consider the prognostic value of the aforementioned protocol [136], which has been recently proven in a single-center study on 52 patients [100]. In particular, the patient is at low risk of clinical worsening when the exam-based cumulative score is less than or equal to 24, whereas the patient is at high risk of worsening when the exam-based cumulative score is greater than 24 [100].

We thus evaluate the algorithm capability of automatically stratifying these two categories of patients by measuring the prognostic-level agreement [82]. Specifically, clinical experts and algorithms are considered in prognostic agreement when both cumulative scores are less than or equal to 24 (low risk) or greater than 24 (high risk) (Th equal to 24) [82].

As we will show in detail in Section 12, it should be noted that we use a proxy for the first two metrics, i.e., the Mean Square Error (MSE), which gives us an advantage over optimizing directly the agreement. In fact, in [82] the authors use several tolerances for the video-level and exam-level agreement: this implies that in practical scenarios one should either use n objectives for each metric, where n is the number of tolerances (e.g., maximize the exam-level agreement with tolerances 2, 5, and 10); or, one should choose one tolerance among all the tolerances, which may lead to DTs that perform well only for that particular tolerance. Instead, using the MSE as we do here allows us to maximize simultaneously the agreement for all the tolerances.

Splitting of the data We split the data randomly into 5 folds. To prevent data leakage, we make sure that all the data belonging to a patient are assigned to the same fold. Moreover, we use 4 folds for the training phase (i.e., to compute the fitness of the individuals) and the remaining one to assess the generalization capabilities of the best-evolved DTs (i.e., as test set).

Feature extraction and aggregation of the outputs

The DT (as shown in Figure 7.4) expects as input video-level features. Instead, the two DNNs' input consists of single frames of each video. To convert the features from frame level to video level, we aggregate the outputs of each DNN. For the labeling DNN, we simply use as features the relative frequency of the prediction of each class (i.e., the argmax of the output vector of each frame). For the segmentation DNN, instead, we aggregate the features by computing the relative frequency of the worst-case (i.e., maximum) predicted classes inside each frame. This distinction between the two DNNs is needed since the segmentation DNN does not produce a single prediction but, instead, produces a mask for the frame taken in input. Moreover, we add to the feature vector also the minimum and maximum prediction made by the two DNNs.

The resulting feature vector is thus composed of 12 features: l_0, l_1, l_2, l_3, l_{min}, l_{max}, s_0, s_1, s_2, s_3, s_{min}, s_{max} where: l_i, $i = 0, 1, 2, 3$, represents the relative frequency of the prediction of the class i made by the labeling DNN; l_{min} and l_{max} represent the minimum and the maximum gravity level predicted by the labeling DNN; s_i, $i = 0, 1, 2, 3$, represents the number of cases in which the class i corresponds to the worst-case in the predictions made by the segmentation DNN; s_{min} and s_{max} represent the minimum and the maximum gravity level predicted by the segmentation DNN. Note that, while l_{min} and l_{max} range in $[0, 3]$, the segmentation DNN can also detect the absence of LUS patterns (i.e., the pixel is assigned a score of -1). For this reason, s_{min} and s_{max} range in $[-1, 3]$.

Evolutionary settings

We use Grammatical Evolution (GE) [120] to evolve programs that resemble DTs (i.e., they are based on an if-then-else structure). GE is an evolutionary algorithm that allows the evolution of grammars, encoded in the Backus-Naur form. It makes use of a genotype, which consists of a list of integers (called *codons*). When the genotype has to be evaluated, it makes use of a translator, which allows to convert the grammar to the corresponding phenotype. The grammar we employ is shown in Table 7.3.

We consider two GE settings, namely: 1) a single-objective one, in which we optimize either the video-level MSE, the exam-level MSE, or the prognostic-level agreement (see Section 12 for details on the three metrics); and 2) a multi-objective one, in which we optimize simultaneously all the three objectives stated previously.

The pseudo-code of the algorithm is shown in Algorithm 6. The algorithm consists in an initialization step (Line 1) followed by an evolutionary loop (Lines 4 - 10). The evolutionary loop starts with the evaluation of the population (Line 4), followed by the replacement of the individuals in the population (Line 5). Then, we perform the usual evolutionary steps, i.e., selection (Line 6), crossover (Line 7), and mutation (Line 8).

We should note that the GE algorithm we use here has some differences with respect to the original one described in [120]. First of all, we do not make use of a variable-length genotype but, instead, we fix its length (as shown in Table 7.4). Fixing the length of the genotype to small values constrain the resulting DTs to be small and, thus, more interpretable than the ones we can obtain by having longer genotypes. In fact, fixing the size of the genotype implicitly sets a maximum number of splits, maximum depth and maximum number of leaves in the DT. For instance, given the grammar shown in Table 7.3, the maximum number of nodes is $max_{nodes} = \lfloor \frac{s_i}{7} \rfloor$, the maximum number of leaves is $max_{leaves} = max_{nodes} + 1$, and $max_{depth} = max_{nodes} + 1$. Moreover, instead of using the genetic operators described in [120], we use traditional operators for genetic algorithms. The reason underlying this choice is the fact that, from preliminary experiments, the original operators seem to achieve worse performance than traditional genetic operators. For this reason, we employ standard operators, described below. Moreover, the operators proposed in [120] could not be used since they are specific to variable-length genotypes.

We make use of the replacement operator (Line 5) described in [24], which replaces a parent from the population only if there is an offspring that outperforms it. In case there are two offspring whose performance are better than only one of the two parents, then

Algorithm 6: Evolutionary process for the optimization of DTs

```
   Input: $s_p$: the size of the population
   Input: $g$: the number of generations
   Result: $pop$: The final population
1  $pop \leftarrow create\_population(s_p)$;
2  $old\_pop \leftarrow []$;
3  for $i = 0; i < g; i{+}{+}$ do
4      $fitnesses \leftarrow evaluate(pop)$;
5      $pop \leftarrow replacement(pop, old\_pop, fitnesses)$;
6      $parents \leftarrow select(pop, fitnesses)$;
7      $offspring \leftarrow crossover(parents)$;
8      $offspring \leftarrow mutation(offspring)$;
9      $old\_pop \leftarrow pop$;
10     $pop \leftarrow offspring$;
11 end
12 return $pop$;
```

the best offspring replaces the worst parent.

Moreover, we use of two different parent-selection operators (for Line 6), depending on whether we are working in the single-objective or multi-objective setting. In the single-objective setting, we use the "best-wise" selection operator, i.e., a selection operator that reorders the population by descending fitness such that, when performing crossover, the $(2i)$-th best mates with the $(2i+1)$-th best. Conversely, in the multi-objective setting, the selection operator we use is the NSGA-II [26] operator, which proved to work very well for multi-objective problems[3].

The crossover operator (Line 7), instead, is the one-point crossover, which produces two offspring from two parents by splitting their genotypes in a randomly chosen point and mixing the corresponding sub-strings obtained from the two parents.

While mutating a solution (Line 8), we employ a uniform mutation, which mutates each codon of the genotype according to a given probability p_{codon}. Its new value is sampled randomly from the possible values.

The parameters we use are presented in Table 7.4. The parameters shown in the table were obtained by manual tuning.

Table 7.3: Grammar used to evolve the DTs. The symbol "|" denotes the possibility to choose between different symbols. When using the grammar to translate a genotype into a phenotype, the rules are expanded in one of the possible choices listed in their production, depending on the value of the genotype.

Rule	**Production**
dt	$\langle if \rangle$
if	$if\ \langle condition \rangle\ then\ \langle output \rangle\ else\ \langle output \rangle$
condition	$\langle var \rangle \langle op \rangle \langle const \rangle \mid \langle var \rangle \langle op \rangle \langle var \rangle$
var	$\{input_i\}$; $i \in [0, 12[$
op	$< \mid > \mid ==$
output	$0 \mid 1 \mid 2 \mid 3 \mid \langle if \rangle$
const	$[0, 1]$ with step 10^{-2}

[3]Since the most "important" metric is the prognostic-level agreement, an attentive reader may point out that using Pareto optimization may not be the ideal choice in this case. However, we cannot use a lexicographic selection, since this would require specifying a *preference* also between the exam- and the video-level MSE while, in this case, we do not have a clear preference. Moreover, using a weighted sum of the three objectives is also not feasible, since this would require assigning a specific weight to each of the three objectives. For these reasons, we optimize the objectives using Pareto optimization and, then, we select the best solution according to the prognostic-level agreement.

Table 7.4: Hyperparameters used for the Grammatical Evolution algorithm.

Parameter	**Value**
Pop size	1000
Generations	1000
Genotype length	50
Crossover probability	0.8
Mutation probability	1
Crossover	One-point
Mutation	Uniform with $p_{codon} = 0.05$
Selection	Best

Fitness evaluation The fitness evaluation phase works as follows. For each training fold, we feed the features of each video to the DT and record its predictions. Then, we compute the following metrics of interest:

1. Video-level MSE (to be minimized):

$$\frac{1}{N_{videos}} \sum_{i=1}^{N_{videos}} (y_i^v - \hat{y}_i^v)^2; \tag{7.2}$$

2. Exam-level MSE (to be minimized):

$$\frac{1}{N_{exams}} \sum_{j=1}^{N_{exams}} (y_j^e - \hat{y}_j^e)^2; \tag{7.3}$$

3. Prognostic-level agreement (to be maximized):

$$\frac{1}{N_{exams}} \sum_{j=1}^{N_{exams}} \mathbb{I}(y_j^p = \hat{y}_j^p)^2; \tag{7.4}$$

where: y_i^v is the ground truth for the video i; $y_j^e = \sum_{i=1}^{14} y_{j,i}^v$ is the ground truth for exam j, i.e., the sum of the scores of each video of the exam; $y_j^p = \mathbb{I}(y_j^e > 24)$ is the ground truth for the prognosis j; $\mathbb{I}$ is the indicator function, i.e., it outputs 1 if the argument is true, otherwise 0. The notation $\hat{y}_b^a$ refers to the output of the DT given the output b in setting a, i.e., it is the approximation made by the DT of the variable y_b^a.

The pseudo-code for the fitness evaluation function (in the most general case, i.e., multi-objective) is shown in Algorithm 7. In the pseudo-code, a lowercase bold variable represents a vector, while an uppercase bold variable represents a matrix. Otherwise, the variable is assumed to be scalar.

The reason underlying the optimization of different metrics is the following. Our overall goal is to maximize the agreement for all the three metrics, as done in [82]. However, as we discussed earlier optimizing the video- and exam-level agreement requires also a specification of the tolerances to use for the computation of the agreement (e.g., in [82], the authors compute the exam-level agreement with a tolerance of 2, 5, and 10 points). Instead, optimizing the MSE for these two metrics (Lines 6-7 of Algorithm 7) allows us to evolve DTs that minimize the distance of the predictions from the ground truth, no matter the threshold. Finally, for the prognostic-level agreement, we cannot use the MSE because this variable is not a score but, instead, it is a binary variable. So, in this case, using the MSE does not give any advantage over directly optimizing the agreement (Line 8).

Then, for each metric, we use as fitness the worst value obtained on the 4 folds used for training (Lines 11-13).

While the single-objective fitness corresponds to a scalar value that consists in the value of a single metric, in the multi-objective setting it is composed of a tuple containing three values, i.e., the video-level MSE, the exam-level MSE, and the prognostic-level agreement.

7.2.3 Results

We perform 10 independent runs for the proposed method in each of the four settings: single-objective, video-level MSE; single-objective, exam-level MSE; single-objective, prognostic-level agreement; multi-objective. For each run, we test (on the test fold) only the best evolved DT, i.e., for the single-objective runs it is the individual with the best fitness while, for the multi-objective runs, it is the one with the maximum prognostic-level agreement (i.e., the most important metric).

Tables 7.5, 7.6, 7.7 show the descriptive statistics computed on the agreement (%) (with the physicians' opinion) computed across all the 5 folds. Bold values represent the best score across all the methods. We compute the statistics on all the 5 folds for the following reason. Since this work is meant to work in a medical scenario, we are not really interested in knowing the training and the test agreements as such. Instead, we are interested in knowing the worst-case scenario, and, to compute it, we need to compute the statistics on all the 5 folds. Note that, for each fold, we compute the three agreement scores on that fold and then we use these scores to compute the statistics across folds. The tables, "Video", "Exam", "Prognostic" and "3 objectives" refer to, respectively, our method evolved on video, exam, prognostic, and three objectives. In the same tables, we

Algorithm 7: Fitness evaluation function (multi-objective case)

```
   Input: $T$: the DT to evaluate
   Input: $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4$: input features of each fold
   Input: $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4$: video-level ground truth
   Input: $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4$: exam-level ground truth
   Input: $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$: prognostic-level ground truth
   Result: $f$: a list of fitnesses
 1 $num\_folds \leftarrow 4$ // Number of folds used for the training
 2 $e_v \leftarrow []$;
 3 $e_e \leftarrow []$;
 4 $a_p \leftarrow []$;
   // Iterate over the folds
 5 for ($i = 0$; $i < num\_folds$; $i{+}{+}$) do
       // For each fold, compute the metrics and concatenate
 6     $e_v \leftarrow concatenate(e_v, [video\_mse(T, \mathbf{X_i}, \mathbf{v_i})])$ // Eq. 7.2
 7     $e_e \leftarrow concatenate(e_e, [exam\_mse(T, \mathbf{X_i}, \mathbf{e_i})])$ // Eq. 7.3
 8     $a_p \leftarrow concatenate(a_p, [prognostic\_agreement(T, \mathbf{X_i}, \mathbf{p_i})])$ // Eq. 7.4
 9 end
   // Assign the worst-case to each metric
10 $f \leftarrow []$;
11 $f[0] \leftarrow max(e_v)$;
12 $f[1] \leftarrow max(e_e)$;
13 $f[2] \leftarrow min(a_p)$;
14 return $f$;
```

also report the state-of-the-art results presented in [82], listed as “JASA L” and “JASA L+S”, which refer to, respectively, the approach using only the labeling DNN, and the one using both the labeling and the segmentation DNNs. Note that we do not use the results shown in [82] but rather we evaluate them on the same folds used for the DTs, to guarantee a uniform evaluation of all the methods.

Finally, it should be considered that even if we use MSE as the metric for the first two objectives, we are still interested in evaluating the agreement with the physicians. For this reason, we do not show the MSE in the tables, but the agreement at the video and exam levels. This allows us also to use the same thresholds as in [82], keeping consistency on the method used for evaluating such models.

We observe that the model evolved on three objectives in some cases has a smaller minimum/mean agreement than that of the model specifically evolved on each metric.

However, it is never smaller than the minimum/mean agreement of the other two models evolved on the other metrics. This suggests that the model evolved by means of multi-objective optimization has a good trade-off between the three objectives.

Surprisingly, we observe that in some cases the performance of the DTs evolved in the multi-objective setting (i.e., the ones evolved on the three objectives simultaneously) exceeds even the performance of the best DTs found in the single-objective setting. This suggests that optimizing for all the metrics simultaneously can enforce a "consistency" between the different metrics and allows the DT to learn better strategies for classifying the samples. In fact, we find that the DTs evolved on single objectives do not generalize well to the other objectives. Instead, the DT evolved in the multi-objective setting is able to keep a good trade-off between the objectives.

Finally, compared with the two methods described in [82], we observe that while the DTs evolved in the multi-objective setting have a comparable (usually better) minimum agreement than that of JASA L and JASA L+S, they perform substantially better when considering the mean agreement computed on the folds.

Table 7.5: Descriptive statistics of the video-level agreement on all the folds. "Th" stands for the video-level threshold (i.e., the tolerance) used for the evaluation of the results.

Method	**Th**	**Min**	**Mean**	**Std**	**Med**	**Max**
JASA L	0	44.70	**50.40**	4.28	50.68	**57.38**
	1	82.74	85.87	2.49	85.91	89.76
JASA L+S	0	42.35	50.10	5.48	**51.43**	56.67
	1	82.74	85.87	2.43	85.36	89.29
Video	0	42.75	46.08	2.41	46.07	49.88
	1	**88.63**	**92.40**	**2.14**	**93.41**	**94.52**
Exam	0	42.35	47.82	4.74	46.14	54.29
	1	80.78	84.42	3.00	85.91	87.86
Prognostic	0	41.18	48.21	4.46	50.68	52.62
	1	79.61	83.71	2.79	85.48	86.43
3 objectives	0	**45.88**	49.48	**2.24**	49.77	52.86
	1	85.47	88.31	2.29	87.84	92.14

7.2.4 Analysis of the decision trees

In this section, we show the best-evolved DTs in each setting and interpret them to understand the relationships they captured on the predictions made by the DNNs. We consider as "best DT" the trees that satisfy the following properties (over the best solutions

Table 7.6: Descriptive statistics of the exam-level agreement on all the folds. "Th" stands for the exam-level threshold (i.e., the tolerance) used for the evaluation of the results.

Method	**Th**	**Min**	**Mean**	**Std**	**Med**	**Max**
JASA L	2	21.05	32.30	6.75	32.26	41.94
	5	45.00	56.95	9.43	57.90	70.97
	10	80.00	89.44	6.44	89.47	**100.00**
JASA L+S	2	21.05	30.97	7.58	35.00	38.71
	5	45.16	59.44	13.23	52.63	**80.64**
	10	80.00	84.58	6.27	81.25	96.77
Video	2	21.05	29.65	8.41	25.81	45.16
	5	40.00	57.75	12.14	**63.16**	70.97
	10	80.00	85.85	**4.47**	84.38	93.55
Exam	2	25.00	31.90	7.89	29.03	**46.88**
	5	**61.29**	**64.27**	**3.28**	**63.16**	70.00
	10	73.68	88.01	7.62	**90.32**	95.00
Prognostic	2	**26.32**	30.38	**3.77**	29.03	37.50
	5	45.00	58.50	8.81	59.38	67.74
	10	78.95	87.06	4.96	87.50	93.55
3 objectives	2	21.05	**36.36**	9.68	**38.71**	**46.88**
	5	45.00	63.98	11.35	62.50	77.42
	10	**85.00**	**91.51**	5.34	**90.32**	**100.00**

Table 7.7: Descriptive statistics of the prognostic-level agreement on all the folds. "Th" stands for the (prognostic) threshold used for the evaluation of the results.

Method	**Th**	**Min**	**Mean**	**Std**	**Med**	**Max**
JASA L	24	63.13	78.12	7.96	80.64	85.00
JASA L+S	24	57.90	76.78	12.01	83.87	90.00
Video	24	57.89	79.63	11.68	**85.00**	90.32
Exam	24	61.29	76.25	12.76	77.42	**95.00**
Prognostic	24	**73.68**	81.89	**5.27**	81.25	90.00
3 objectives	24	68.42	**82.11**	7.51	84.38	90.00

obtained in the 10 runs). For the setup considering only the video-level MSE, the best tree is the one that obtains the smallest MSE. When we only consider exam-level MSE, the best tree is the tree that achieves the smallest exam-level MSE. When we consider only the prognostic-level agreement, the best tree is the one with the highest prognostic-level agreement. Finally, when we consider all the objectives simultaneously, the best tree is the one that achieves the best prognostic-level agreement. In this case, if there are ties between two solutions, we choose the one that has the best trade-off between video- and exam-level MSE. In all the cases, the conditions of the DT are numbered as when doing a pre-order traversal of the DT.

Decision tree evolved on the video-level MSE Figure 7.8 shows the best DT obtained in this setting. While this DT performs worse than JASA L and JASA L+S when no tolerance is given to the prediction, it outperforms them significantly when a threshold of 1 is allowed. For this reason, we will interpret it considering that each prediction $\hat{y}$ must be considered as a value ranging in $[\hat{y} - 1, \hat{y} + 1]$ (constraining the values in $[0, 3]$).

This DT checks very few things about the predictions made by the two DNNs. In fact, the first split ($l_{max} > s_{max}$) captures a very simple pattern: when the maximum class of risk predicted by the labeling DNN is greater than the maximum risk class predicted by the segmentation DNN, then it assigns the video a risk varying in $[0, 2]$, i.e., it excludes the class 3. On the other hand, when the root condition is false, it checks whether the fraction of frames classified as maximum risk (by the labeling DNN) is bigger than the fraction of predictions made by the segmentation DNN in which the highest score is 2 ($s_2 < l_3$). If so, it assigns the video a score varying in $[2, 3]$, i.e., high risk. Basically, this condition checks whether the video refers to a high-risk patient. In fact, the condition can be interpreted as:

If the ratio of samples classified as maximum risk by the labeling DNN is bigger than the ratio of samples classified as risk 2 by the segmentation DNN, then give the priority to the labeling DNN and assign the maximum score to the video.

To confirm this hypothesis, in Figure 7.9c we plot the histogram of the number of videos assigned to each class that fall in the case explained above (note that in the other sub-figures of Figure 7.9 we do the same for all the other conditions in the DT). We observe that the number of videos belonging to class 3 is significantly higher w.r.t. the other classes. Finally, in the third condition ($l_2 < 0.15$) the DT makes an extremely simple check:

If the ratio of frame labeled with class 2 is low (i.e., under a threshold of 0.15), then probably the number of frames assigned to class 3 will be even lower, so assign a score ranging in $[0, 2]$ to the video. Otherwise, there is a high chance that the severity score is higher than 0, so assign a score ranging in $[1, 3]$ to the video.

From this DT, we infer that the labeling DNN may be "biased" toward high scores. The DT is then evolved to make use of the output of the segmentation DNN in order to reduce this bias.

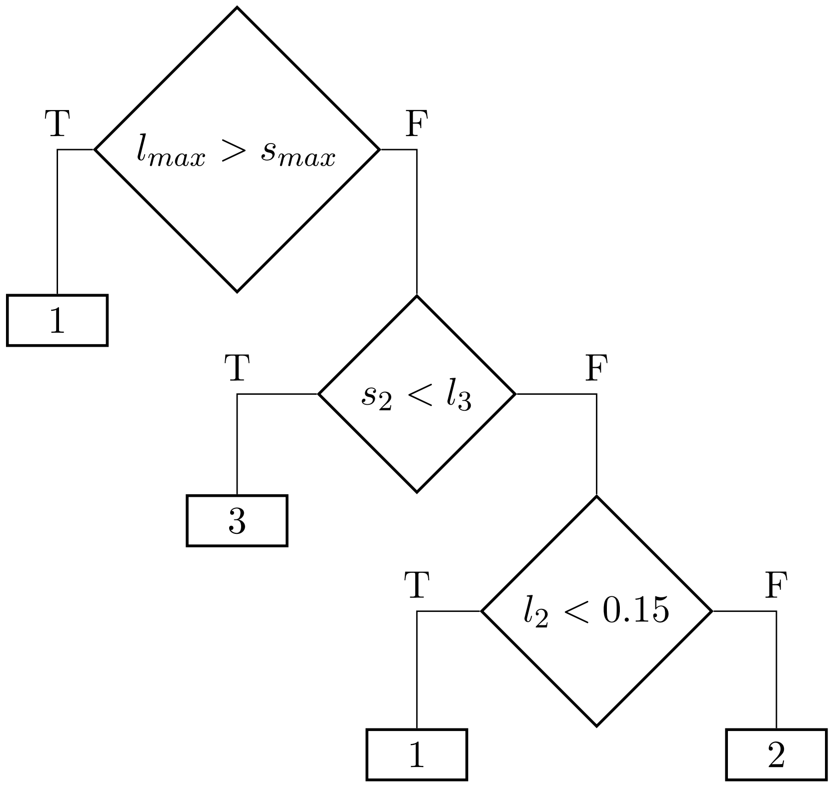
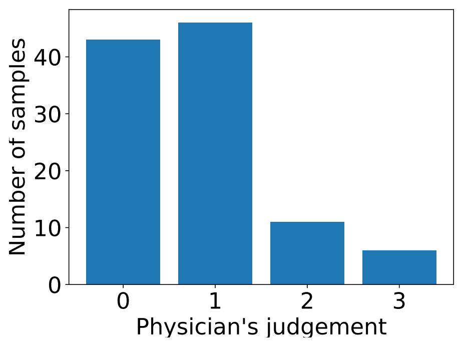


Figure 7.8: Best DT evolved on the video-level MSE.

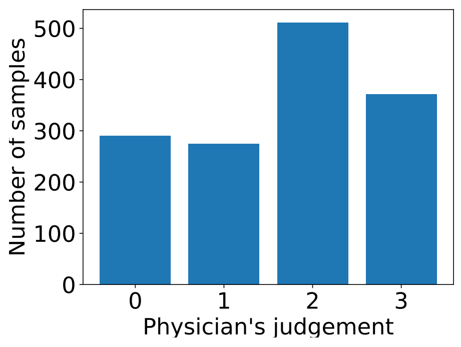
Decision tree evolved on the exam-level MSE This DT (shown in Figure 7.10) achieves a better worst-case agreement with low thresholds (2 and 5). However, in this case (and the following ones), we cannot use the threshold as a tolerance value to be used on the output value of the DT. This is due to the fact that, in this case, the DT outputs the gravity for each video, but the tolerance is expressed at the exam level.

The first condition of this DT ($l_0 < 0.72$) checks that the severity of the patient is high, by ensuring that the fraction of frames labeled as minimum risk is lower than an evolved threshold. If so, it then assesses the severity of the conditions by using the segmentation DNN, checking if the fraction of frames that are classified as maximum risk is more than the half ($s_3 > 0.51$). If so, it assigns the maximum risk to the video.

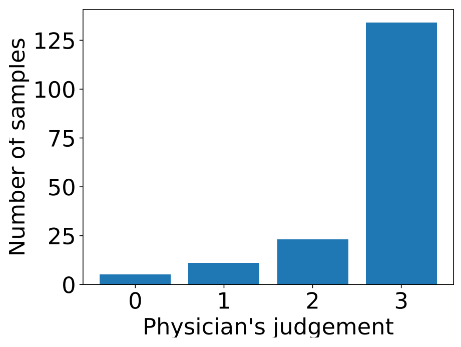
If the first condition is false, then the DT performs additional checks. In fact, the right part of the DT is basically a decision list, i.e., an extremely unbalanced DT, which isolates one particular case at each split. While the first condition on the right ($s_2 > l_{max}$)



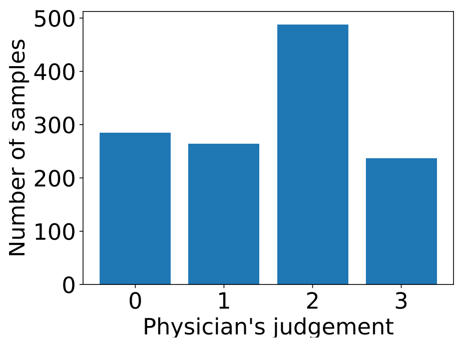
(a) Left branch, 1st condition.



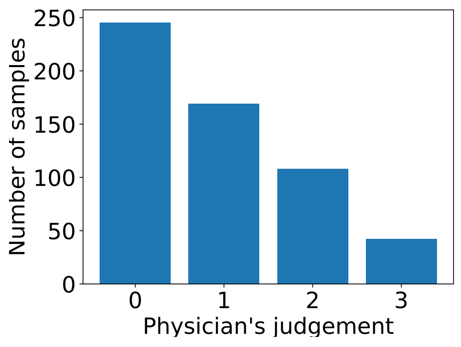
(b) Right branch, 1st condition.



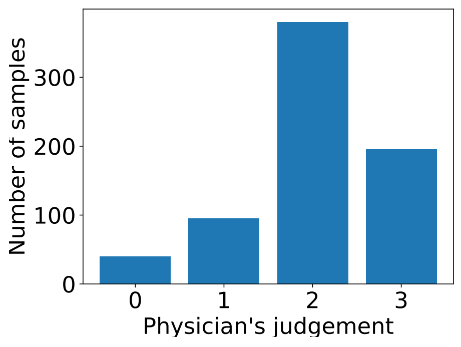
(c) Left branch, 2nd condition.



(d) Right branch, 2nd condition.



(e) Left branch, 3rd condition.



(f) Right branch, 3rd condition.

Figure 7.9: Class histograms for each of the branches of the DT shown in Figure 7.8. Note that the nodes are counted as in a pre-order traversal of the DT.

may make no sense at a first sight, it is a simple trick that the DT uses to perform an *and* between two conditions. In fact, we know that $s_2 \in [0, 1]$ and $l_{max} \in \{0, 1, 2, 3\}$. This means that the condition $s_2 > l_{max}$ evaluates to true only in case $s_2 > 0$ and $l_{max} = 0$.

The second case isolated by the decision list checks for a particular case ($s_1 = l_3$). By analyzing the training set, we observe that this case only happens when $s_1 = l_3 = 0$. Moreover, in these cases, s_1 and l_3 are the only variables that are always equal to zero.

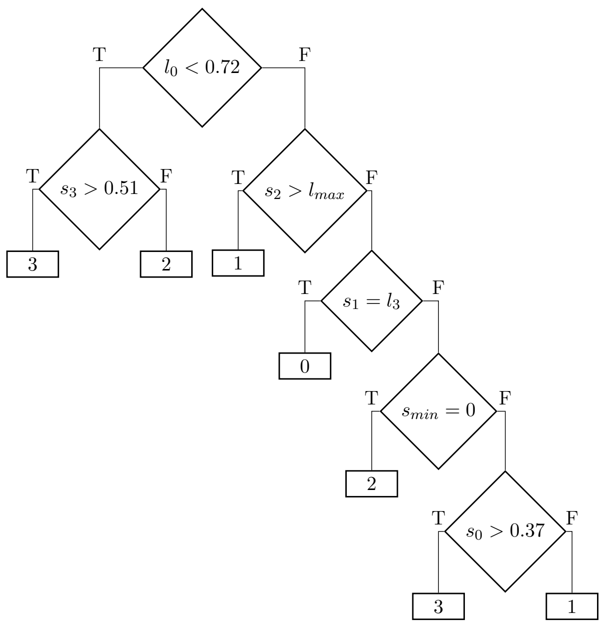


Figure 7.10: Best DT evolved on the exam-level MSE.

While this may seem a remote possibility, we found that this condition (conjoined with the conditions that are evaluated before it) evaluates to true for about a quarter of the samples in the training set. Hence, this condition exploits a bias of the two DNNs to detect cases in which the severity is likely to be low (45% of the cases with score 0, 31% of the cases with score 1, 17% with score 2, 7% with score 3).

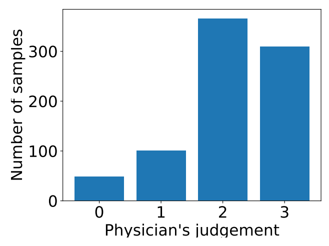
The third condition on the right branch ($s_{min} = 0$) checks whether the patient has at least one frame with minimum risk (i.e., 0, as opposed to a case in which no damaged tissue is detected, i.e., -1) by checking the outputs of the segmentation DNN. If so, it assigns class 2 to the video.

Finally, the last condition ($s_0 > 0.37$) checks the number of frames with minimum risk (detected by the segmentation model): if they consist of more than the 37% of the frames in the video, the risk assigned to the video is very high (3). Otherwise, it is assigned a lower score (1).

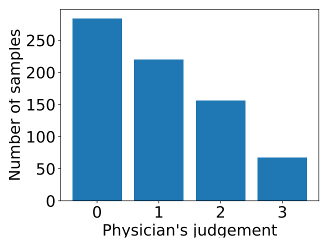
It is important to note that the video-level predictions performed by this model aim to reduce the worst-case exam-level MSE w.r.t. the physicians' judgment.

Also in this case, for the sake of completeness, we report in Figure 7.11 the distribution of classes for each condition in the DT.

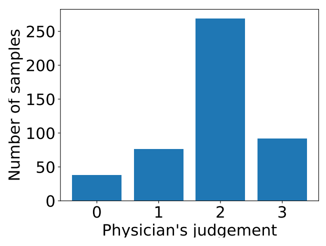
Decision tree evolved on the prognostic-level agreement This DT (shown in Figure 7.12) achieves the best worst-case prognostic-level agreement among all the best DTs (in this case, evolved with the single threshold value, 24).



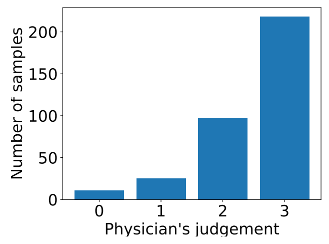
(a) Left branch, 1st condition.



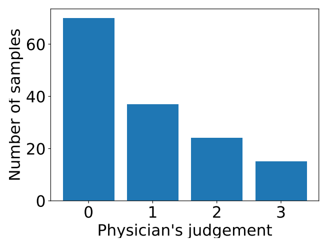
(b) Right branch, 1st condition.



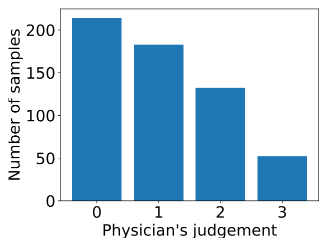
(c) Left branch, 2nd condition.



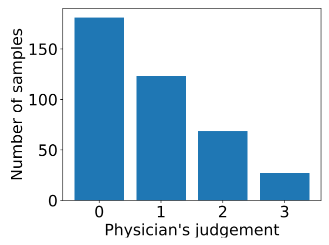
(d) Right branch, 2nd condition.



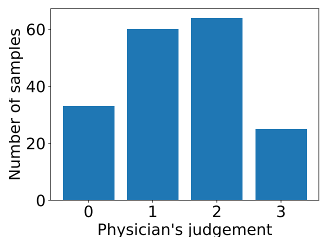
(e) Left branch, 3rd condition.



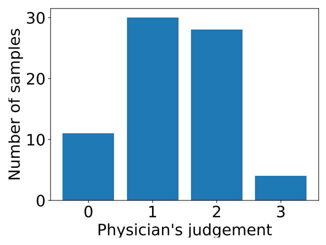
(f) Right branch, 3rd condition.



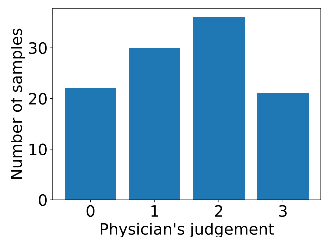
(g) Left branch, 4th condition.



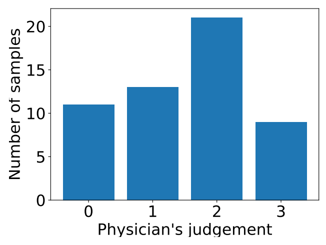
(h) Right branch, 4th condition.



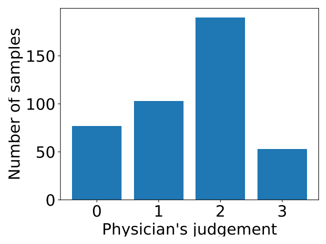
(i) Left branch, 5th condition.



(j) Right branch, 5th condition.



(k) Left branch, 6th condition.



(l) Right branch, sixth condition.

Figure 7.11: Class histograms for each of the branches of the DT shown in Figure 7.10. Note that the nodes are counted as in a pre-order traversal of the DT.

In the root condition ($l_3 < s_1$), this DT checks whether the confidence given to class 3 from the labeling DNN is smaller than the confidence given to class 1 by the segmentation DNN. This, intuitively, tries to filter out the cases where the probability of having the maximum risk is high. In fact, as shown in Figure 7.14a, the ratio of samples belonging to class 3 is not so high in this case (12.5%), see the other sub-figures in Figure 7.14 for the distributions of classes corresponding to the other conditions in the DT.

The second condition (i.e., the left branch of the root, $l_0 > 0.75$) naturally follows the first one: given that, as shown in Figure 7.14a, the distribution of the classes is skewed towards class 2, is there a way to filter out the samples belonging to class 2? While this condition does not filter perfectly the samples belonging to class 2, it is able to filter 67.9% of them (as shown in Figures 7.14c and 7.14d).

The third condition ($l_{max} = l_{min}$) seeks cases in which the maximum class and the minimum class predicted by the labeling DNN are equal. Of course, this condition is way more likely to happen in low-risk frames, as confirmed by Figure 7.14e. However, as we can see from Figure 7.14f, not all the samples with low risk are filtered out by this condition.

For this reason, the purpose of the fourth condition ($s_3 < s_0$) is to separate the low-risk cases from the higher-risk ones. In fact, what it does is simply check the predictions made by the segmentation DNN: if the ratio of samples assigned to class 0 is higher than the ratio of samples assigned to class 3, then it predicts 0, otherwise 2.

Note that this DT has been optimized to maximize the prognostic-level agreement. This explains why, in some cases, the outputs are not coherent with what a human expects when trying to predict the label for each *video*. In fact, we hypothesize that these counter-intuitive tests aim to soften the contributions of each video to the prognostic score. This can be seen in the fact that predictions for class 3 never appear in this DT, and also that the predictions for class 1 are not frequent (even when they would minimize the video-level MSE, which is not taken into account when optimizing this DT).

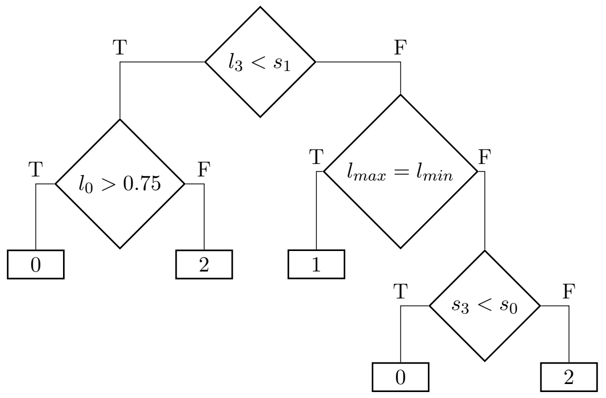


Figure 7.12: Best DT evolved on the prognostic-level agreement.

Decision tree evolved on three objectives This DT (shown in Figure 7.13) has comparable, but often better, performance with respect to all the other DTs evolved in the

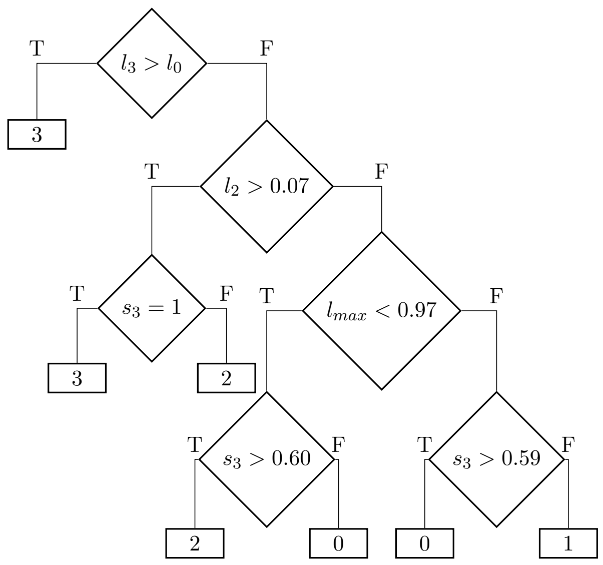


Figure 7.13: Best DT evolved on the three objectives.

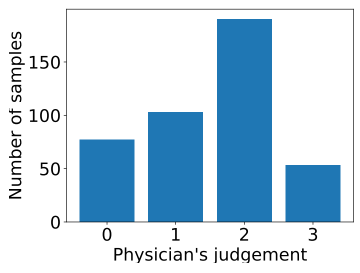
other settings. One interesting feature of this DT is that it uses the labeling DNN to make coarse-grained decisions, that are then refined by using the segmentation DNN.

In the first condition ($l_3 > l_0$), this DT simply checks the outputs coming from the labeling DNN to address the gravity of the conditions. Surprisingly, only checking if the ratio of samples assigned to risk 3 is higher than the ratio of samples assigned to risk 0 is enough to discriminate very well the high-risk cases, as shown in Figure 7.15a (see the other sub-figures in Figure 7.15 for the distributions of classes corresponding to the other conditions in the DT).

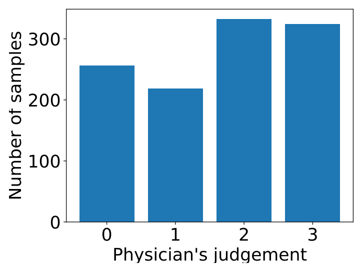
In the second condition ($l_2 > 0.07$), the DT checks the ratio of labels assigned to class 2 by the labeling DNN. If they are more than 7%, then it makes a simple refinement using the segmentation DNN: if the segmentation DNN classifies all the samples as class 3 ($s_3 = 1$), then it assigns the video the maximum score, otherwise, it assigns the video a score of 2.

If the second condition evaluates to false, then the DT checks whether $l_{max} < 0.97$. Since l_{max} is an integer, this corresponds to checking whether $l_{max} = 0$. If so, again, the DT makes use of the segmentation DNN to refine the decision: if the ratio of samples assigned to class 3 by the segmentation DNN is more than the 60% ($s_3 > 0.60$), then it assigns the video a score of 2. Otherwise, the gravity of the condition is not high enough, so it assigns a score of 0 to the video. This condition handles a bias of the labeling DNN, which happens when this DNN classifies all the frames with a severity of 0, but, instead, their actual score is very different from 0.

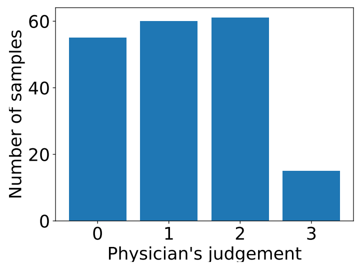
Finally, if $l_{max} > 0.97$, it uses a similar check ($s_3 > 0.59$) to assign the samples either



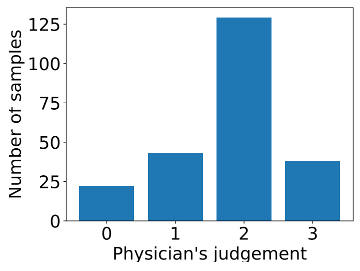
(a) Left branch, 1st condition.



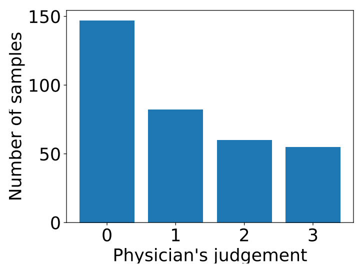
(b) Right branch, 1st condition.



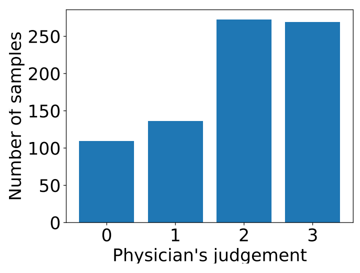
(c) Left branch, 2nd condition.



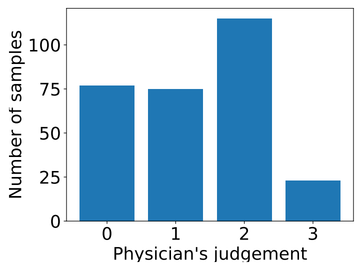
(d) Right branch, 2nd condition.



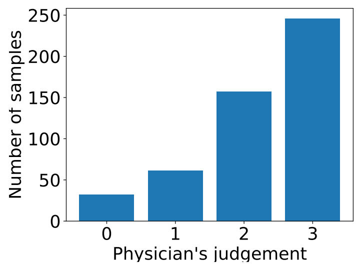
(e) Left branch, 3rd condition.



(f) Right branch, 3rd condition.



(g) Left branch, 4th condition.



(h) Right branch, 4th condition.

Figure 7.14: Class histograms for each of the branches of the DT shown in Figure 7.12. Note that the nodes are counted as in a pre-order traversal of the DT.

to class 0 or 1. Surprisingly, when s_3 is greater than 59%, the DT assigns the sample to class 0 while, as we can see from Figure 7.15k, assigning it a value equal to 1 would reduce the video-level MSE. However, this reasoning applies to the video-level predictions, but it may affect negatively the other two metrics. On the other hand, when $s_3 \leq 0.59$, we observe that the probability for class 3 is quite low, so the DT classifies the sample as belonging to class 1, probably to minimize the video-level MSE.

7.2.5 Conclusions

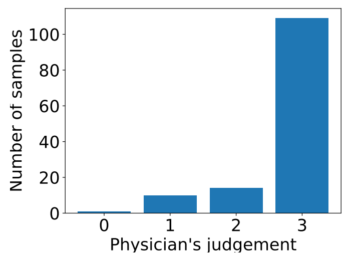
In this section, we combined two previously proposed DNNs as *feature extractors*, and then we used a DT for combining the two predictions. We use both single- and multi-objective evolutionary optimization to evolve the DT that takes in input the predictions made by the two DNNs aggregated at the video level (i.e., a collection of frames). When evaluating our approach on three different levels of agreement with the physicians' judgment, we find that the multi-objective optimization approach leads to DTs that, in general, perform in most cases comparably or better than the DTs evolved on single objectives. Moreover, our approach appears to perform better (in terms of descriptive statistics) than the approach presented in [82].

In light of these limitations, future work should aim at (1) collecting more data, in order to increase the size of the dataset, and evolving DTs by using different types of conditions, including oblique ones (which may lead to different insights); and (2) assessing the interpretability of the DTs produced with physicians.

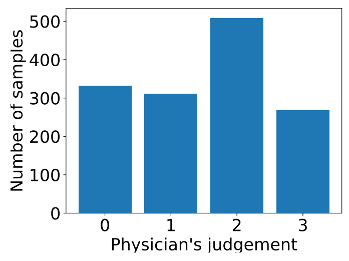
Finally, we highlight that we make our data publicly available for further development and reproducibility[4]. Moreover, in a separate repository[5] we release the scripts used to produce the results shown in this section.

[4]https://drive.google.com/drive/folders/1Or4dF2fAM23H5fd_yxtq1vyAS8b7pL0s

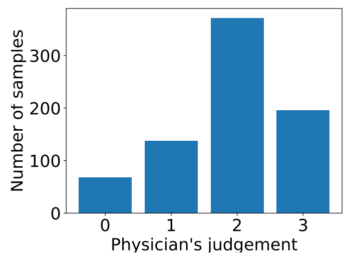
[5]https://gitlab.com/leocus/neurosymbolic-covid19-scoring



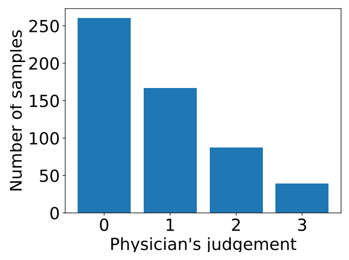
(a) Left branch, 1st condition.



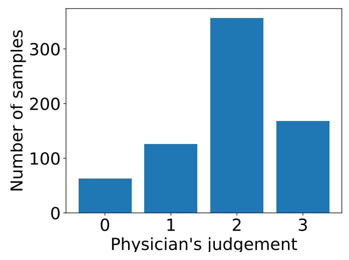
(b) Right branch, 1st condition.



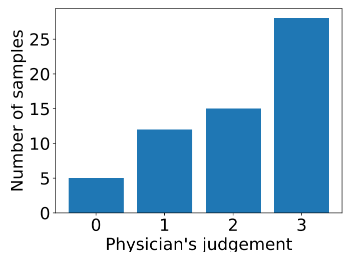
(c) Left branch, 2nd condition.



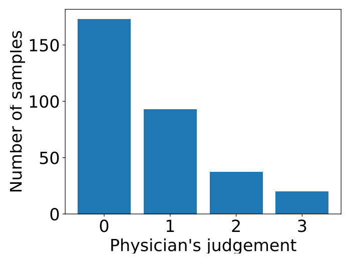
(d) Right branch, 2nd condition.



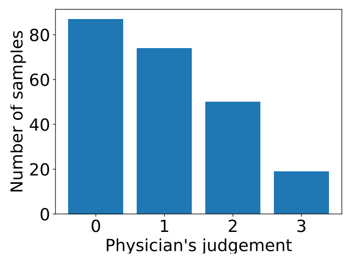
(e) Left branch, 3rd condition.



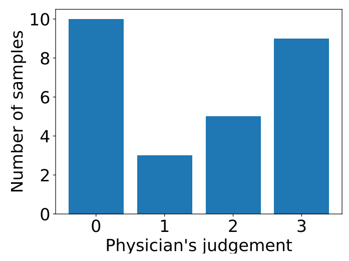
(f) Right branch, 3rd condition.



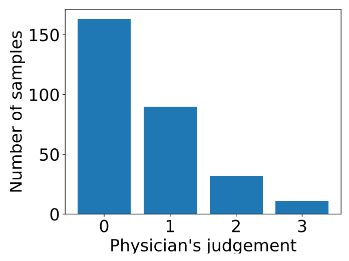
(g) Left branch, 4th condition.



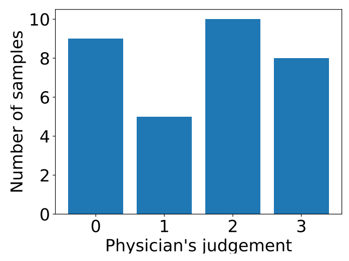
(h) Right branch, 4th condition.



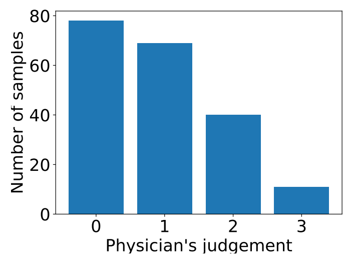
(i) Left branch, 5th condition.



(j) Right branch, 5th condition.



(k) Left branch, 6th condition.



(l) Right branch, 6th condition.

Figure 7.15: Class histograms for each of the branches of the DT shown in Figure 7.13. Note that the nodes are counted as in a pre-order traversal of the DT.

Chapter 8

Conclusions and Future Directions

While in recent years AI made 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 to have a comprehensive understanding of what happens in the model. In fact, understanding how models take their decisions is extremely important for their real-world applications, as pointed out by the European Union and UNESCO. However, these models are not widely used in practice because of their widely-thought lower performance. Moreover, little work has been done at the intersection of Interpretable AI and Reinforcement Learning.

In this thesis, we took steps in this direction, proposing methods for the application of Decision Trees in Reinforcement Learning contexts focused on interpretability. The experimental results presented in this thesis indicate that these approaches can be competitive w.r.t. black-box methods (in score) while being extremely easier to interpret. These results suggest that the widely thought performance-interpretability trade-off does not always hold (as suggested by [116]) 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 presented also the results obtained in scenarios that are much closer to the real world. In the pandemic control task, interpretability was a *hard* constraint. In fact, to be applied in the real world, the policy needs to be translated into law. Also in these cases, our methods have shown to be competitive with the state of the art, confirming that it is indeed a promising direction. On the other hand, in the automatic assessment of COVID-19, our methodology, besides improving performance, was helpful to understand the biases of the two neural networks that were being used to extract features.

In all the sections, we practically showed how easily these models can be interpreted.

This is very important, as interpretability does not have to be confused with transparency [6]. In fact, as stated in [66], very deep DTs may not be interpretable. We also found that this is in agreement with the metrics of interpretability used in this thesis [149, 5].

While these methods seem promising, they need extensive experimentation on a variety of domains in order to thoroughly understand their points of strength and weaknesses. Moreover, the number of interactions needed with the environment needed by the methodologies proposed in this thesis is often significantly worse than those needed for non-interpretable, state-of-the-art approaches, e.g., DQN [85], Proximal Policy Optimization [128], and similar. This means that the training time for the methods discussed in this is significantly higher than that of black-box state-of-the-art methods. Finally, they include several components (e.g., GE/GP, $\mathcal{Q}$-Learning, co-evolutionary algorithm). This means that the set of hyperparameters is significantly bigger than most state-of-the-art RL methods.

Future work should address these points:

1. extending the experimentation presented in this thesis to a wide set of well-known RL benchmarks, such as the MuJoCo [144] or the full Atari ALE [9, 75] benchmarks;

2. increasing the efficiency of such methods, making them competitive with the non-interpretable state-of-the-art methods also from the point of view of the computational budget needed for the training process;

3. automating the hyperparameter optimization process of such methods in such a way that the practitioner can adapt the method to their own use case by modifying a reduced set of hyperparameters;

4. assisting the human operator in interpreting the solutions obtained. To this end, one may make use of Large Language Models that, taking in input a summary of the background of a user and the solution obtained, tries to explain as simply as possible the solution to the user.

Papers published during the Doctoral Programme

Selected for the thesis

Custode, Leonardo L., and Giovanni Iacca. 'Evolutionary Learning of Interpretable Decision Trees'. IEEE Access, IEEE, 2023: in this paper (which is presented in Chapter 3) we propose the main methodology for evolving DTs by means of Grammatical Evolution and Q-Learning.

Custode, Leonardo Lucio, and Giovanni Iacca. 'A Co-Evolutionary Approach to Interpretable Reinforcement Learning in Environments with Continuous Action Spaces'. 2021 IEEE Symposium Series on Computational Intelligence (SSCI), 2021: this paper, explained in Chapter 4, extends the methodology from Chapter 3 to deal with environments with continuous action spaces.

Custode, Leonardo Lucio, and Giovanni Iacca. 'Interpretable pipelines with evolutionary optimized modules for reinforcement learning tasks with visual inputs'. Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2022, pp. 224–227: in this paper, we present the methodology explained in Chapter 5 to evolve IRL agents for environments with images as states.

Crespi, Marco, et al. 'Towards Interpretable Policies in Multi-Agent Reinforcement Learning Tasks'. Bioinspired Optimization Methods and Their Applications: 10th International Conference, BIOMA 2022, Maribor, Slovenia, November 17–18, 2022, Proceedings, Springer International Publishing Cham, 2022, pp. 262–276: in this paper, presented in Chapter 6, we extend the methodology from Chapter 3 to multi-agent scenarios.

Custode, Leonardo Lucio, and Giovanni Iacca. 'Interpretable AI for Policy-Making in Pandemics'. Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2022, pp. 1763–1769: this paper, presented in Section 7.1, uses the methodology from Chapter 3 for containing pandemics.

Custode, Leonardo Lucio, Federico Mento, Francesco Tursi, et al. 'Multi-Objective Automatic Analysis of Lung Ultrasound Data from COVID-19 Patients by Means of Deep Learning and Decision Trees'. Applied Soft Computing, vol. 133, Elsevier, 2023, p. 109926: this paper proposes a multi-objective approach for the training of decision trees to aggregate the predictions of neural networks for the automatic assessment of the COVID-19 patients' conditions. This approach is described in Section 7.2.

Not selected for the thesis

Lorandi, Michela, et al. 'Genetic Improvement of Routing Protocols for Delay Tolerant Networks'. ACM Transactions on Evolutionary Learning and Optimization, vol. 1, no. 1, ACM New York, NY, 2021, pp. 1–37: this paper presents a methodology for optimizing network protocols in a data-driven manner using Genetic Programming.

Lotito, Quintino Francesco, et al. 'A Signal-Centric Perspective on the Evolution of Symbolic Communication'. Proceedings of the Genetic and Evolutionary Computation Conference, 2021, pp. 120–128: in this paper, we study the emergent communication between two neural networks that learn how to communicate.

Qiu, Hao, et al. 'Black-Box Adversarial Attacks Using Evolution Strategies'. Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2021, pp. 1827–1833: this paper studies the application of evolution strategies to the evolution of adversarial perturbations (i.e., human-imperceptible modifications of the input of a deep neural network that leads to a misclassification).

Mo, Hyunho, et al. 'Evolutionary Neural Architecture Search for Remaining Useful Life Prediction'. Applied Soft Computing, vol. 108, Elsevier, 2021, p. 107474: in this paper, we show how evolutionary algorithms may be helpful in evolving neural network architectures for predictive maintenance tasks.

Custode, Leonardo Lucio, and Giovanni Iacca. 'One Run to Attack Them All: Finding Simultaneously Multiple Targeted Adversarial Perturbations'. 2021 IEEE

Symposium Series on Computational Intelligence (SSCI), 2021 shows how we can use bio-inspired algorithms to find adversarial attacks that are universal (i.e., they do not depend on the image to attack) and targeted (i.e., they allow us to choose the predicted class).

Custode, Leonardo Lucio, Hyunho Mo, Andrea Ferigo, et al. 'Evolutionary Optimization of Spiking Neural P Systems for Remaining Useful Life Prediction'. Algorithms, vol. 15, no. 3, MDPI, 2022, p. 98: in this paper, we employ Spiking Neural P Systems to predict the remaining useful life of aircraft components.

Custode, Leonardo Lucio, Hyunho Mo, and Giovanni Iacca. 'Neuroevolution of Spiking Neural p Systems'. Applications of Evolutionary Computation: 25th European Conference, EvoApplications 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20–22, 2022, Proceedings, Springer International Publishing Cham, 2022, pp. 435–451: in this paper we use the NEAT neuroevolutionary algorithm to evolve Spiking Neural P (SN P) systems.

Custode, Leonardo Lucio, Federico Mento, Sajjad Afrakhteh, et al. 'Neuro-Symbolic Interpretable AI for Automatic COVID-19 Patient-Stratification Based on Standardised Lung Ultrasound Data'. Proceedings of Meetings on Acoustics 182ASA, vol. 46, Acoustical Society of America, 2022, p. 020002 uses decision trees, evolved by means of Grammatical Evolution, for aggregating predictions of two neural networks for the automatic assessment of the severity of COVID-19 patients' lungs.

Carbognin, Alberto, et al. 'Genetic Improvement of TCP Congestion Avoidance'. Bioinspired Optimization Methods and Their Applications: 10th International Conference, BIOMA 2022, Maribor, Slovenia, November 17–18, 2022, Proceedings, Springer International Publishing Cham, 2022, pp. 114–126: in this paper, we employ Genetic Programming to evolve specialized variants of the TCP network protocol.

Ferigo, Andrea, et al. 'Quality Diversity Evolutionary Learning of Decision Trees'. (accepted at the ACM Symposium of Applied Computing 2023 conference) shows how quality-diversity approaches can help in evolving decision trees for reinforcement learning tasks. Available on ArXiv at the following link: https://arxiv.org/abs/2208.12758

Acknowledgements

First, I want to thank my advisor, Giovanni Iacca, for everything he taught me over these years, for being a fantastic advisor, and for supporting and caring about me even when no one else would have done it. I could not have asked for a better advisor. Thank you very much, Giovanni.

I want to thank my girlfriend, Rosa, for being incredibly supportive and patient throughout these years. Thank you for being so kind and helping me be the best version of myself. Thank you for being at my side in the bad moments I've been through. I love you.

I also want to thank my family (in Italian). Voglio ringraziare mio padre, che è stato il primo ad accorgersi di questa mia inclinazione, e che mi ha incoraggiato a perseguire i miei sogni. I tuoi consigli sono stati molto preziosi e mi hanno portato a fare ciò che amo. Grazie a mia madre, per tutti i sacrifici che ha fatto e che sta facendo, per tutto l'amore e il supporto che mi dimostra tutti i giorni. Un grazie va anche a mia sorella, Lucia, per esserci sempre per me e per rendere la mia vita più luminosa. Voglio anche ringraziare mia nonna, Lucia, per tutto quello che ha fatto per me, da sempre. Infine, voglio dedicare un pensiero a tutte le persone che sono state importanti per me e che non sono più tra noi: i miei nonni Antonietta, Carmelo e Leonardo, e mio zio Franco.

Moreover, I want to thank my friends, who always show me their love and support. Thanks to Giovanni, Antonio, Ciro, Francesco, Alessandro, Luigi, Serena, Filomena, Antonio, Isabella, Enrica, Miriam, Michela, Sara, Anna, Mariagiulia, and all the others that I met before and during this path. Thanks for all the special moments we spent together. I want you to know that you will always have a place in my heart.

Finally, I want to thank the colleagues of my research group: Hyunho, Andrea, Elia, and all the others with whom I shared a part of this path. Thank you for all the valuable conversations we had.

Bibliography

[1] Marco Allinovi, Alberto Parise, Martina Giacalone, Andrea Amerio, Marco Delsante, Anna Odone, Andrea Franci, Fabrizio Gigliotti, Silvia Amadasi, Davide Delmonte, Niccolò Parri, and Angelo Mangia. Lung Ultrasound May Support Diagnosis and Monitoring of COVID-19 Pneumonia. *Ultrasound in Medicine and Biology*, 46(11):2908–2917, sep 2020.

[2] Jose M. Alonso, Ciro Castiello, and Corrado Mencar. *Interpretability of Fuzzy Systems: Current Research Trends and Prospects*, pages 219–237. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[3] Dirk V Arnold and Hans-Georg Beyer. *Noisy optimization with evolution strategies*, volume 8. Springer, Boston, MA, USA, 2002.

[4] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the Atari Human Benchmark, 2020. arXiv:2003.13350.

[5] Pablo Barceló, Mikaël Monet, Jorge Pérez, and Bernardo Subercaseaux. Model Interpretability through the Lens of Computational Complexity. *arXiv:2010.12265 [cs]*, 33, November 2020. arXiv: 2010.12265.

[6] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, June 2020.

[7] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpretability via Model Extraction.

[8] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpreting Blackbox Models via Model Extraction. *arXiv:1705.08504 [cs]*, January 2019. arXiv: 1705.08504.

[9] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[10] Roman Beltiukov. Optimizing Q-Learning with K-FAC Algorithm. In *International Conference on Analysis of Images, Social Networks and Texts*, pages 3–8. Springer, Cham, 2020.

[11] Martijn C. J. Bot. Improving induction of linear classification trees with genetic programming. In *Genetic and Evolutionary Computation Conference*, pages 403–410, San Francisco, CA, USA, July 2000. Morgan Kaufmann Publishers Inc.

[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540*, 2016. arXiv:1606.01540.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016. arXiv:1606.01540.

[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[15] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, March 2008.

[16] Erick Cantú-Paz and Chandrika Kamath. Inducing Oblique Decision Trees With Evolutionary Algorithms. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 7(1):15, 2003.

[17] Zixuan Cao, Mengzhi Shi, Zhanbo Zhao, and Xiujun Ma. Pool: Pheromone-inspired communication framework forlarge scale multi-agent reinforcement learning, 2022.

[18] L Carrer, E Donini, D Marinelli, M Zanetti, Federico Mento, E Torri, A Smargiassi, R Inchingolo, G Soldati, L Demi, F Bovolo, and L Bruzzone. Automatic Pleural Line

Extraction and COVID-19 Scoring from Lung Ultrasound Data. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 67(11):2207–2217, 2020.

[19] Xiangxiang Chu and Hangjun Ye. Parameter Sharing Deep Deterministic Policy Gradient for Cooperative Multi-agent Reinforcement Learning. *arXiv:1710.00336*, October 2017. arXiv:1710.00336.

[20] William G Cochran. *Sampling techniques.* John Wiley & Sons, 2007.

[21] Leonardo Lucio Custode and Giovanni Iacca. Evolutionary learning of interpretable decision trees, 2020.

[22] Leonardo Lucio Custode and Giovanni Iacca. A co-evolutionary approach to interpretable reinforcement learning in environments with continuous action spaces. In *IEEE Symposium Series on Computational Intelligence*, pages 1–8, New York, NY, USA, 2021. IEEE.

[23] Leonardo Lucio Custode and Giovanni Iacca. Evolutionary learning of interpretable decision trees, April 2021. arXiv:2012.07723.

[24] Leonardo Lucio Custode and Giovanni Iacca. Evolutionary learning of interpretable decision trees, 2021.

[25] Marcin Czajkowski and Marek Kretowski. A multi-objective evolutionary approach to Pareto-optimal model trees. *Soft Computing*, 23(5):1423–1437, March 2019.

[26] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *International conference on parallel problem solving from nature*, pages 849–858. Springer, 2000.

[27] Libertario Demi. Lung ultrasound: The future ahead and the lessons learned from COVID-19. *The Journal of the Acoustical Society of America*, 148(4):2146–2150, oct 2020.

[28] Yashesh Dhebar, Kalyanmoy Deb, Subramanya Nageshrao, Ling Zhu, and Dimitar Filev. Interpretable-AI Policies using Evolutionary Nonlinear Decision Trees for Discrete Action Systems. *arXiv:2009.09521 [cs, eess, stat]*, September 2020. arXiv: 2009.09521.

[29] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words:

Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, 2021.

[30] Gary Duclos, Alexandre Lopez, Marc Leone, and Laurent Zieleskiewicz. “No dose” lung ultrasound correlation with “low dose” CT scan for early diagnosis of SARS-CoV-2 pneumonia. *Intensive Care Medicine*, 46(6):1103–1104, 2020.

[31] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, Feb 2021.

[32] IU Ekanayake, DPP Meddage, and Upaka Rathnayake. A novel approach to explain the black-box nature of machine learning in compressive strength predictions of concrete using Shapley additive explanations (SHAP). *Case Studies in Construction Materials*, 16:e01059, 2022.

[33] Guangzhe Fan and J. Brian Gray. Regression Tree Analysis Using TARGET. *Journal of Computational and Graphical Statistics*, 14(1):206–218, 2005.

[34] Jiajun Fan, Changnan Xiao, and Yue Huang. GDI: Rethinking What Makes Reinforcement Learning Different From Supervised Learning, 2022. arXiv:2106.06232.

[35] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[36] Gianluigi Folino, Clara Pizzuti, and Giandomenico Spezzano. A Cellular Genetic Programming Approach to Classification. In *Genetic and Evolutionary Computation Conference*, pages 1015–1020. Citeseer, Citeseer, 1999.

[37] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware Minimization for Efficiently Improving Generalization. In *International Conference on Learning Representations*, 2021.

[38] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration, 2017. arXiv:1706.10295.

[39] Eibe Frank, Yong Wang, Stuart Inglis, Geoffrey Holmes, and Ian H. Witten. Using Model Trees for Classification. *Machine Learning*, 32(1):63–76, July 1998.

[40] Oz Frank, Nir Schipper, Mordehay Vaturi, Gino Soldati, Andrea Smargiassi, Riccardo Inchingolo, Elena Torri, Tiziano Perrone, Federico Mento, Libertario Demi,

Meirav Galun, Yonina C Eldar, and Shai Bagon. Integrating Domain Knowledge into Deep Networks for Lung Ultrasound with Applications to COVID-19. *IEEE Transactions on Medical Imaging*, page 1, 2021.

[41] Taiki Fuji, Kiyoto Ito, Kohsei Matsumoto, and Kazuo Yano. Deep multi-agent reinforcement learning using dnn-weight evolution to optimize supply chain performance. In *51st Hawaii International Conference on System Sciences*, pages 1278–1287, Honolulu, HI, USA, 2018. HICSS.

[42] Abhiroop Ghosh, Yashesh Dhebar, Ritam Guha, Kalyanmoy Deb, Subramanya Nageshrao, Ling Zhu, Eric Tseng, and Dimitar Filev. Interpretable AI Agent Through Nonlinear Decision Trees for Lane Change Problem. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–08, 2021.

[43] Vinicius G. Goecks, Gregory M. Gremillion, Vernon J. Lawhern, John Valasek, and Nicholas R. Waytowich. Integrating Behavior Cloning and Reinforcement Learning for Improved Performance in Dense and Sparse Reward Environments. *arXiv:1910.04281 [cs, stat]*, April 2020. arXiv: 1910.04281.

[44] Ahmed Hallawa, Thorsten Born, Anke Schmeink, Guido Dartmann, Arne Peine, Lukas Martin, Giovanni Iacca, A. E. Eiben, and Gerd Ascheid. EVO-RL: Evolutionary-Driven Reinforcement Learning. *arXiv:2007.04725 [cs, stat]*, July 2020. arXiv: 2007.04725.

[45] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation*, pages 312–317, New York, NY, USA, 1996. IEEE, IEEE.

[46] Thomas Haynes, Roger L. Wainwright, Sandip Sen, and Dale A. Schoenefeld. Strongly Typed Genetic Programming in Evolving Cooperation Strategies. In *6th International Conference on Genetic Algorithms*, pages 271–278, San Francisco, CA, USA, July 1995. Morgan Kaufmann Publishers Inc.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[48] David Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *IJCAI*, volume 1993, pages 1002–1007. Citeseer, Citeseer, 1993.

[49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[50] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay, 2018. arXiv:1803.00933.

[51] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, May 1976.

[52] Cezary Z Janikow. Fuzzy decision trees: issues and methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(1):1–14, 1998.

[53] Ulf Johansson, Rikard König, and Lars Niklasson. The truth is in there-rule extraction from opaque models using genetic programming. In *FLAIRS Conference*, pages 658–663. Miami Beach, FL, 2004.

[54] Ulf Johansson, Lars Niklasson, and Rikard König. Accuracy vs. comprehensibility in data mining models. In *Proceedings of the seventh international conference on information fusion*, volume 1, pages 295–300. Citeseer, 2004.

[55] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR, 2017.

[56] Varun Kompella*, Roberto Capobianco*, Stacy Jong, Jonathan Browne, Spencer Fox, Lauren Meyers, Peter Wurman, and Peter Stone. Reinforcement Learning for Optimization of COVID-19 Mitigation policies, 2020.

[57] John R. Koza. Concept formation and decision tree induction using the genetic programming paradigm. In Hans-Paul Schwefel and Reinhard Männer, editors, *Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, pages 124–128, Berlin, Heidelberg, 1991. Springer.

[58] John R Koza and Riccardo Poli. Genetic programming. In *Search methodologies*, pages 127–164. Springer, Boston, MA, USA, 2005.

[59] Marek Krętowski. A Memetic Algorithm for Global Induction of Decision Trees. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 531–540, Berlin, Heidelberg, 2008. Springer.

[60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. Burges, L. Bottou,

and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[61] Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Jure Leskovec. Interpretable & explorable approximations of black box models, 2017.

[62] Balaji Lakshminarayanan. *Decision trees and forests: a probabilistic perspective*. PhD thesis, University College London, 2016.

[63] P Larranaga, R Etxeberria, JA Lozano, JM Pena, JM Pe, et al. Optimization by learning and simulation of Bayesian and Gaussian networks. Technical report, University of the Basque Country, 1999.

[64] Pedro Larrañaga and Jose A Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media, 2001.

[65] Martin Lauer and Martin A. Riedmiller. An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems. In *7th International Conference on Machine Learning*, page 535–542, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[66] Zachary C. Lipton. The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability is Both Important and Slippery. *Queue*, 16(3):31–57, June 2018.

[67] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Morgan Kaufmann, San Francisco (CA), 1994.

[68] Jingbin Liu, Xinyang Gu, Shuai Liu, and Dexiang Zhang. Soft Q-network. *arXiv:1912.10891 [cs]*, 2019. arXiv:1912.10891 [cs].

[69] Vadim Liventsev, Aki Härmä, and Milan Petković. BF++: a language for general-purpose neural program synthesis. *arXiv:2101.09571 [cs]*, January 2021. arXiv: 2101.09571.

[70] Xavier Llorà and Josep M. Garrell. Evolution of Decision Trees. In *4th Catalan Conference on Artificial Intelligence*, pages 115–122, 2001.

[71] Xavier Llorà and Josep M. Garrell. Knowledge-Independent Data Mining with Fine-Grained Parallel Evolutionary Algorithms. In *3rd Annual Conference on Genetic and Evolutionary Computation*, page 461–468, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[72] Pascal Lomoro, Francesco Verde, Filippo Zerboni, Igino Simonetti, Claudia Borghi, Camilla Fachinetti, Anna Natalizi, and Alberto Martegani. COVID-19 pneumonia manifestations at the admission on chest ultrasound, radiographs, and CT: single-center study and comprehensive radiologic literature review. *European Journal of Radiology Open*, 7, jan 2020.

[73] Quintino Francesco Lotito, Leonardo Lucio Custode, and Giovanni Iacca. A signal-centric perspective on the evolution of symbolic communication. In *Genetic and Evolutionary Computation Conference - Companion*, pages 120–128. Association for Computing Machinery, New York, NY, USA, June 2021.

[74] Scott Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. *arXiv:1705.07874 [cs, stat]*, November 2017. arXiv: 1705.07874.

[75] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[76] Sergio Valcarcel Macua, Aleksi Tukiainen, Daniel García-Ocaña Hernández, David Baldazo, Enrique Munoz de Cote, and Santiago Zazo. Diff-DAC: Distributed Actor-Critic for Average Multitask Deep Reinforcement Learning. *arXiv:1710.10363*, 2019. arXiv:1710.10363.

[77] Mikel Malagon and Josu Ceberio. Evolving Neural Networks in Reinforcement Learning by means of UMDAc. *arXiv:1904.10932 [cs]*, April 2019. arXiv: 1904.10932.

[78] Laetitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. Hysteretic Q-learning : an algorithm for Decentralized Reinforcement Learning in Cooperative Multi-Agent Teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 64–69, New York, NY, USA, 2007. IEEE/RSJ.

[79] Andrew Kachites Mccallum and Dana Ballard. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, The University of Rochester, Eastman School of Music, 1996.

[80] Wenjia Meng, Qian Zheng, Long Yang, Pengfei Li, and Gang Pan. Qualitative Measurements of Policy Discrepancy for Return-Based Deep Q-Network. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–7, 2019.

[81] Federico Mento and Libertario Demi. On the Influence of Imaging Parameters on Lung Ultrasound B-line Artifacts, in vitro study. *Journal of the Acoustical Society of America*, 148(2):975–983, 2020.

[82] Federico Mento, Tiziano Perrone, Anna Fiengo, Andrea Smargiassi, Riccardo Inchingolo, Gino Soldati, and Libertario Demi. Deep learning applied to lung ultrasound videos for scoring COVID-19 patients: A multicenter study. *The Journal of the Acoustical Society of America*, 149(5):3626–3634, 2021.

[83] Federico Mento, Gino Soldati, Renato Prediletto, Marcello Demi, and Libertario Demi. Quantitative Lung Ultrasound Spectroscopy Applied to the Diagnosis of Pulmonary Fibrosis: The First Clinical Study. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 67(11):2265–2273, 2020.

[84] Risto Miikkulainen, Olivier Francon, Elliot Meyerson, Xin Qiu, Darren Sargent, Elisa Canzani, and Babak Hodjat. From prediction to prescription: evolutionary optimization of nonpharmaceutical interventions in the COVID-19 pandemic. *IEEE Transactions on Evolutionary Computation*, 25(2):386–401, 2021.

[85] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, December 2013. arXiv:1312.5602 [cs] version: 1.

[86] Kaustav Mohanty, John Blackwell, Thomas Egan, and Marie Muller. Characterization of the Lung Parenchyma Using Ultrasound Multiple Scattering. *Ultrasound in Medicine and Biology*, 43(5):993–1003, may 2017.

[87] David J Montana. Strongly typed genetic programming. *Evolutionary computation*, 3(2):199–230, 1995.

[88] Heinz Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1997.

[89] Sreerama K Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of artificial intelligence research*, 2:1–32, 1994.

[90] Antonio Nouvenne, Andrea Ticinesi, Alberto Parise, Beatrice Prati, Marcello Esposito, Valentina Cocchi, Emanuele Crisafulli, Annalisa Volpi, Sandra Rossi, Elena Giovanna Bignami, Marco Baciarello, Ettore Brianti, Massimo Fabi, and Tiziana Meschi. Point-of-Care Chest Ultrasonography as a Diagnostic Resource for COVID-19 Outbreak in Nursing Homes. *Journal of the American Medical Directors Association*, 21(7):919–923, jul 2020.

[91] Declan Oller, Tobias Glasmachers, and Giuseppe Cuccu. Analyzing Reinforcement Learning Benchmarks with Random Weight Guessing. *arXiv:2004.07707 [cs, stat]*, April 2020. arXiv: 2004.07707.

[92] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P. How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *34th International Conference on Machine Learning - Volume 70*, pages 2681–2690, Sydney, NSW, Australia, August 2017. JMLR.org.

[93] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with Large Scale Deep Reinforcement Learning, December 2019. arXiv:1912.06680 [cs, stat].

[94] Afshin OroojlooyJadid and Davood Hajinezhad. A Review of Cooperative Multi-Agent Deep Reinforcement Learning. *arXiv:1908.03963*, 2020. arXiv:1908.03963.

[95] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[96] Rohan Paleja, Yaru Niu, Andrew Silva, Chace Ritchie, Sugju Choi, and Matthew Gombolay. Learning Interpretable, High-Performing Policies for Autonomous Driving, May 2022. arXiv:2202.02352 [cs].

[97] Qian-Yi Peng, Xiao-Ting Wang, Li-Na Zhang, and Chinese Critical Care Ultrasound Study Group (CCUSG). Findings of lung ultrasonography of novel corona virus pneumonia during the 2019–2020 epidemic. *Intensive Care Medicine*, 46(5):849–850, 2020.

[98] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning. *arXiv:1910.00177 [cs, stat]*, October 2019. arXiv: 1910.00177.

[99] Diego Perez, Miguel Nicolau, Michael O'Neill, and Anthony Brabazon. Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution. In

Applications of Evolutionary Computation, pages 123–132. Springer, Berlin, Heidelberg, 2011.

[100] Tiziano Perrone, Gino Soldati, Lucia Padovini, Anna Fiengo, Gianluca Lettieri, Umberto Sabatini, Giulia Gori, Federica Lepore, Matteo Garolfi, Ilaria Palumbo, Riccardo Inchingolo, Andrea Smargiassi, Libertario Demi, Elisa Eleonora Mossolani, Francesco Tursi, Catherine Klersy, and Antonio Di Sabatino. A New Lung Ultrasound Protocol Able to Predict Worsening in Patients Affected by Severe Acute Respiratory Syndrome Coronavirus 2 Pneumonia. *Journal of Ultrasound in Medicine*, nov 2020.

[101] Erika Poggiali, Alessandro Dacrema, Davide Bastoni, Valentina Tinelli, Elena Demichele, Pau Mateo Ramos, Teodoro Marcianò, Matteo Silva, Andrea Vercelli, and Andrea Magnacavallo. Can Lung US Help Critical Care Clinicians in the Early Diagnosis of Novel Coronavirus (COVID-19) Pneumonia? *Radiology*, 295(3):E6–E6, mar 2020.

[102] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. Coevolutionary Principles., 2012. Citeseer.

[103] Mitchell A. Potter and Kenneth A. De Jong. A cooperative coevolutionary approach to function optimization. In *Parallel Problem Solving from Nature — PPSN III*, pages 249–257, Berlin, Heidelberg, 1994. Springer.

[104] Larry D Pyeatt and Adele E Howe. Decision Tree Function Approximation in Reinforcement Learning. Technical report, Colorado State University, 1998.

[105] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986.

[106] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[107] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners.

[108] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents, April 2022. arXiv:2204.06125 [cs] version: 1.

[109] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-Shot Text-to-Image Generation, February 2021. arXiv:2102.12092 [cs].

[110] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-Agnostic Interpretability of Machine Learning. *arXiv:1606.05386 [cs, stat]*, June 2016. arXiv: 1606.05386.

[111] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *arXiv:1602.04938 [cs, stat]*, August 2016. arXiv: 1602.04938.

[112] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[113] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models, April 2022. arXiv:2112.10752 [cs].

[114] Aaron M. Roth, Nicholay Topin, Pooyan Jamshidi, and Manuela Veloso. Conservative Q-Improvement: Reinforcement Learning for an Interpretable Decision-Tree Policy. *arXiv:1907.01180*, 2019. arXiv:1907.01180.

[115] S Roy, W Menapace, S Oei, B Luijten, E Fini, C Saltori, I Huijben, N Chennakeshava, Federico Mento, A Sentelli, E Peschiera, R Trevisan, G Maschietto, E Torri, R Inchingolo, A Smargiassi, G Soldati, P Rota, A Passerini, R J G Van Sloun, E Ricci, and L Demi. Deep learning for classification and localization of COVID-19 markers in point-of-care lung ultrasound. *IEEE Transactions on Medical Imaging*, 39(8):2676–2687, 2020.

[116] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, May 2019. Number: 5 Publisher: Nature Publishing Group.

[117] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable Machine Learning: Fundamental Principles and 10 Grand Challenges. *arXiv:2103.11251 [cs, stat]*, July 2021. arXiv: 2103.11251.

[118] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys*, 16:1–85, 2022.

[119] Cynthia Rudin and Joanna Radin. Why Are We Using Black Box Models in AI When We Don't Need To? A Lesson From An Explainable AI Competition. *Harvard Data Science Review*, 1(2), November 2019.

[120] Conor Ryan, Jj Collins, and Michael O Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *European Conference on Genetic Programming*, volume 1391, pages 83–96. Springer, Berlin, Heidelberg, 1998.

[121] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning, 2017. arXiv:1703.03864.

[122] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *arXiv:1902.04043*, December 2019. arXiv:1902.04043.

[123] Tuomas W. Sandholm and Robert H. Crites. On multiagent Q-learning in a semicompetitive domain. In *Adaption and Learning in Multi-Agent Systems*, volume 1042, pages 191–205. Springer, Berlin, Heidelberg, 1996.

[124] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, and et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, Dec 2020.

[125] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*, 588(7839):604–609, December 2020. arXiv:1911.08265 [cs, stat].

[126] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and Offline Reinforcement Learning by Planning with a Learned Model, 2021. arXiv:2104.06294.

[127] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems*, 42(3):1–21, 2017.

[128] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, August 2017. arXiv: 1707.06347.

[129] Andrew Silva, Taylor Killian, Ivan Dario Jimenez Rodriguez, Sung-Hyun Son, and Matthew Gombolay. Optimization Methods for Interpretable Differentiable Decision Trees in Reinforcement Learning. *arXiv:1903.09338 [cs, stat]*, pages 1855–1865, June 2020. arXiv: 1903.09338.

[130] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[131] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, April 2015. arXiv:1409.1556 [cs].

[132] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to Communicate at Scale in Multiagent Cooperative and Competitive Tasks. *arXiv:1812.09755*, 2018. arXiv:1812.09755.

[133] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 180–186, 2020.

[134] Gino Soldati and Marcello Demi. The use of lung ultrasound images for the differential diagnosis of pulmonary and cardiac interstitial pathology. *Journal of Ultrasound*, 20, apr 2017.

[135] Gino Soldati, Andrea Smargiassi, Riccardo Inchingolo, Danilo Buonsenso, Tiziano Perrone, Domenica Federica Briganti, Stefano Perlini, Elena Torri, Alberto Mariani, Elisa Eleonora Mossolani, Francesco Tursi, Federico Mento, and Libertario Demi. Is There a Role for Lung Ultrasound During the COVID-19 Pandemic? *Journal of Ultrasound in Medicine*, 39(7):1459–1462, 2020.

[136] Gino Soldati, Andrea Smargiassi, Riccardo Inchingolo, Danilo Buonsenso, Tiziano Perrone, Domenica Federica Briganti, Stefano Perlini, Elena Torri, Alberto Mariani, Elisa Eleonora Mossolani, Francesco Tursi, Federico Mento, and Libertario Demi. Proposal for International Standardization of the Use of Lung Ultrasound for Patients With COVID-19. *Journal of Ultrasound in Medicine*, 39(7):1413–1419, 2020.

[137] Peter Stone and Manuela Veloso. Multiagent Systems: A Survey from a Machine Learning Perspective:. Technical report, Defense Technical Information Center, Fort Belvoir, VA, December 1997.

[138] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In *17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2085–2087, Stockholm, Sweden, July 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[139] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA, 2018.

[140] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent Cooperation and Competition with Deep Reinforcement Learning. *arXiv:1511.08779*, November 2015. arXiv:1511.08779.

[141] Ming Tan. *Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents*, page 487–494. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[142] Yujin Tang, Duong Nguyen, and David Ha. Neuroevolution of self-interpretable agents. *Genetic and Evolutionary Computation Conference*, Jun 2020. arXiv:2003.08165.

[143] Justin K Terry, Benjamin Black, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Clemens Dieffendahl, Niall L Williams, Yashas Lokesh, Caroline Horsch, et al. Pettingzoo: Gym for multi-agent reinforcement learning, 2020. arXiv:2009.14471.

[144] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE/RSJ, 2012.

[145] Alexander Trott, Sunil Srinivasa, Douwe van der Wal, Sebastien Haneuse, and Stephan Zheng. Building a foundation for data-driven, interpretable, and robust policy design using the ai economist, 2021.

[146] William TB Uther and Manuela M Veloso. Tree based discretization for continuous state space reinforcement learning. In *AAAI/IAAI*, pages 769–774, 1998.

[147] Yuri Viazovetskyi, Vladimir Ivashkin, and Evgeny Kashin. StyleGAN2 Distillation for Feed-Forward Image Manipulation. In Andrea Vedaldi, Horst Bischof, Thomas

Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, volume 12367, pages 170–186. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science.

[148] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, November 2019. Number: 7782 Publisher: Nature Publishing Group.

[149] Marco Virgolin, Andrea De Lorenzo, Eric Medvet, and Francesca Randone. Learning a Formula of Interpretability to Learn Interpretable Formulas. *arXiv:2004.11170 [cs, stat]*, pages 79–93, May 2020. arXiv: 2004.11170.

[150] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning, 2016. arXiv:1511.06581.

[151] Christopher Watkins. *Learning From Delayed Rewards*. PhD thesis, King's College, May 1989.

[152] Changyang Xing, Qiaoying Li, Hong Du, Wenzhen Kang, Jianqi Lian, and Lijun Yuan. Lung ultrasound findings in patients with COVID-19 pneumonia. *Critical Care*, 24(1):174, 2020.

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

[154] Junyu Xuan, Jie Lu, Zheng Yan, and Guangquan Zhang. Bayesian Deep Reinforcement Learning via Deep Kernel Learning. *International Journal of Computational Intelligence Systems*, 12(1):164–171, November 2018.

[155] W Xue, C Cao, J Liu, Y Duan, H Cao, J Wang, X Tao, Z Chen, M Wu, J Zhang, H Sun, Y Jin, X Yang, R Huang, F Xiang, Y Song, M You, W Zhang, L Jiang,

Z Zhang, S Kong, Y Tian, L Zhang, D Ni, and M Xie. Modality alignment contrastive learning for severity assessment of COVID-19 from lung ultrasound and clinical information. *Medical Image Analysis*, 69, 2021.

[156] Jiachen Yang, Alireza Nakhaei, David Isele, Kikuo Fujimura, and Hongyuan Zha. CM3: Cooperative Multi-goal Multi-stage Multi-agent Reinforcement Learning. *arXiv:1809.05188*, January 2020. arXiv:1809.05188.

[157] Kosuke Yasukawa and Taro Minami. Point-of-Care Lung Ultrasound Findings in Patients with COVID-19 Pneumonia. *The American Journal of Tropical Medicine and Hygiene*, 102(6):1198–1202, 2020.

[158] Chao Yu, Jiming Liu, and Shamim Nemati. Reinforcement Learning in Healthcare: A Survey. *arXiv:1908.08796*, April 2020. arXiv:1908.08796.

[159] Gemin Zhang, Jie Zhang, Bowen Wang, Xionglin Zhu, Qiang Wang, and Shiming Qiu. Analysis of clinical characteristics and laboratory findings of 95 cases of 2019 novel coronavirus pneumonia in Wuhan, China: a retrospective analysis. *Respiratory research*, 21(1):74, mar 2020.

[160] Lianmin Zheng, Jiacheng Yang, Han Cai, Ming Zhou, Weinan Zhang, Jun Wang, and Yong Yu. MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1):8222–8223, April 2018.