Multi-Objective Genetic Programming for Explainable Reinforcement Learning

Mathurin Videau, Alessandro Ferreira Leite, Olivier Teytaud, Marc Schoenauer

Multi-Objective Genetic Programming for Explainable Reinforcement Learning

Mathurin Videau[1⋆], Alessandro Leite[1], Olivier Teytaud[2], and Marc Schoenauer[1]

[1] TAU, Inria Saclay, LISN
[2] Meta AI Research

Abstract. Deep reinforcement learning has met noticeable successes recently for a wide range of control problems. However, this is typically based on thousands of weights and non-linearities, making solutions complex, not easily reproducible, uninterpretable and heavy. The present paper presents genetic programming approaches for building symbolic controllers. Results are competitive, in particular in the case of delayed rewards, and the solutions are lighter by orders of magnitude and much more understandable.

Keywords: Genetic Programming · Reinforcement Learning · Explainable Reinforcement Learning (XRL) · Genetic Programming Reinforcement Learning (GPRL)

1 Introduction

Interpretability plays an important role in the adoption and acceptance of machine learning (ML) models. A model is normally considered explainable if its users can understand the reasons for an output of a given input and if they can include it into their decision processes. When a model lacks understandability, its usefulness reduces since it may be hard to ensure correctness. While the literature has mostly focused on developing different techniques to explain supervised and unsupervised machine learning models and neural networks, reinforcement learning (RL) methods also require explanations to be adopted by a broad audience. In RL, an agent learns a particular behavior through repeated trial-and-error interactions with a specific environment. At each time step, the agent observes the state of the environment, chooses an action, and in return, receives a reward [23,51]. RL methods can be difficult to understand as they depend on the way the reward functions are defined, on how the states are encoded, and on the chosen policy. For this reason, different explainable artificial intelligence (XAI) methods have been proposed over the last years.

Explainable artificial intelligence (XAI) comprises a set of methods that aim to highlight the process that a machine learning model employed to output a prediction using terms interpretable by humans [13,3]. Interpretability in this case means the degree to which humans understand the causes for a model output [36].

⋆ Now at Meta AI Research

It contrasts with the concept of a black-box which cannot be understood by humans. Although, interpretability and explainability are usually used interchangeably, the former is an intrinsic property of a predictive model, whereas the latter is modeled as *post-hoc explanations* and thus, separated from the predictive model [18]. Genetic programming, linear regression, and decision trees up to a certain depth are examples of interpretable models, while neural networks and ensemble methods are considered black-boxes. *Post-hoc explanations* can be global or local. While *global explanations* describe the overall logic of a model independently of input queries, *local explanations* try to elucidate the reasons for an output produced for a specific input query. Post-hoc explainers can be agnostic or dependent of the underlying approach used by the black-box. *Feature importance* is an example of the former, and is hence used by many explanation methods to describe how crucially each feature contributes to a prediction, thus helping to understand the underlying process. In other words, they use the weights of each feature as a proxy for explicability. Examples include Local Interpretable Model-Agnostic Explanations (LIME) [41] and SHapley Additive exPlanations (SHAP) [32]. LIME explains a black-box by weighting the distance between the predictions made by an interpretable model and by the black-box for a given query instance. Then, it outputs the contribution of each feature for the prediction. SHAP, on the other hand, relying on cooperative game theory, describes how to distribute the total gains of the game to players according to their contributions, and similarly, it outputs the importance of each feature for the prediction. Other approaches include partial dependence plot (PDP) [7] and saliency map [47,48], for instance.

This paper focuses on *explainable reinforcement learning (XRL)*. We propose and analyze genetic programming (GP)-based RL (GPRL) policies for different control tasks (Section 3). The policies are programs representing symbolic expressions. In this work, the proxy used for interpretability is the size of the expression of the policy, and their terms (i.e., features). Experimental results (Section 4) show that GP-based RL policies can outperform state-of-the-art neural networks on various tasks, while being much more interpretable. Furthermore, they also demonstrate that traditional GP methods require additional support to solve motion tasks. Map-Elites [38] demonstrates to be a good option without deteriorating the interpretability of the policies, in contrast with imitation learning approaches. Source code is available at *gitlab.inria.fr/trust-ai/xrl/gpxrl*.

2 Related Work

Recent years have seen an increasing interest in explaining reinforcement learning policies. It is mainly due to the advancement of deep reinforcement learning (DRL) methods. On the one hand, many works have used decision trees [14,30,5,12,43] to try to understand how the state space is segmented into different actions: decision trees distill the behavior of the neural networks [30,5,12]. On the other hand, other studies rely on symbolic expressions to represent DRL policies either by program synthesis [52,22] or by symbolic regression meth-

ods [27,22,53,54,29]. The former uses neural networks to generate the programs while the latter relies mostly on genetic programming due to its historical performance when employed as a policy regularizer [27,22,53,54]. For example, Verma et al. [52] use the base *sketch* to generate programs that try to mimic a neural network using the mean squared error as proxy metric. Their results show that in addition to be interpretable, the policies can generalize better than the neural networks. Liventsev et al. [31] use the *Brainfuck* language to describe policies that are driven by both genetic programming and by a recurrent neural network (RNN). Their results show that although this combination lead to better program structure, they underperform deep reinforcement learning policies. Also, they might bring sparsity, but not really explainability. Recently, symbolic regression methods have been used to explain complex RL tasks. For instance, Wilson et al. [53] showed that GP-based policies can outperform neural networks in some specific Atari games tasks. However, their number of operations make them hard to understand. Also, on Atari games, Kelly et al. [24] used Tangled Program Graphs (TPG) to evolve a symbolic program that solves all the games. Much less complex than their deep neural network (DNN) challengers, their programs allow some explainability of their behavior, but remain difficult to understand globally. Hein et al. [22] demonstrated that similar performance can be observed on continuous control tasks, but their approach need sample trajectories to derive the GP policies. Kubalík et al. [27], on the other hand, got analytical expressions of the value function by framing the problem as a fixed point of the Bellman equation. Therefore, this approach demands knowledge about the transition function to enable learning. Landajuela et al. [29] showed that for continuous control tasks, symbolic regression methods helped by neural networks may outperform neural networks-only policies. In addition, in the cases where the dynamics of the tasks are known, the authors proved the stability of the symbolic policies. Nevertheless, for problems with multidimensional actions, this process requires pre-trained neural networks.

Table 1 summarizes the main existing works on GP-based RL policies. While literature has concentrated on evaluating the performance of RL policies for specific task environments, this work focuses on studying both the performance and interpretability of GP-based RL policies on various control tasks. Furthermore, we also investigate strategies to handle some identified limitations of traditional GP approaches when dealing with complex control tasks.

3 Explainable Reinforcement Learning using GP

Reinforcement learning (RL) is usually formalized as a Markov decision process (MDP), i.e., a tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces. At each discrete time step t, an agent observes the system state $s_t \in \mathcal{S}$ and takes an action $a_t \in \mathcal{A}$. The system state then changes to state s_{t+1} with probability $p(s_{t+1}|s_t, a_t)$ and the agent receives a reward $r(s_t, a_t, s_{t+1}) \in \mathbb{R}$, for a reward function $r : \mathcal{S} \times \mathcal{A} \times S \mapsto \mathbb{R}$. A *policy* is a function from $\mathcal{S}$ into $\mathcal{A}$ that

Table 1: Summary of interpretable GP-based RL policies

Work	Approach	Environment	Objective(s)
[33]	MCTS	Mountain Car Acrobot	Cumulative mean reward + complexity
[52]	NN + Bayesian opt	TORCS	Cumulative reward
[29]	Deep Symbolic Regression	OpenAI Gym: Classical control, Box2D, Mujocco	Cumulative reward
[31]	RNN + GP	OpenAI Gym (4 envs)	Cumulative reward
[54]	NN + EFS	Acrobot Mountain Car Industrial Benchmark[21]	MSE
[53]	Mixed type Cartesian GP	Atari	Cumulative reward
[24]	Tangle Program Graphs	Atari	Cumulative reward
[27]	Multi-Gene GP	1-DOF, 2-DOF Magman	Bellman MSE
[22]	Tree GP	Montain Car, CartPole Industrial Benchmark [21]	Cumulative reward + complexity
This work	**Tree GP + Linear GP**	**OpenAI Gym: Classical control, Box2D, Mujoco**	**Cumulative reward + complexity**

associates to each state the action to be taken in that state (*only deterministic policies are considered in this work*).

The solution of an RL problem with time horizon $H \in [0, +\infty]$ and a discount factor $\gamma \in [0, 1]$ is a policy π^* that maximizes the expected discounted cumulative reward over all starting states $s_0 \in \mathcal{S}$ and all realizations of the trajectory starting from s_0 according to the stochastic state transition p, hereafter called its *score*, and defined as follows.

$$\text{score}(\pi) = \mathbb{E}_{s_0}\left[\sum_{t=0}^{H-1} \gamma^t R(s_t, \pi(s_t))\right] \tag{1}$$

The expectation in Eq. (1) above is estimated from several Monte-Carlo simulations of the policy at hand with random starting states (more below).

Explainability Because there is no formal definition of explainability, a common practice is to use as a proxy the complexity of the obtained solutions [13,18]. *The size of an expression and the number of selected features in its formula determine the complexity of a policy.* Each operator and terminal is assigned a given elementary complexity (see Table 2), and the *global complexity is the sum over the whole formula of these elementary complexities [22].* Furthermore, two ways are experimented with in this work to obtain simple solutions: two-objective opti-

mization, with cumulated reward and complexity as objectives; biased operators that favor removal over additions of terms in the solution (details in Section 4.1).
Bandit-like Evaluation Strategy GP policies have heterogeneous sizes and computational costs. In order to obtain an accurate estimate of their scores, we try to allocate our simulation budget as efficiently as possible, and in particular to run more Monte-Carlo simulations for promising individuals, using a multi-armed-bandit-like strategy as follows: a total budget T of simulations is allocated at each generation (details below). Each individual of the population (i.e., policy) is attributed an *upper confidence bound (UCB) value* defined by $\bar{x} + c\sqrt{\frac{\ln(n'+T)}{n}}$, where c is an exploration constant, $\bar{x}$ is the mean score of the policy and n' is an offset accounting for past simulations of this policy prior to this generation, and n the number of times this policy was simulated over all generations. Policies are chosen to be simulated based on this UCB score, by batches of k policies in parallel (i.e., the k ones with the highest UCB values are chosen). The policies with high UCB values are either the ones that have a high score, or the ones that have been simulated a small number of times. This process is repeated until the total budget T is exhausted. Individuals that have never been simulated, and hence have an infinite UCB value, are therefore simulated at least once. After each simulation, the UCB values of the simulated policies are updated: $\bar{x}$ takes into account the new score, and n is incremented by one. Note that in case of multi-objective optimization, this batch strategy is biased toward the best scoring individuals of the Pareto front – but these are the ones of main interest here.

Per generation simulation budget In conjunction with the bandit like strategy presented above, the global simulation budget per generation is gradually increased between generations. The reason is that the differences in scores between different policies are likely to decrease, and the variance of the scores needs to be decreased to improve the robustness of the ranking among policies when their differences become more and more subtle. The detailed parametrization of this scheduling is available in the code source (*bit.ly/3JdlHXx*).

4 The Experiments

The first goal of the experiments is to quantify the score/interpretability trade-offs when using GP compared to the traditional direct policy search strategies usually embraced by RL approaches. Different benchmarks are used, with different levels of difficulty, and GP are compared with state-of-the-art algorithms, either pertaining to direct policy search, or to classical RL techniques. Furthermore, two different GP representations are considered in that respect, classical parse trees [26], and linear GP [8]. Likewise, as the literature lacks a recognized measure of interpretability, we will also deeply analyze and compare the results beyond complexity, regarding feature importance on the one hand, and exact analytical policies on the other hand.

4.1 GP Representations

We experimented with two GP representations, and implemented them in `python` using the `DEAP` library [16].

Tree-based GP The first representation is the standard parse tree [26], with *one point crossover*, and a *mutation operator* that either adds Gaussian noise to a random ephemeral constant with probability 0.3, or replaces a random subtree with a randomly generated one (with probability 0.7). We used non-dominated sorting genetic algorithm (NSGA)-II *non-dominated sorting tournament selection* based on the two objectives, score and complexity. But some RL tasks require multiple continuous actions as output, whereas trees only have one output. In such contexts, individuals are teams of trees, where each tree corresponds to one continuous action. All variation operators are performed on one of the trees of the team, and trees with same index are crossed-over. Unfortunately, this results in completely independent action policies, something rather inefficient.

Linear GP We thus moved to linear GP [8], that can consider multiple continuous actions natively (as different registers). In such context, shared features can arise, improving both the score and the complexity of the policies. Standard *crossovers* for linear GP are used (i.e., exchange of a segment of instructions) and *mutation* is either an insertion or a deletion of a random instruction, or a random modification of an existing instruction. Tournament selection with tournament size of five is used. Furthermore, the probability of instruction removal is twice that of an insertion, creating a bias toward small programs (though NSGA-II would also be a viable alternative, not implemented at the moment).

Hyper-parameters For both representations, evolution's parameters can be found in Table 3. The optimization ends after the fixed number of generations is reached. In both setups, it was experimentally assessed that high crossover rates did not improve the score of the resulting policies, while increasing their sizes: a low (or zero) crossover rate is hence used throughout this work.

Finally, the choice between (μ, λ) and $(\mu + \lambda)$ strategies was based on preliminary experiments. On the one hand, (μ, λ) tends to be more robust to noise but consistently forgets the best solution between generations, which produces high variance and non-monotonic convergence. On the other hand, $(\mu + \lambda)$ can find better solutions, but it is more subject to noise if individuals are not sufficiently simulated. Indeed, some solutions could have good fitness just because of a few lucky simulations. In our bandit like setup, this issue is less likely to occur since the individuals are continually tested throughout the evolution process. So, in the following experiments, we used the $(\mu + \lambda)$ schema. Also, these experiments showed us that the exploration constant c of the bandit doesn't have that much impact on the results, and it was set by default to $\sqrt{2}$.

4.2 Benchmarks and Evaluation

Open AI gym [9] was used to evaluate all policies here, on three different control environments: classical control, Mujoco, and Box2D. While classic control environment proposes easy tasks, Box2D and Mujoco offer more complex ones.

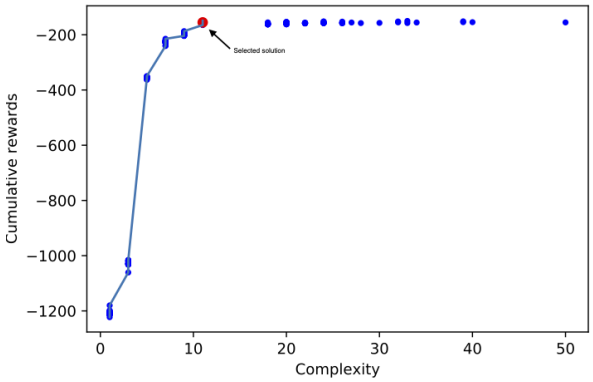


Fig. 1: A Pareto front found for the Pendulum, and the returned solution

Choosing the solution In the case of multi-objective optimization (for tree GP), the returned solution is manually chosen from the Pareto front as the one having the best complexity while being only slightly sub-optimal in terms of score. An example of such a selected solution can be seen in Fig. 1. For linear GP, the best individual according to the score, is returned. In all cases, the score of the returned solution is computed through 1000 independent simulations. Three runs have been performed for each setting (i.e., environment and hyper-parameters as in Table 3), and the best of the three is used in Table 2 and Fig. 2 (low variations between runs were observed). There is also a Colab Notebook showing the efficiency and portability of the symbolic policies presented in Table 2 at *shortest.link/VHv*.

4.3 Baselines

We compare the evolved GP-based policies with the ones obtained by neural network and Nested Monte-Carlo Search, two state-of-the-art approaches in RL and games. What is called the "neural network" policy below is in fact the best performing policy between Proximal Policy Optimization (PPO) [46], Soft Actor Critic (SAC) [19], or Advantage Actor Critic (A2C) [37] algorithms. PPO is an on-policy strategy that optimizes a first-order approximation of the expected reward while cautiously ensuring that it stays closed to the real value. SAC, on the other hand, is an off-policy algorithm that balances between the expected return and the entropy. Finally, A2C is an asynchronous gradient-based algorithm to turn agents' exploration process into a stationary one. These methods achieved impressive performance when used in different application domains. However, these baselines use a lot of information from the problem, including rewards at each time step. As this information is not necessarily available in all real-world settings, we add other methods which use only the total reward per episode, as the GP methods proposed in this work. These methods include Nested Monte-Carlo

Search (NMCS) [10] and direct policy search (DPS) [34,45,49,28]. The "DPS" baseline below reports the best-performing DPS between CMA [20] or NGOpt [35], Population control [6], Differential Evolution [50], PSO [25], Meta-Models [4], or complex combinations that are proposed in Nevergrad [40], including NoisyRL, SpecialRL, and MixDeterministicRL, defined as follows. MixDeterministicRL runs PSO, GeneticDE and DiagonalCMA and keeps the best of the three for the second half of the budget. NoisyRL uses a chaining of a MixDeterministicRL plus a method for fine-tuning robustly to noise, and SpecialRL uses a chaining of MixDeterministicRL combined with population control for fine-tuning.

In order to reduce the complexity of DPS controllers, we tested (i) various architectures of neural networks as in [35,40] (e.g., shallow, semi-deep, deep), (ii) different sizes of hidden layers, (iii) different optimization methods, and (iv) different initial step-sizes[3], keeping and plotting only the best result. In spite of being the most versatile tools with minimum constraints on the environment, we shall see that GP and DPS are competitive, while GP controllers have the lowest complexity.

5 Experimental Results

5.1 Quantitative Analysis

Table 2 shows the cumulative rewards of the policies in the various environments. GP-based policies outperform neural networks for simple and non-motion tasks. For the motion tasks, GP-based policies are obviously trapped in some local optimum. Indeed, they only manage to keep the agent in the upright position as much as possible. It is due to its high variance compared to the ones obtained by a neural network, as can be seen in Fig. 3.

We obtained good and relatively compact controllers by DPS, i.e., methods only using the total cumulated reward. In short, the scores are satisfying, competitive with PPO, SAC, and NMCS (sometimes better and sometimes worse). On the other hand, in spite of testing many architectures and optimization methods, DPS failed to compete with the GP-based approaches in terms of complexity, as shown in Fig. 2.

Beyond complexity, we evaluated the interpretability of GP-based policies through the number of features used in the symbolic expressions. As can be seen in Fig. 4, for each environment, GP-based policies does figure out the most important features to solve the tasks. Furthermore, they are sometimes easily explainable. For example, for the *LunarLander* environment, the behavior of the agent is Eq. (2): (1) stops the main engine after landing, (2) reduce rocket's speed as it moves close to the platform, (3) stabilize the rocket in its vertical axis and keep it in the center of the platform, and (4) reduce the speed of the rocket close to the center of the platform. We can even notice that the policy ignores the orientation of the rocket when deciding the action to execute, and decide to

[3] This DPS benchmark has been merged in Nevergrad, and our experiments can be reproduced by « python −m nevergrad.benchmark gp −−num_workers=60 −−plot ».

Table 2: Scores (average cumulated rewards), after each policy was re-simulated 1000 times. DPS refers to the best result for moderate complexity of Direct Policy Search by Nevergrad, and NN to the best of PPO, SAC and A2C. For TreeGP and LinearGP we use the best training error of 3 runs and test it; we validated results by rerunning the whole process (including the best of 3) a second time and got similar results.

Environment	# in	# out	DPS	Tree GP	Linear GP	NN	NMCS
Control tasks							
Cartpole	4	2	500.0	500.0	500.0	500.0†	484.27
Acrobot	6	3	**-72.74**	-83.17	-80.99	$-82.98^{\dagger}$	-89.69
MountainCarC0	2	1	**99.4**	99.31**	88.16	94.56‡	97.89
Pendulum	3	1	**-141.9**	-154.36	-164.66	-154.69‡	-210.71
Mujoco							
InvDoublePend	9	1	**9360**	9092.17	9089.50	9304.32‡	–
InvPendSwingUp	5	1	893.3	**893.35**	887.08	891.45‡	–
Hopper	15	3	2094	999.19	949.27	**2604.91**‡	–
Box2D							
LunarLanderC0	8	2	282.5	**287.58**	262.42	269.31†	–
BipedalWalker	24	4	**310.1**	268.85	257.22	299.44*	–
BipedalWalkerHardcore	24	4	8.16	9.25	10.63	**246.79***	–

†PPO [46], ‡SAC[19] *A2C[37]

** Rely on operators' overflow protections, keeping only the best one for the non-linearity

repair this "bug" in the policy. On the other hand, for abstract environments such as the *BipedalWalker* similar analysis is difficult. However, we can still observe that the actions strongly rely on their own articulations and not on the lidar (Equation (4)).

$$\text{main engine:} a_0 = \underbrace{(y > 0)}_{(1)} ? \underbrace{(-0.37y - \dot{y} + 0.1)}_{(2)} : 0 \tag{2}$$

$$\text{side engine: } a_1 = (\underbrace{4(\theta - x)}_{(3)} \underbrace{-\dot{x})4}_{(4)} \tag{3}$$

$$\text{hip1} : a_0 = \text{knee1}.\theta \tag{4}$$

$$\text{knee1} : a_1 = \frac{\text{knee1}.\dot{\theta}}{\text{lidar}_5} \tag{5}$$

$$\text{hip2} : a_2 = \frac{\text{lidar}_7}{\text{knee1}.\dot{\theta}} - \text{hip2}.\theta + \text{hull}.\theta \tag{6}$$

$$\text{knee2} : a_3 = \text{hull}.\theta - \text{knee2}.\theta \tag{7}$$

5.2 Dealing with the Local Minimum Trap

From these experiments, it seems that, on discontinuous search spaces, GP-based policies have a high risk of converging to some local minima. Indeed, since

Operator	Complexity
Variables	1
Constants	1
$+, -, \times$	1
$/$	2
exp, log, $\hat{}$	3
sin, cos	4
$<, >$	4
if	5

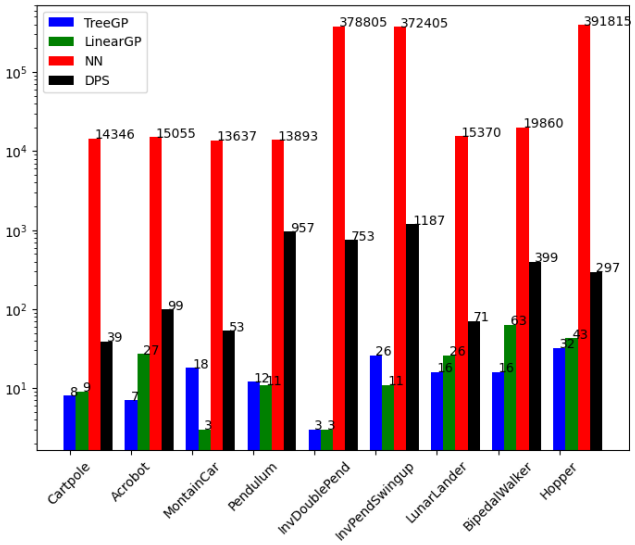
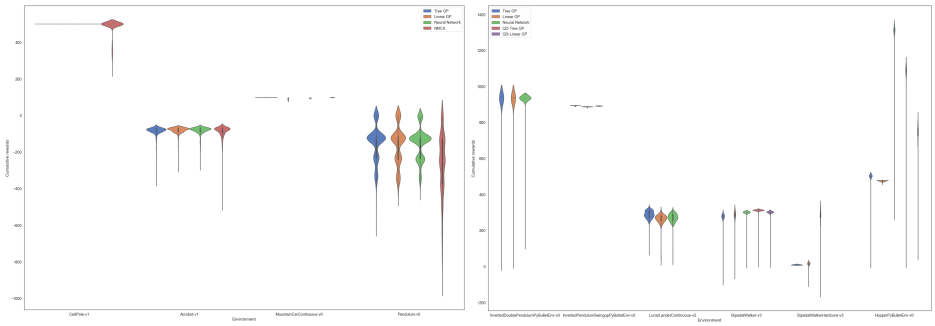


Fig. 2: Complexity of the policies based on expressions' size. Y-axis is in logarithmic scale



(a) Classical (b) Complex

Fig. 3: Cumulative rewards of the solution policies with a more detailed view on the distributions over the post-evolution one thousand simulations. For graphic comprehension, inverse double pendulum and Hopper rewards were divided by ten and two respectively

changing a unique operation leads to important programs' changes, evolution has a hard time to transition from one local optimum to another. As a result, there is a lack of exploration for solving the task at hand. Next sections describe how imitation learning [1,2] and quality diversity (QD) [39] could be used to tackle this issue and at the same time improve the score of symbolic RL policies – and how only QD succeeds.

Table 3: Evolution parameters

Representation	**Tree GP**	**Linear GP**
Classic control environments		
Strategy	NSGA-II	tournament
Function set	$\{+, -, \times, /, \text{if}\}$	$\{+, -, \times, /, \text{if}\}$
Evolution's parameters	$P_{\text{mut}} = 0.9$	$P_{\text{mut}} = 1.0$
	$P_{\text{crossover}} = 0.1$	$P_{\text{ins}} = 0.3, P_{\text{del}} = 0.6$
		$P_{\text{Instruction\_mut}} = 0.5$
		$P_{\text{crossover}} = 0.0$
Parents μ	100	100
Offspring λ	100	100
Number of generations	500	500
Mujoco and Box2D environments		
Strategy	NSGA-II	tournament
Function set	$\{+, -, \times, /, \text{if}, \exp, \log, \sin\}$	$\{+, -, \times, /, \text{if}, \exp, \log, \sin\}$
Evolution's parameters	$P_{\text{mut}} = 0.9$	$P_{\text{mut}} = 1.0$
	$P_{\text{crossover}} = 0.1$	$P_{\text{ins}} = 0.3, P_{\text{del}} = 0.6$
		$P_{\text{Instruction\_mut}} = 0.5$
		$P_{\text{crossover}} = 0.0$
Parents μ	500	500
Offspring λ	500	500
Number of generations	2000	2000

5.2.1 Imitation Learning

Imitation learning aims to accelerate learning by somehow mimicking successful demonstrations of solutions of the task at hand. In this case, the agent acquires knowledge by observing a teacher which performs the target task. The problem can be formulated as a supervised learning approach in which for a data set mapping states to actions, the goal of the agent results in minimizing the error of deviating from the demonstrated output. Formally, from a set of n demonstrations $\mathcal{D} = \{(s_0, a_0), \ldots, (s_n, a_n)\}$ with $s_i, a_i \in \mathcal{S} \times \mathcal{A}$, an agent learns a policy π such that $\pi(s_i) = a_i$, where $\pi(s_i)$ is the predicted action by the teacher. In this work, a pre-trained neural network played the role of a teacher π and a GP program (the learner) tries to imitate it by learning a policy $\hat{\pi}$ that minimizes a loss $\ell(\hat{\pi}(s_i), a_i)$, with ℓ being the mean square error for the continuous tasks and cross-entropy for the discrete ones. Results of two variants of this approach can be seen in Table 4. Behavioral cloning (BC) [44] comprises the basic imitation strategy that uses a static database of trajectories. DAGGER [42], on the other hand, uses a dynamic set of trajectories and runs additional simulations of the selected trajectories. None of these trials led to better result than the one presented in Section 4.1. Furthermore, these trials tend to produce more complex policies, reducing their interpretability (Figure 4). This approach was hence abandoned.

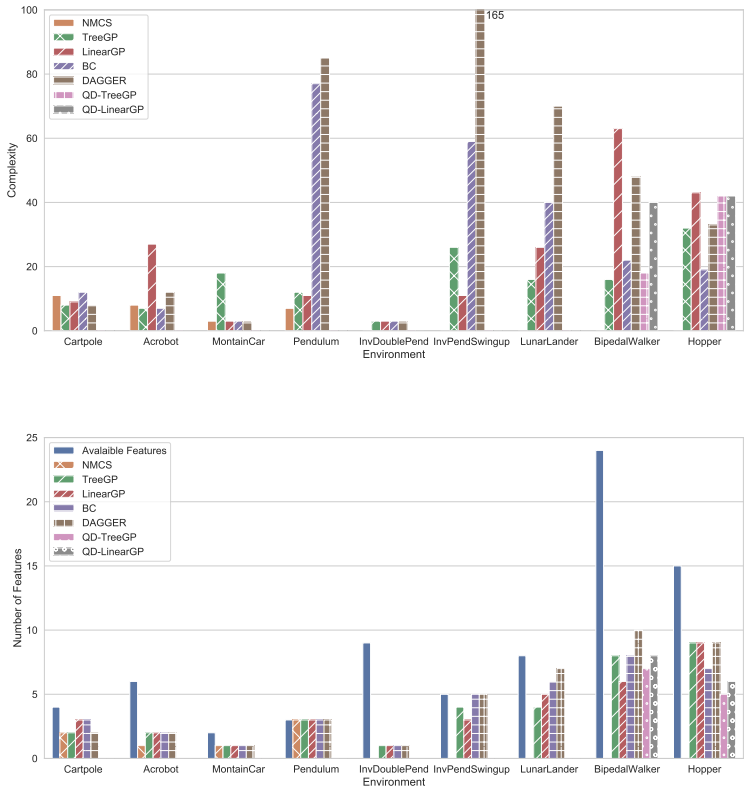


Fig. 4: Complexity of the GP-based policy based on both expression's size and selected features

Table 4: Mean cumulative rewards of policy obtained by Imitation Learning (left) and Map-Elites (right) on 1000 simulations

Environment	**BC**[44]	**DAGGER**[42]
Cartpole	500.0	500.0
Acrobot	-84.17	-81.302
MountainCar	94.06	94.46
Pendulum	-1032.09	-280.22
InvDoublePend	8523.78	8874.56
InvPendSwingUp	-104.01	427.92
LunarLander	247.28	272.62
BipedalWalker	-0.66	32.74
Hopper	59.87	713.78

Environment	**QD-Tree GP**	**QD-Linear**	**NN**
BipedalWalker	311.34	299.64	299.44*
Hopper	2152.19	1450.11	2604.91‡

* A2C [37] ‡ *SAC* [19]

5.2.2 Quality Diversity

Quality diversity is an optimization algorithm that aims to find a set of strong and maximally diverse solutions [39]. Diversity is measured in the behavioral space, considering that all the behaviors are equally important. In this work, in order to maintain and even maximize the diversity in the population, we rely on the Map-Elites algorithm [38] and in particular, a variant that takes into account noisy fitness values [15]. Based on its behavioral features, each individual is placed on a grid. All selections of individuals are independent, and proceed as follows: first, a cell is uniformly selected among the non-empty cells of the grid. If there are more than one individual in the cell, a random one is selected. To avoid too large grids, we use only two behavioral features, discretize them in ten intervals, and limit the number of individuals per cell to ten individuals by removing the worst individuals when new ones are placed there. These aspects, when put together, give a population size of at most one thousand individuals. To estimate the behavioral features and scores, at each generation, each new individual is simulated on three episodes. All the other parameters are the same from the one presented in Table 3 in the Mujoco and Box2D part. Once the Map-ELites algorithm round is finished, a subset of the grid is selected according to a score threshold, and is the initial population of the algorithm presented in Section 4.1, run for a hundred generations. This last part aims to fine-tune the solutions found by Map-Elites and to reduce their complexity by using NSGA-II. All these experiments were done with the `QDpy` library [11]. For *BipedalWalker*, the behavioral features are the mean amplitude of the hip1 and hip2 joints (i.e., the mean of $|s_4|, |s_9|$). For *Hopper*, the behavioral features are the mean amplitude of foot and leg joints (i.e., the mean of $|s_{13}|, |s_{11}|$).

Table 4 shows the results obtained by employing our QD approach. For the two locomotion environments, the score of the policies significantly increased. This improvement is clearly visible for the *Hopper* task. Indeed, in this environment, the policy changed from a static behavior (i.e., tree GP) to a walking one (i.e., QD-tree GP). Furthermore, the policies remain interpretable with respect to their degree of complexity and the number of features as can be observed in Fig. 4.

6 Conclusion and Further Work

In RL, a policy maps the state of an environment to the actions to be taken by an agent [51], and the agent must learn the action-state mapping that maximizes the cumulative rewards. Therefore, some states may be irrelevant or even inadequate for some policies. As a result, they may lead to policies that are hard to understand and explain. Nevertheless, the literature has mostly focused on using traditional machine learning methods and neural networks in order to explain reinforcement learning, whereas it is necessary to deeper understand its functioning and decisions. Likewise, the majority of works in explainable reinforcement learning has focused on specific task environments. In this paper, we investigated the use of GP-based RL policies considering both score and interpretability for several environments simultaneously. Our approach relied

on parse trees [26] and linear GP [8] to represent the programs combined with a multi-arm bandit strategy to allocate the computational budget across the generations. Experimental results on three different types of control environments show that the GP-based RL policy can have score similar to state-of-the-art methods (e.g., neural networks), while still being explainable when considering the size of their expressions, and the selected features. Furthermore, we observed that standard GP methods need help to solve motion tasks correctly, as they stay stuck in local optima. Map-Elites [38] revealed to be an appropriate option without penalizing the interpretability of the policies. Nevertheless, the size of the grids determines the quality of the solutions and the convergence time. Consequently, it may be unsuitable for high-dimension problems. Bayesian optimization may handle convergence issue by selecting the grid to explore [17]. Another alternative comprises in incrementally increases the number of features to train a population to explore both the features and grids' size diversity. Further direction also includes the usage of Tangled Program Graphs (TPG) to enable code reuse. However, it still misses at the moment some native support for continuous actions.

Acknowledgements

This research was partially funded by the European Commission within the HORIZON program (TRUST-AI Project, Contract No. 952060).

References

1. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: ICML. p. 1 (2004)
2. Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robotics and autonomous systems **57**(5), 469–483 (2009)
3. Arrieta, A.B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatilaf, R., Herrerag, F.: Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. IF **58**, 82–115 (2020)
4. Auger, A., Schoenauer, M., Teytaud, O.: Local and global order 3/2 convergence of a surrogate evolutionary algorithm. In: GECCO. p. 8 (2005)
5. Bastani, O., Pu, Y., Solar-Lezama, A.: Verifiable reinforcement learning via policy extraction. arXiv:1805.08328 (2018)
6. Beyer, H.G., Hellwig, M.: Controlling population size and mutation strength by meta-es under fitness noise. In: FOGA. p. 11–24 (2013)
7. Biecek, P., Burzykowski, T.: Explanatory model analysis: explore, explain, and examine predictive models. CRC Press (2021)
8. Brameier, M.F., Banzhaf, W.: Linear genetic programming. Springer (2007)
9. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. arXiv:1606.01540 (2016)
10. Cazenave, T.: Nested monte-carlo search. In: IJCAI (2009)
11. Cazenille, L.: Qdpy: A python framework for quality-diversity. bit.ly/3s0uyVv (2018)

12. Coppens, Y., Efthymiadis, K., Lenaerts, T., Nowé, A., Miller, T., Weber, R., Magazzeni, D.: Distilling deep reinforcement learning policies in soft decision trees. In: CEX Workshop. pp. 1–6 (2019)
13. Doshi-Velez, F., Kim, B.: Towards a rigorous science of interpretable machine learning. arXiv:1702.08608 (2017)
14. Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. JMLR **6**, 503–556 (2005)
15. Flageat, M., Cully, A.: Fast and stable map-elites in noisy domains using deep grids. In: ALIFE. pp. 273–282 (2020)
16. Fortin, F.A., De Rainville, F.M., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: Evolutionary algorithms made easy. JMLR **13**, 2171–2175 (2012)
17. Gaier, A., Asteroth, A., Mouret, J.B.: Data-efficient exploration, optimization, and modeling of diverse designs through surrogate-assisted illumination. In: GECCO. pp. 99–106 (2017)
18. Gilpin, L., Bau, D., Yuan, B., Bajwa, A., Specter, M., Kagal, L.: Explaining explanations: An approach to evaluating interpretability of ML. arXiv:1806.00069 (2018)
19. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: ICML. pp. 1861–1870 (2018)
20. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. ECO **11**(1) (2003)
21. Hein, D., Depeweg, S., Tokic, M., Udluft, S., Hentschel, A., Runkler, T.A., Sterzing, V.: A benchmark environment motivated by industrial control problems. In: IEEE SSCI. pp. 1–8 (2017)
22. Hein, D., Udluft, S., Runkler, T.A.: Interpretable policies for reinforcement learning by genetic programming. Engineering Applications of Artificial Intelligence **76**, 158–169 (2018)
23. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. JAIR **4**, 237–285 (1996)
24. Kelly, S., Heywood, M.I.: Multi-task learning in atari video games with emergent tangled program graphs. In: GECCO. pp. 195–202 (2017)
25. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: IJCNN. pp. 1942–1948 (1995)
26. Koza, J.R.: Genetic Programming: On the Programming of Computers by means of Natural Evolution. MIT Press, Massachusetts (1992)
27. Kubalík, J., Žegklitz, J., Derner, E., Babuška, R.: Symbolic regression methods for reinforcement learning. arXiv:1903.09688 (2019)
28. Kwee, I., Hutter, M., Schmidhuber, J.: Gradient-based reinforcement planning in policy-search methods. In: Wiering, M.A. (ed.) EWRL. vol. 27, pp. 27–29 (2001)
29. Landajuela, M., Petersen, B.K., Kim, S., Santiago, C.P., Glatt, R., Mundhenk, N., Pettit, J.F., Faissol, D.: Discovering symbolic policies with deep reinforcement learning. In: ICML. pp. 5979–5989 (2021)
30. Liu, G., Schulte, O., Zhu, W., Li, Q.: Toward interpretable deep reinforcement learning with linear model u-trees. In: ECML PKDD. pp. 414–429 (2018)
31. Liventsev, V., Härmä, A., Petković, M.: Neurogenetic programming framework for explainable reinforcement learning. arXiv:2102.04231 (2021)
32. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: NeurIPS. pp. 4768–4777 (2017)

33. Maes, F., Fonteneau, R., Wehenkel, L., Ernst, D.: Policy search in a space of simple closed-form formulas: Towards interpretability of reinforcement learning. In: ICDS. pp. 37–51 (2012)
34. Mania, H., Guy, A., Recht, B.: Simple random search provides a competitive approach to reinforcement learning. arXiv:1803.07055 (2018)
35. Meunier, L., Rakotoarison, H., Wong, P.K., Roziere, B., Rapin, J., Teytaud, O., Moreau, A., Doerr, C.: Black-box optimization revisited: Improving algorithm selection wizards through massive benchmarking. IEEE TEVC (2021)
36. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. Artificial intelligence **267**, 1–38 (2019)
37. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: ICML. pp. 1928–1937 (2016)
38. Mouret, J.B., Clune, J.: Illuminating search spaces by mapping elites. arXiv:1504.04909 (2015)
39. Pugh, J.K., Soros, L.B., Stanley, K.O.: Quality diversity: A new frontier for evolutionary computation. Frontiers in Robotics and AI **3**, 40 (2016)
40. Rapin, J., Teytaud, O.: Nevergrad - A gradient-free optimization platform. bit.ly/3g8wghU (2018)
41. Ribeiro, M.T., Singh, S., Guestrin, C.: Why should I trust you? explaining the predictions of any classifier. In: SIGKDD. pp. 1135–1144 (2016)
42. Ross, S., Gordon, G., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: AISTATS. pp. 627–635 (2011)
43. Roth, A.M., Topin, N., Jamshidi, P., Veloso, M.: Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy. arXiv:1907.01180 (2019)
44. Russell, S.: Learning agents for uncertain environments. In: COLT. pp. 101–103 (1998)
45. Schoenauer, M., Ronald, E.: Neuro-genetic truck backer-upper controller. In: IEEE CEC. pp. 720–723 (1994)
46. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv:1707.06347 (2017)
47. Selvaraju, R.R., Cogswell, M., et al.: Grad-CAM: Visual explanations from deep networks via gradient-based localization. In: ICCV. pp. 618–626 (2017)
48. Shrikumar, A., Greenside, P., Kundaje, A.: Learning important features through propagating activation differences. In: ICML. pp. 3145–3153 (2017)
49. Sigaud, O., Stulp, F.: Policy search in continuous action domains: an overview. arXiv:1803.04706 (2018)
50. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. JGO **11**(4), 341–359 (1997)
51. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press, 2nd edn. (2018)
52. Verma, A., Murali, V., Singh, R., Kohli, P., Chaudhuri, S.: Programmatically interpretable reinforcement learning. In: ICML. pp. 5045–5054 (2018)
53. Wilson, D.G., Cussat-Blanc, S., Luga, H., Miller, J.F.: Evolving simple programs for playing atari games. In: GECCO. pp. 229–236 (2018)
54. Zhang, H., Zhou, A., Lin, X.: Interpretable policy derivation for reinforcement learning based on evolutionary feature synthesis. Complex Intell Syst **6**(3), 741–753 (2020)