Naturally Interpretable Control Policies via Graph-Based Genetic Programming

Giorgia Nadizar, Eric Medvet, Dennis G Wilson

Naturally Interpretable Control Policies via Graph-based Genetic Programming

Giorgia Nadizar[1][0000−0002−3535−9748], Eric Medvet[2][0000−0001−5652−2113], and Dennis G. Wilson[3][0000−0003−2414−0051]

[1] Department of Mathematics and Geosciences, University of Trieste, Italy
[2] Department of Engineering and Architecture, University of Trieste, Italy
[3] ISAE-SUPAERO, University of Toulouse, France

Abstract. In most high-risk applications, interpretability is crucial for ensuring system safety and trust. However, existing research often relies on hard-to-understand, highly parameterized models, such as neural networks. In this paper, we focus on the problem of policy search in continuous observations and actions spaces. We leverage two graph-based Genetic Programming (GP) techniques—Cartesian Genetic Programming (CGP) and Linear Genetic Programming (LGP)—to develop effective yet interpretable control policies. Our experimental evaluation on eight continuous robotic control benchmarks shows competitive results compared to state-of-the-art Reinforcement Learning (RL) algorithms. Moreover, we find that graph-based GP tends towards small, interpretable graphs even when competitive with RL. By examining these graphs, we are able to explain the discovered policies, paving the way for trustworthy AI in the domain of continuous control.

Keywords: Graph-based Genetic Programming · Cartesian Genetic Programming · Linear Genetic Programming · Interpretable Policy · Continuous Control

1 Introduction

In the last decade, Artificial Neural Networks (ANNs) have achieved outstanding results in a myriad of domains, surpassing human performance even in scenarios where fast decision making (e.g., drone control [20]) or long-term planning (e.g., strategy games [41]) are needed. This success comes in conjunction with the multiplication of available optimization techniques for ANNs [40], which have become sample-efficient, oftentimes robust, and even transferable to the real world [38].

However, ANNs carry along a natural downside: their large structures, with as many as billions of parameters, remain often completely unintelligible for human observers. In fact, although several post-hoc explanation methods have been recently put forward [1, 15, 35], ANN-based controllers remain inherently black-box models, the true functioning of which can only be surmised through statistical inference.

In opposition to this paradigm, there has been renewed interest in the development of *interpretable models*, i.e., models which can be "directly" understood, without the need of additional proxies or post-hoc procedures. Even in the absence of a shared definition of interpretability [28, 34, 46], models which comply to some notion of interpretability are considered preferable w.r.t. black-boxes [37], as they allow humans to directly conceive their working rationales. Not surprisingly, interpretability plays a fundamental role principally in high-stakes applications, where transparency is among the key factors for trustworthiness [14].

In particular, the domain of robotics is starting to witness a call for more transparent systems, as most people are reluctant to the adoption of autonomous robots in the wild, especially when their control algorithms remain obscure. More generally, this need affects the broader family of control, where the objective is to find a control law that links inputs and outputs towards the achievement of some goal.

In this work, we aim at discovering efficient yet interpretable control laws, by leveraging functional graphs, which are well-suited for capturing complex mappings between observations and actions while maintaining a fair degree of interpretability when their size is contained. For searching the space of graphs, we rely on two graph-based Genetic Programming (GP) techniques, Cartesian Genetic Programming (CGP) and Linear Genetic Programming (LGP), targeting our optimization at performance, and letting interpretability naturally emerge.

Our experiments, conducted on eight continuous control problems from the Mujoco suite [42], reveal that the obtained policies are often comparable with the state-of-the-art in terms of performance, while being naturally more interpretable, even in the absence of an explicit incentive for interpretability. Moreover, the results achieved via graph-based GP appear more consistent and less sensitive to hyper-parameters tuning than those obtained with Reinforcement Learning (RL).

In the remainder of the paper, we briefly survey some relevant related works in Section 2, before describing our methodology and the needed background notions in Section 3. Last, we detail our experiments and discuss their results in Sections 4 and 5, before drawing conclusions in Section 6.

2 Related works

In recent years, the field of Explainable Reinforcement Learning (XRL) has gained notable traction [35, 47], mostly abiding to the necessity of understanding the rationale underlying decisions made by artificial agents intended for deployment in the wild. Taking a step even further, considerable effort has been put into developing policies that are understandable per se, paving the way towards interpretable RL [13]. In this context, several proposals for interpretable policies have appeared, ranging from symbolic policies to decision trees, together with a wide range of methods for obtaining them [27]. Among them, a popular choice consists in leveraging evolutionary computation techniques in combination with

classical RL approaches [50]. In particular, GP and its derivatives are naturally suited for yielding interpretable policies: from its very origins in the '90s, Koza and Rice [26] proved how GP could be used for searching a policy to control a robot.

Following this seminal work, GP has often been used to solve various control tasks, be it via direct policy search or through imitation learning, where an agent learns by mimicking the actions/behavior of an expert, i.e., of a well-performing policy. For instance, Verma et al. [43] distilled an ANN policy into an interpretable program for The Open Racing Car Simulator (TORCS), obtaining interpretability and robustness at the cost of a slight performance degradation. However, although imitation learning partially addresses the poor sample efficiency of GP [40], it is not always as effective as direct policy search. In particular, Hein et al. [17] have considered a model-based RL setting and have shown that symbolic policies learned via direct interaction with the world model outperform those obtained with symbolic regression to imitate the behavior of a pre-trained ANN. Furthermore, sometimes imitation learning can even yield completely unprofitable policies, especially when the robot/environment interactions are too complex to be captured via imitation only [31].

Instead, many works have used GP for direct policy search, mostly focusing on scenarios with discrete action spaces. Custode and Iacca [6] and Ferigo et al. [9, 10] have evolved decision trees, using GP hybridized with RL and quality-diversity optimization, respectively, achieving interpretable and effective policies on simple control tasks, such as cart pole and mountain car. Video games have often been used as test-beds, given the difficulty of processing high-dimensional visual inputs as the full gamut of pixels on a screen. In this context, most of the efforts have been devoted to solving the Atari benchmark [30], e.g., with co-evolved decision trees [7], or with Tangled Program Graphss (TPGs) [21], even obtaining a single program able to solve multiple games [22, 24]. Closer to our work, Wilson et al. [48] relied on mixed-type CGP for finding simple and well-performing policies for playing the Atari games. Interestingly, as in our case, the simplicity of the policies was not promoted explicitly during evolution, but emerged naturally thanks to the representation employed.

Moving even closer to our study, fewer works have succeeded in solving problems in continuous and multi-dimensional action spaces, as these problems require finding and exploiting interdependencies between outputs to achieve coordination. For instance, Medvet and Nadizar [31] have experimented with an ensemble of GP trees for controlling the actuation values of a simulated soft robot, even matching the performance of ANNs. CGP has also been applied to continuous control problems: Wilson et al. [49] proposed a positional variant of CGP where the position of nodes was subject to evolution, and tested it on 9 benchmark problems, including three Mujoco environments. This study is again strongly linked with ours, although we consider the standard CGP variant on more environments, and obtain generally better results. Videau et al. [44] have also employed multi-tree GP for solving a set of Mujoco environments and have also used LGP to allow information sharing among outputs. This work is closely

related to ours, although they consider a bi-objective optimization where they explicitly reward policy simplicity, while we let program simplicity emerge from evolution only. Moreover, we consider a larger set of environments. More recently, LGP has also been successfully used on a complex robotic agent, the Laikago robot, for discovering robust and resilient policies which are also easy to understand, being expressed as transparent computer programs [23]. Last, Amaral et al. [2] have shown the potential of Symbiotic Bid-based GP hybridized with TPGs for a continuous control locomotion task.

GP has been shown as a useful means of discovering interpretable policies on a number of environments. In this work, we hope to further this literature by demonstrating that graph-based GP can naturally find simple, interpretable policies for continuous control on complex simulated robotics tasks.

3 Graph-based Genetic Programming for continuous control

We concentrate on the domain of continuous control, with the aim of finding an *interpretable* yet effective controller, whose actions will steer the system towards the achievement of a certain goal. Namely, we consider those tasks where both the observations and the possible actions are real-valued and multivariate, in a discrete-time simulated environment.

More formally, an environment is a Markov Decision Process (MDP) [36], i.e., a tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S} \subseteq \mathbb{R}^n$ and $\mathcal{A} \subseteq \mathbb{R}^m$ are the state and action spaces, respectively, $p : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the function describing the dynamics of the environment, and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. We consider partial observability, thus we introduce an observation space $\mathcal{O} \subseteq \mathbb{R}^q$, with $q \leq n$, and an observation function $\phi : \mathcal{S} \to \mathcal{O}$ mapping states to the corresponding observations. At each simulation time step t, the environment is in a state $\boldsymbol{s}_t \in \mathcal{S}$, the agent observes a subset of the current state, $\phi(\boldsymbol{s}_t) = \boldsymbol{o}_t \in \mathcal{O}$, and takes an action $\boldsymbol{a}_t \in \mathcal{A}$ according to a *policy* $\pi : \mathcal{O} \to \mathcal{A}$, which results in the system changing into state $\boldsymbol{s}_{t+1}$ with probability $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ and the agent receiving a reward $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The objective consists in finding a policy which maximizes the cumulative reward obtainable in a simulated episode of duration T, $R_T(\pi) = \sum_{t=0}^{T} r(\boldsymbol{s}_t, \pi(\boldsymbol{o}_t))$, starting from an initial state $\boldsymbol{s}_0$. While we consider the formalization of an MDP, in this study we use optimization methods based only on the total episode reward $R_T(\pi)$. Unifying the terminology with the previous paragraph, π is the controller we search for, which determines the actions in the environment.

In our case, π is a graph which represents a multi-variate function. This family of policies can be inherently interpretable because each graph results from the high-level composition of simple functions, e.g., $+$, $-$, or sin, and is constrained in size. We rely on two flavours of graph-based GP for searching the space of graphs, namely CGP and LGP. We define the fitness of a policy graph π as the aforementioned cumulative reward, $f(\pi) \coloneqq R_T(\pi)$, which evolution maximizes.

3.1 Graph-based Genetic Programming

Although GP was primarily born as a technique for evolving computer programs, internally represented as trees [25], throughout the years graph-based alternatives have gained notable traction [11]. Here we consider two variants of graph-based GP, CGP [33] and LGP [19], which share a linear integer genotype and a phenotype that can be interpreted as a Directed Acyclic Graph (DAG). We describe them in further details in Sections 3.2 and 3.3, while we devote the remainder of this section to describing the Genetic Algorithm (GA) we leverage for both of them.

The first step of the optimization process consists in the initialization of a population of size n_{pop}, i.e., in the generation of n_{pop} genotypes; each genotype $\boldsymbol{\theta}$ is then mapped into a policy graph $\pi_{\boldsymbol{\theta}}$, and evaluated according to the fitness function $f(\pi_{\boldsymbol{\theta}})$. Then, the GA proceeds by iterating the following steps for n_{gen} generations. First, n_p parents are selected from the population with tournament selection (with size n_{tour}). Then, the collection of offspring, also of size n_p, is generated by applying some genetic operators. Next, the offspring is encoded and evaluated. Last, the offspring is merged with the best $n_{\text{pop}} - n_p$ individuals of the parent population (selected with truncation selection), obtaining the new population, which is used to start the following loop.

The genotype, the encoding procedure, and the genetic operators employed differ between CGP and LGP, as we describe in the following.

3.2 Cartesian Genetic Programming

CGP represents programs as grids of nodes in a Cartesian plane [32, 33]. Each node represents a function, which can use either the outputs of the nodes of the previous layers or the program inputs as arguments. The final outputs of the program are collected from the outputs of n_{out} selected nodes.

Here, without loss of generality, we consider a uni-dimensional grid, i.e., a sequence of nodes of length n_{nodes}. Hence, to fully determine a CGP graph, we need (1) n_{out} indexes to specify the program outputs, and (2) n_{nodes} tuples $(h, i_1, \ldots, i_{m_{\text{ar}}})$, specifying the function and its arguments for each node, where m_{ar} is the maximum arity of the functions available in the function set H, 2 in our case. We remark that for each of the tuples, h is an index bounded by the cardinality of H, whereas each input index can range from 0 to $n_{\text{in}} + j - 1$, where j indicates the current node position. Thus, the genome of a CGP graph is a sequence of bounded integers of size $n_{\text{out}} + (1 + m_{\text{ar}})n_{\text{nodes}}$.

To initialize each genome, we sample a uniform distribution over the allowed integer values for each position. As a genetic operator, we use int-flip mutation, i.e., we sample a new integer value in the allowed interval, with a different probability for node inputs p_i, node functions p_f, and program outputs p_o. While CGP traditionally used a $(1 + \lambda)$ EA for evolution [33], the use of a GA with multiple elites, as we have done here, has shown similar results [32] and allows for greater parallelization and a direct comparison with LGP.

3.3 Linear Genetic Programming

In LGP, programs are lists of instructions from a programming language, where the result of each instruction is assigned to a register from a predefined set [4]. The inputs of the program are copied into the first n_{in} registers, whereas the outputs are taken from the last n_{out} registers. If we consider the information flow in a LGP program, we can interpret it as a DAG, and hence LGP falls under the category of graph-based GP [11].

To describe a LGP program, each instruction is determined by specifying (1) the index of the register to be assigned, (2) the function to execute from those available in H, and (3) the sequence of arguments to be passed to the function $(i_1, \dots, i_{m_{\text{ar}}})$, expressed in terms of register indexes ($m_{\text{ar}} = 2$ being the maximum arity of functions in H). Hence, to fully determine a program of n_{lines} lines, the genome is a sequence of bounded integers of size $(2+m_{\text{ar}})n_{\text{lines}}$. In this case the bounds for register indexes are constituted by the amount of available registers n_{reg}, as even those not explicitly assigned before are initialized to 0 and are available to select.

To generate each genome, we sample a uniform distribution over the allowed integer values for each position. Regarding the genetic operators, we apply one-point crossover (constraining instructions to be atomic) followed by int-flip mutation, with a different probability for the left-hand side of each program line, i.e., the registers to be assigned, p_a, functions p_f, and function inputs p_i. We provide hyperparameters and other experimental details below.

4 Experimental evaluation

In our experimental evaluation we address the following research question: *"can graph-based GP yield effective yet interpretable policies for continuous control tasks?"* To this end, we perform several optimizations for both CGP and LGP on 8 continuous control tasks from the Mujoco suite. To have meaningful effectiveness baselines, we compare the performance of the resulting graph policies with those obtained with two state-of-the-art RL algorithms which represent policies using ANNs. Moreover, we measure the complexity of the graphs obtained by GP, and analyze example policies in detail to appraise their interpretability.

We open-source our code and our experimental results at `https://github.com/giorgia-nadizar/cgpax`.

4.1 Continuous control benchmark tasks

We employ 8 benchmark continuous control tasks from the Mujoco suite [42]. Namely, we consider environments of growing complexity in terms of input and output space dimensionality, as we summarize in Table 1.

Specifically, we consider two balancing tasks, *inverted pendulum* and *inverted double pendulum*, where the controller is rewarded for preventing the pendulum from falling (and also penalized for excessive movements for the inverted double

Environment	Obs. $\mathcal{O}$	Act. $\mathcal{A}$	Environment	Obs. $\mathcal{O}$	Act. $\mathcal{A}$
Inverted pendulum	$\mathbb{R}^4$	$[-3, 3]$	Hopper	$\mathbb{R}^{11}$	$[-1, 1]^3$
Inverted double pendulum	$\mathbb{R}^8$	$[-1, 1]$	Walker2d	$\mathbb{R}^{17}$	$[-1, 1]^6$
Reacher	$\mathbb{R}^{11}$	$[-1, 1]^2$	Half cheetah	$\mathbb{R}^{18}$	$[-1, 1]^6$
Swimmer	$\mathbb{R}^8$	$[-1, 1]^2$	Ant	$\mathbb{R}^{27}$	$[-1, 1]^8$

Table 1: Observation and action space sizes for the considered problems.

pendulum). We also incorporate a target-aiming task, *reacher*, where a robotic arm has to reach an object and is rewarded for getting as close to the object as possible and penalized for excessive movements. Last, we include 5 locomotion tasks, *swimmer*, *hopper*, *walker2d*, *half cheetah*, and *ant*, which all use a positive reward for the distance covered and a penalty term for excessive movements. Among the locomotion problems, hopper, walker2d, and ant can be unstable; they therefore have an additional reward term for maintaining their balance throughout the simulation. For these problems, if the agent falls to the ground, the episode is terminated earlier.

For all tasks, we use the Brax [12] implementation which leverages parallelism on GPU accelerators with JAX [3]. We use the v1 simulation engine of Brax and set all parameters to their default values. We run all episodes to a duration $T = 1000$ time steps.

4.2 Reinforcement Learning baselines

As baselines for comparing the performance of the graph policies, we use two state-of-the-art RL algorithms for the optimization of ANNs, Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC).

PPO [39] is an on-policy RL algorithm that optimizes agent policies within a "trust region", balancing the exploration-exploitation trade-off. It achieves this by optimizing a first-order approximation of the expected reward while ensuring that policy updates remain close to the actual values.

SAC [16] is an off-policy RL algorithm that balances the maximization of two objectives: the expected cumulative reward and the policy entropy. By simultaneously promoting exploration through entropy maximization and exploitation through return maximization, SAC ensures robust learning in complex environments with continuous action spaces.

These two algorithms are considered state-of-the-art for deep RL and are the default algorithms for the Brax library. We use the Brax library implementations for both PPO and SAC.

4.3 Parameter settings

For the GP evolutionary loop we set $n_{\text{pop}} = 100$, $n_p = 90$, and $n_{\text{tour}} = 3$. Concerning the amount of iterations performed n_{gen}, instead, we have a different value for each problem, depending on how difficult the optimization task is. Namely, for inverted pendulum we set $n_{\text{gen}} = 10$, for half-cheetah we set $n_{\text{gen}} =$

500, for inverted double pendulum, reacher, and swimmer we set $n_{\text{gen}} = 1000$, for hopper and walker2d we set $n_{\text{gen}} = 1500$, and for ant we set $n_{\text{gen}} = 2500$. Regarding the representation specific parameters, we set $n_{\text{nodes}} = 50$, $p_i = p_f = 0.1$, and $p_o = 0.3$ for CGP, while we set $n_{\text{reg}} = n_{\text{in}} + n_{\text{out}} + 5$, $n_{\text{lines}} = 15$, $p_a = 0.3$, and $p_f = p_i = 0.1$ for LGP.

For both CGP and CGP we consider the following function set $H = \{\bullet + \bullet, \bullet - \bullet, \bullet \times \bullet, \bullet \div^* \bullet, |\bullet|, \exp \bullet, \sin \bullet, \cos \bullet, \log^* \bullet, \bullet < \bullet, \bullet > \bullet\}$, where $\bullet$ represents an operand, and operators marked with * are protected. The last two functions are Boolean functions, where the output is 1 if the condition is satisfied and 0 otherwise. Moreover, we add two constant inputs to each observation, namely $\{0.1, 1\}$, thus increasing all observation spaces dimensionalities displayed in Table 1 by 2.

For the RL algorithms, we optimize an ANN with 4 layers of size 32 with SiLU activation for PPO, and an ANN of 2 layers of size 256 with ReLU activation for SAC, following the default provided in Brax. Concerning the algorithm specific hyper-parameters we apply the default ones from the Brax implementations of both PPO and SAC with a few minor variations (we report the full parameter list at `https://github.com/giorgia-nadizar/cgpax/blob/main/rl_run.py`).

In all cases, we repeat the optimization for 10 independent times to ensure results consistency. Furthermore, for the GP techniques we evaluate each individual on 5 distinct episodes and consider the median reward across these episodes as fitness, in order to make the evolutionary search more stable. For the same reason, we also re-evaluate the elite individuals upon merging them into the offspring population.

Last, when analyzing results in a comparative manner we perform a Mann-Whitney U test between pairs of distributions, considering equivalence between the two as null hypothesis. When performing multiple pairwise comparisons, we apply the Bonferroni correction, thus dividing the significance level $\alpha = 0.05$ by the number of pairwise comparisons computed.

5 Experimental results and discussion

5.1 Performance results

We report the results in terms of performance in Figure 1. We show the distribution across 10 runs of the cumulative reward $R_T(\pi^\star)$ obtained by the best policy $\pi^\star$ discovered at the end of the optimization on episodes of $T = 1000$ time steps. We divide our data by environment and by optimization technique to ease the visual comparison of results. We also show the p-values resulting from the pairwise Mann-Whitney U tests performed in Table 2, again dividing data by environment.

The first observation we can make regards the comparison between the two employed graph-based GP techniques: in all the considered environments they achieve similar results, with no statistical difference ever observed among the pairs of distributions. We also note that the results are usually consistent for

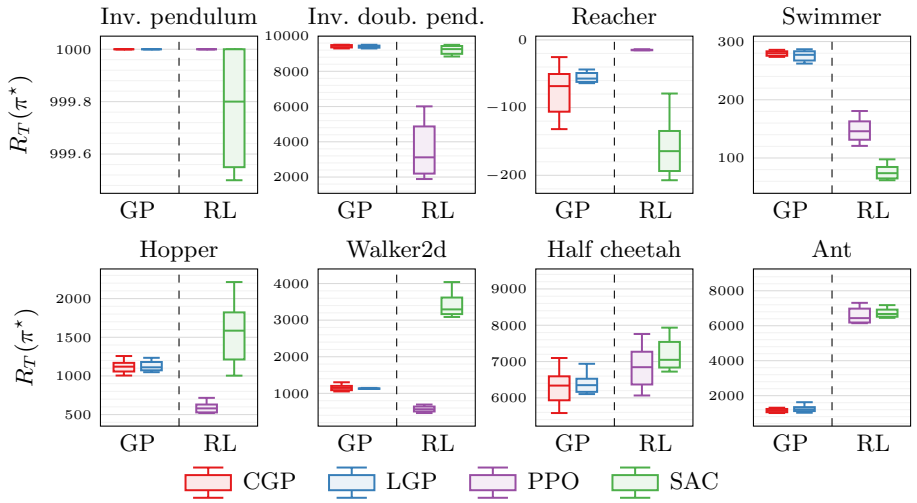


Fig. 1: Box plots of the cumulative reward $R_T(\pi^\star)$ collected by the best policy discovered at the end of optimization $\pi^\star$ in an episode of duration $T = 1000$ time steps.

both CGP and LGP, as opposed to the two glsrl techniques, where the distributions appear generally more spread and with greater differences between the two considered algorithms, PPO and SAC. This suggests that graph GP is more consistent than RL, which is a desirable properly for achieving robust results. We speculate this partially descends from the high sensitivity of these RL algorithms to the chosen hyper-parameters.

For a more detailed comparative analysis, we return to the first part of our research question, concerning the effectiveness of the graph policies found. In the scope of this work, we consider a policy to be effective if it is not significantly worse than the examined RL baselines. Given the reduction in the number of parameters used and the subsequent interpretability gain, we consider policies which achieve similar, but not necessarily better, results to state-of-the-art policies as effective. Among the studied environments, graph policies are not significantly worse than either RL policy in 50 % of cases (inverted pendulum, inverted double pendulum, swimmer, and hopper), while they are not worse than at least one of the RL ones in three more cases (reacher, walker2d, and half cheetah). In fact, the only environment where graph policies are not able to match neither of the RL algorithms is the ant, likely because of the higher dimensionality of both action and observation spaces, and of the known difficulty of graph GP of finding good mappings between high-dimensional spaces. Thus, we can give a generally affirmative answer regarding the effectiveness of both CGP and LGP policies.

Interestingly, in some cases, graph GP is even able to significantly outperform RL. Namely, in four environments (inverted double pendulum, reacher, hopper, and walker2d) both CGP and LGP significantly surpass one of the two RL

	PPO	SAC
CGP	0.168	0.078
LGP	0.168	0.078

(a) Inv. pendulum

	PPO	SAC
CGP	**0.005**	0.571
LGP	**0.006**	0.623

(b) Inv. doub. pend.

	PPO	SAC
CGP	$\approx$**0**	**0.003**
LGP	$\approx$**0**	**0.003**

(c) Reacher

	PPO	SAC
CGP	$\approx$**0**	$\approx$**0**
LGP	$\approx$**0**	$\approx$**0**

(d) Swimmer

	PPO	SAC
CGP	$\approx$**0**	0.021
LGP	$\approx$**0**	0.017

(e) Hopper

	PPO	SAC
CGP	$\approx$**0**	$\approx$**0**
LGP	$\approx$**0**	$\approx$**0**

(f) Walker2d

	PPO	SAC
CGP	0.064	**0.003**
LGP	0.031	**0.001**

(g) Half cheetah

	PPO	SAC
CGP	$\approx$**0**	$\approx$**0**
LGP	$\approx$**0**	$\approx$**0**

(h) Ant

Table 2: p-values resulting from pairwise Mann-Whitney U tests comparing, for each environment, the algorithm on the row with that on the column with the null hypothesis of equivalence of the distributions. We write ≈ 0 for all cells with $p < 0.001$. We consider a significance level of $\alpha = 0.05/5 = 0.01$ (and highlight all significant values in bold) according to the Bonferroni correction, as we performed 5 pairwise comparisons per environment: the ones reported in the table plus the comparison between CGP and LGP (which never yielded statistically significant differences).

algorithms, whereas on swimmer both GP methods neatly outperform both RL algorithms. Arguably, some of these outcomes might come from sub-optimal parameter tuning for the poor-performing RL algorithm, which could potentially yield better results with more tweaking. However, this further corroborates our previous point concerning the robustness of GP results w.r.t. hyper-parameter tuning, unlike RL which is highly sensitive to changes in hyperparameters [18].

To gain deeper insight on the evolutionary process underlying the GP policy search, we also display the progression of the fitness of the best individual in the population, i.e., the cumulative reward $R_T(\pi^\star)$, at each iteration in Figure 2. For all the reported plots, we note that evolution has achieved a plateau, where additional generations would likely result in null or negligible fitness improvements. Moreover, in most cases we can distinguish an initial phase where the fitness grows steeply, followed by a longer phase with minor improvements. This highlights how evolution tends to converge rather quickly—a trait that has both positive and negative implications. Namely, for the environments where the results are satisfactory (i.e., comparable to the state-of-the-art) this implies a reduction in computational costs. However, for the tasks where GP has not matched RL, it indicates that evolution has stagnated in a hard-to-escape local optimum, which clearly hinders the overall performance.

Reasoning further in this direction, we examine some videos of the behaviors of the graph policies and study the reward model of hopper, walker2d, and ant, to investigate the reasons underlying premature convergence. From the visual analysis, we see that the graph policies do make the agents move, yet slowly and very cautiously. This is coherent with the reward model of the environment, where (1) agents are rewarded for maintaining their stability, and (2) the episode

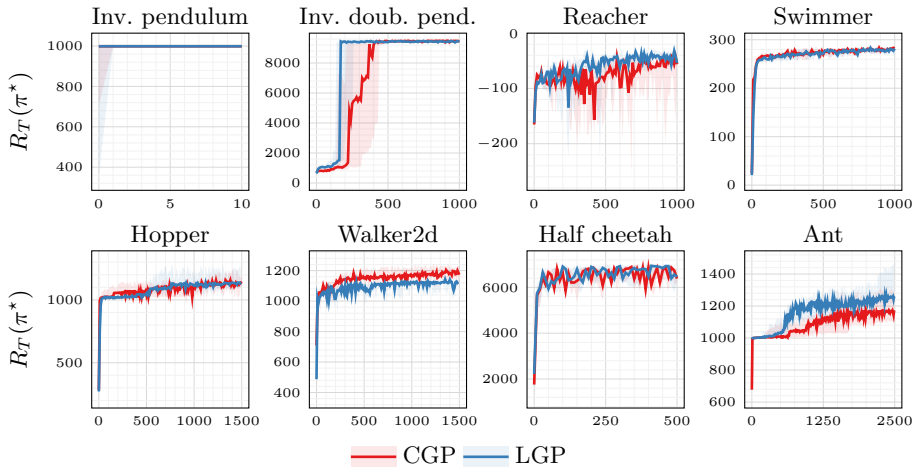


Fig. 2: Progression of the cumulative reward $R_T(\pi^\star)$ collected by the best policy in the population $\pi^\star$ at each generation in an episode of duration $T = 1000$ time steps; median and inter-quartile range across 10 independent runs.

is terminated early if an agent falls (thus preventing the collection of any additional rewards). We speculate that such a model creates a large local optimum for graph GP, where innovative policies which would require some fine tuning to become stable get swept away by evolutionary pressure.

Along this hypothesis, we experimented disabling the early termination of the episode upon the agent fall. However, the results did not show notable improvements, hinting that future work is needed to examine the phenomenon more carefully. Yet, interestingly, we found some physically unrealistic policies for the hopper, which would collect a reward as high as 30 000 by making the agent fly (video[4]). Clearly, these policies are exploiting some instability within the Brax simulator [12], though being white-boxes they could be thoroughly studied to addressing the discovered shortcoming.

5.2 Interpretability of resulting policies

To evaluate the interpretability of a graph policy π, we consider $\rho_c(\pi)$, which we define as the fraction of complexity employed w.r.t. the available one. In practical terms, for CGP, ρ_c is the fraction between the amount of nodes in the policy graph and the maximum possible nodes (50 in our case), whereas for LGP, ρ_c derives from the amount of program lines that contribute to the output computation divided by the total lines of the program (15 in our case).

Clearly, this is a *proxy* for interpretability, as (1) there is no clear shared definition of interpretability, and (2) interpretability can be a highly subjective notion [46]. Therefore, to ensure the general validity of our results, we repeated

[4] `https://giorgia-nadizar.github.io/cgpax//hopper_cgp_flying`

the same analysis with two other interpretability measures: namely, the number of edges composing the policy graph, and the formula ϕ found in [45]. For applying the latter, which derives from a user study targeted at evaluating decomposability and simulatability of mathematical formulae, we first extracted the equations mapping the inputs to the outputs in the graph, and then computed the median value of ϕ across these equations. Since the results obtained with all the considered metrics were consistent, we only display them in terms of ρ_c, for reasons of space.

In Figure 3 we report the progression of $\rho_c(\pi^\star)$ for the best policy graph in the population $\pi^\star$ in terms of median and inter-quartile range across the 10 runs, diving our data by environment and GP technique. From the plots we can distinctly note that in all scenarios the obtained policies rely only on a relatively small fraction of the available complexity, in most cases keeping a ρ_c smaller than one third for the entirety of evolution. Moreover, we can highlight that there is no overall sharp growing trend for ρ_c along generations: although some plots display a slight upward tendency, most of them stay approximately constant or even decrease with generations. This is particularly interesting to observe in comparison with Figure 2. In fact, we discover that a sharp increase in fitness does not imply the growth of the policies, i.e., performance does not come at the cost of interpretability.

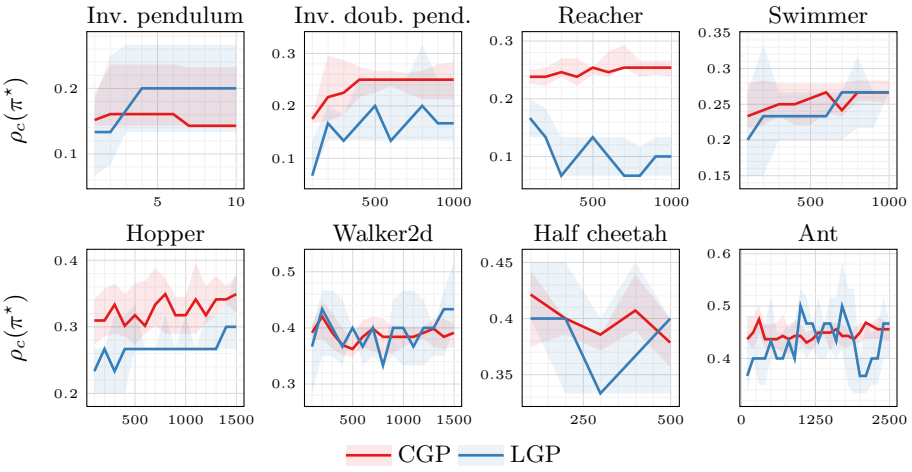


Fig. 3: Fraction of complexity employed $\rho_c(\pi^\star)$ for the best performing policy graph in the population $\pi^\star$; median and inter-quartile range across 10 independent runs.

Notably, these results naturally emerge from the representations of CGP and LGP [11, 32], without any form of explicit interpretability promotion, which leads us to confirm the natural bias of these graph GP techniques towards small and interpretable graphs.

For completeness, we also repeated the optimization in a bi-objective setting—aiming at both effectiveness and interpretability—as in [44], employing NSGA-II [8] in place of the GA described in Section 3.1. As before, we performed 10 runs for each scenario, although we only considered two environments: inverted double pendulum and swimmer.

We present such results in Figure 4, where we plot the cumulative reward $R_T(\pi^\star)$ vs. the fraction of complexity $\rho_c(\pi^\star)$ for all the policies $\pi^\star$ in each of the final Pareto fronts. For context and reference, we also include the policies obtained with the standard GA. From this figure we notice that NSGA-II is able to discover simpler policies than the standard GA, though it oftentimes fails to find slightly larger but more effective ones, due to its bias towards the satisfaction of its easiest objective [29]. Thus, we can conclude that for these GP techniques, where interpretability tends to naturally emerge, it is advisable to maintain a single objective in the search.

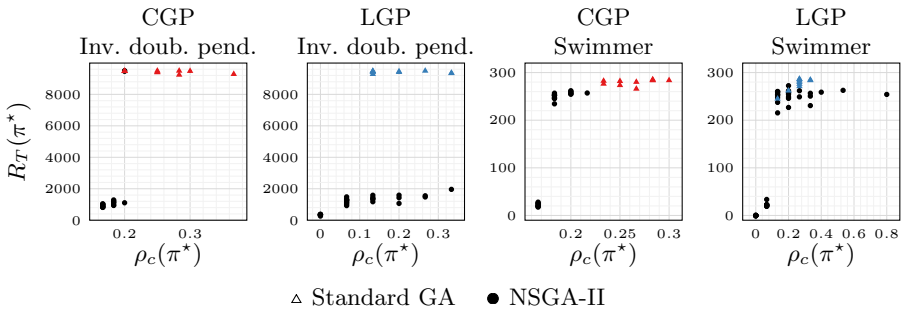
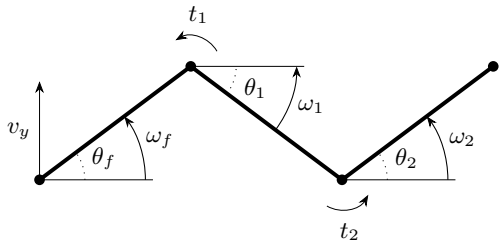


Fig. 4: Cumulative reward $R_T(\pi^\star)$ vs. fraction of complexity $\rho_c(\pi^\star)$ for each policy $\pi^\star$ in each pareto front at the end of evolution with NSGA-II. Policies found with the standard GA added for reference.

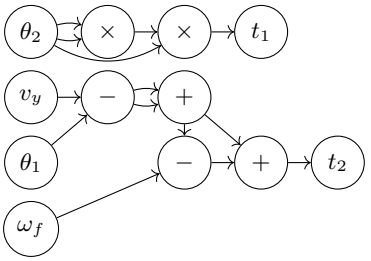
To conclude our study on the interpretability of the obtained policies, we analyze two policies for the swimmer environment. We report a schematic representation of the environment [5], together with the two policies and their conversion in equation form, in Figure 5, and make the corresponding videos available[5].

By looking at both Figures 5b and 5c, we can immediately perceive their simplicity and their compliance to the transparency objective in interpretability. Although in a dynamical system it is hard to achieve full understanding without considering all the equations governing it, as, e.g., those from the simulator, we can still gain some intuitions from the reported formulae. In particular, we can notice a dependency of each torque control value from the state of the opposite side of the agent, either in terms of angles or in terms of angular or linear velocities. This trait, which could not be easily detected in a black-box policy, is of paramount importance for achieving coordination of movement, and is likely a key factor for accomplishing outstanding results at the locomotion task.

[5] `https://giorgia-nadizar.github.io/cgpax/swimmer_cgp`
`https://giorgia-nadizar.github.io/cgpax/swimmer_lgp`



(a) Swimmer schematic.




```
def controller(inputs, r):
  # inputs to registers
  r[0:10] = inputs
  # perform computation
  r[13] = r[1] - r[5]
  r[16] = r[16] - r[7]
  r[15] = r[13] - r[6]
  r[15] = r[15] + r[2]
  r[16] = r[16] - r[1]
  # registers to output
  return r[-2:]
```

$$t_1 = \theta_2^3$$
$$t_2 = 4(v_y - \theta_1) - \theta_f$$

(b) CGP solution.

$$t_1 = \theta_1 - \omega_f - \omega_1 + \theta_2$$
$$t_2 = -\omega_2 - \theta_1$$

(c) LGP solution.

Fig. 5: Schematic representation (5a) of the swimmer environment with the observed variables $\theta_f, \theta_1, \theta_2, \omega_f, \omega_1, \omega_2, v_y$ and the control variables t_1, t_2, and the policies found with CGP (5b) and LGP (5c), shown in their original form and as formulae.

6 Concluding remarks

In this work, we leveraged two graph-based Genetic Programming (GP) techniques, Cartesian Genetic Programming (CGP) and Linear Genetic Programming (LGP), to address several continuous control problems, the goal being that of obtaining effective yet interpretable control policies. Experimenting on eight Mujoco environments, we found graph policies that not only did perform at state-of-the-art level on a subset of tasks, but were also simple enough to be observed and directly understood. In fact, we confirmed the natural trend of both CGP and LGP to yield simple graphs, even in the absence of an explicit incentive, and found single objective optimization to be more successful in finding a good trade-off between performance and simplicity, in comparison to bi-objective optimization.

In the future, we plan to tackle some of the observed shortcomings of graph-based GP, as, e.g., premature convergence. To this end, we intend to experiment with quality-diversity optimization, novelty search, or diversity preserving mechanisms, to prevent stagnation and enable the discovery of more effective policies.

Acknowledgements

The paper is based upon work from a scholarship supported by SPECIES (`http://species-society.org`), the Society for the Promotion of Evolutionary Computation in Europe and its Surroundings. This study was carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) - Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS_00000043).

References

[1] Adadi, A., Berrada, M.: Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). IEEE access **6**, 52138–52160 (2018)
[2] Amaral, R., Ianta, A., Bayer, C., Smith, R.J., Heywood, M.I.: Benchmarking genetic programming in a multi-action reinforcement learning locomotion task. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 522–525 (2022)
[3] Bradbury, J., Frostig, R., Hawkins, P., Johnson, M.J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., et al.: Jax: composable transformations of python+ numpy programs (2018)
[4] Brameier, M., Banzhaf, W., Banzhaf, W.: Linear genetic programming, vol. 1. Springer (2007)
[5] Coulom, R.: Reinforcement learning using neural networks, with applications to motor control. Ph.D. thesis, Institut National Polytechnique de Grenoble-INPG (2002)
[6] Custode, L.L., Iacca, G.: Evolutionary learning of interpretable decision trees. arXiv preprint arXiv:2012.07723 (2020)
[7] Custode, L.L., Iacca, G.: Interpretable pipelines with evolutionary optimized modules for reinforcement learning tasks with visual inputs. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 224–227 (2022)
[8] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE transactions on evolutionary computation **6**(2), 182–197 (2002)
[9] Ferigo, A., Custode, L.L., Iacca, G.: Quality Diversity Evolutionary Learning of Decision Trees. arXiv preprint arXiv:2208.12758 (2022)
[10] Ferigo, A., Custode, L.L., Iacca, G.: Quality–diversity optimization of decision trees for interpretable reinforcement learning. Neural Computing and Applications pp. 1–12 (2023)
[11] Françoso Dal Piccol Sotto, L., Kaufmann, P., Atkinson, T., Kalkreuth, R., Porto Basgalupp, M.: Graph representations in genetic programming. Genetic Programming and Evolvable Machines **22**(4), 607–636 (2021)
[12] Freeman, C.D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., Bachem, O.: Brax–a differentiable physics engine for large scale rigid body simulation. arXiv preprint arXiv:2106.13281 (2021)

[13] Glanois, C., Weng, P., Zimmer, M., Li, D., Yang, T., Hao, J., Liu, W.: A survey on interpretable reinforcement learning. arXiv preprint arXiv:2112.13112 (2021)
[14] Glass, A., McGuinness, D.L., Wolverton, M.: Toward establishing trust in adaptive agents. In: Proceedings of the 13th international conference on Intelligent user interfaces, pp. 227–236 (2008)
[15] Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM computing surveys (CSUR) **51**(5), 1–42 (2018)
[16] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning, pp. 1861–1870, PMLR (2018)
[17] Hein, D., Udluft, S., Runkler, T.A.: Interpretable policies for reinforcement learning by genetic programming. Engineering Applications of Artificial Intelligence **76**, 158–169 (2018)
[18] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep reinforcement learning that matters. In: Proceedings of the AAAI conference on artificial intelligence, vol. 32 (2018)
[19] Kantschik, W., Banzhaf, W.: Linear-graph gp-a new gp structure. In: European Conference on Genetic Programming, pp. 83–92, Springer (2002)
[20] Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V., Scaramuzza, D.: Champion-level drone racing using deep reinforcement learning. Nature **620**(7976), 982–987 (2023)
[21] Kelly, S., Heywood, M.I.: Emergent tangled graph representations for atari game playing agents. In: Genetic Programming: 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings 20, pp. 64–79, Springer (2017)
[22] Kelly, S., Heywood, M.I.: Multi-task learning in atari video games with emergent tangled program graphs. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 195–202 (2017)
[23] Kelly, S., Park, D.S., Song, X., McIntire, M., Nashikkar, P., Guha, R., Banzhaf, W., Deb, K., Boddeti, V.N., Tan, J., et al.: Discovering adaptable symbolic algorithms from scratch. arXiv preprint arXiv:2307.16890 (2023)
[24] Kelly, S., Voegerl, T., Banzhaf, W., Gondro, C.: Evolving hierarchical memory-prediction machines in multi-task reinforcement learning. Genetic Programming and Evolvable Machines **22**, 573–605 (2021)
[25] Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Statistics and computing **4**(2), 87–112 (1994)
[26] Koza, J.R., Rice, J.P.: Automatic programming of robots using genetic programming. In: AAAI, vol. 92, pp. 194–207 (1992)
[27] 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: International Conference on Machine Learning, pp. 5979–5989, PMLR (2021)
[28] Lipton, Z.C.: The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. Queue **16**(3), 31–57 (2018)

[29] Liu, D., Virgolin, M., Alderliesten, T., Bosman, P.A.: Evolvability degeneration in multi-objective genetic programming for symbolic regression. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 973–981 (2022)

[30] Machado, M.C., Bellemare, M.G., Talvitie, E., Veness, J., Hausknecht, M., Bowling, M.: Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. Journal of Artificial Intelligence Research **61**, 523–562 (2018)

[31] Medvet, E., Nadizar, G.: GP for Continuous Control: Teacher or Learner? The Case of Simulated Modular Soft Robots. Genetic Programming Theory and Practice XX (2023)

[32] Miller, J.F.: Cartesian genetic programming: its status and future. Genetic Programming and Evolvable Machines **21**, 129–168 (2020)

[33] Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Genetic Programming, pp. 121–132, Springer Berlin Heidelberg, Berlin, Heidelberg (2000), ISBN 978-3-540-46239-2

[34] Nadizar, G., Rovito, L., De Lorenzo, A., Medvet, E., Virgolin, M.: An analysis of the ingredients for learning interpretable symbolic regression models with human-in-the-loop and genetic programming. ACM Transactions on Evolutionary Learning (2024)

[35] Puiutta, E., Veith, E.M.: Explainable reinforcement learning: A survey. In: International cross-domain conference for machine learning and knowledge extraction, pp. 77–95, Springer (2020)

[36] Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)

[37] Rudin, C.: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature machine intelligence **1**(5), 206–215 (2019)

[38] Salvato, E., Fenu, G., Medvet, E., Pellegrino, F.A.: Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. IEEE Access **9**, 153171–153187 (2021)

[39] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

[40] Sigaud, O., Stulp, F.: Policy search in continuous action domains: an overview. Neural Networks **113**, 28–40 (2019)

[41] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484–489 (2016)

[42] Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ international conference on intelligent robots and systems, pp. 5026–5033, IEEE (2012)

[43] Verma, A., Murali, V., Singh, R., Kohli, P., Chaudhuri, S.: Programmatically interpretable reinforcement learning. In: International Conference on Machine Learning, pp. 5045–5054, PMLR (2018)

[44] Videau, M., Leite, A., Teytaud, O., Schoenauer, M.: Multi-objective genetic programming for explainable reinforcement learning. In: European Conference on Genetic Programming (Part of EvoStar), pp. 278–293, Springer (2022)
[45] Virgolin, M., De Lorenzo, A., Medvet, E., Randone, F.: Learning a formula of interpretability to learn interpretable formulas. In: Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part II 16, pp. 79–93, Springer (2020)
[46] Virgolin, M., De Lorenzo, A., Randone, F., Medvet, E., Wahde, M.: Model learning with personalized interpretability estimation (ml-pie). In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1355–1364 (2021)
[47] Wells, L., Bednarz, T.: Explainable ai and reinforcement learning—a systematic review of current approaches and trends. Frontiers in artificial intelligence **4**, 550030 (2021)
[48] Wilson, D.G., Cussat-Blanc, S., Luga, H., Miller, J.F.: Evolving simple programs for playing atari games. In: Proceedings of the genetic and evolutionary computation conference, pp. 229–236 (2018)
[49] Wilson, D.G., Miller, J.F., Cussat-Blanc, S., Luga, H.: Positional cartesian genetic programming. arXiv preprint arXiv:1810.04119 (2018)
[50] Zhou, R., Hu, T.: Evolutionary approaches to explainable machine learning. arXiv preprint arXiv:2306.14786 (2023)