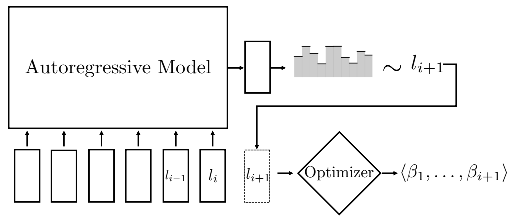
# Generative Design of Decision Tree Policies for Reinforcement Learning

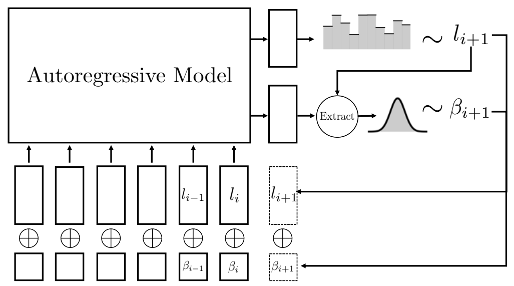
**Jacob F. Pettit** [1] **Chak Shing Lee** [1] **Jiachen Yang** [1 2] **Alex Ho** [1 3] **Daniel Faissol** [1] **Brenden Petersen** [1 4] **Mikel Landajuela** [1]

## Abstract

Decision trees are an attractive choice for modeling policies in control environments due to their interpretability, conciseness, and ease of implementation. However, generating performant decision trees in this context has several challenges, including the hybrid discrete-continuous nature of the search space, the variable-length nature of the trees, the existence of parent-dependent constraints, and the high computational cost of evaluating the objective function in reinforcement learning settings. Traditional methods, such as Mixed Integer Programming or Mixed Bayesian Optimization, are unsuitable for these problems due to the variable-length constrained search space and the high number of objective function evaluations required. To address these challenges, we propose to extend approaches in the field of neural combinatorial optimization to handle the hybrid discrete-continuous optimization problem of generating decision trees. Our approach demonstrates significant improvements in performance and sample efficiency over the state-of-the-art methods for interpretable reinforcement learning with decision trees.



(a) Standard decoupled approach



(b) DisCo-DSO

Figure 1: Comparison of the standard decoupled approach and DisCo-DSO for discrete-continuous optimization using an autoregressive model. In the decoupled approach, the discrete skeleton $\tau_{\rm d} = \langle (l_1, \cdot), \ldots, (l_T, \cdot) \rangle$ is sampled first and then the continuous parameters $\beta_1, \ldots, \beta_T$ are optimized independently. In contrast, DisCo-DSO models the joint distribution over the sequence of tokens $\langle (l_1, \beta_1), \ldots, (l_T, \beta_T) \rangle$. Here, the notation $\oplus$ stands for concatenation of vectors.

## 1. Introduction

Deep learning methods have demonstrated success on important combinatorial optimization problems (Bello et al., 2016), including generating interpretable policies for continuous control (Landajuela et al., 2021a) and symbolic regression (SR) for discovering underlying mathematical equations from data (Petersen et al., 2021a; Biggio et al., 2021; Kamienny et al., 2022). Existing approaches train a generative model that constructs a solution to the optimization problem by sequentially choosing from a set of discrete tokens, using the objective function value as the terminal reward for learning. However, these approaches do not jointly optimize the discrete and continuous components of such hybrid problems: certain discrete tokens require the additional specification of an associated real-valued parameter, such as the threshold value at a decision tree node, but the learned generative model does not produce these values. Instead, they adopt the design choice of *decoupled optimization*, whereby only the construction of a discrete solution skeleton is optimized by deep learning, while the associated continuous parameters are left to a separate black-box optimizer.

[1]Lawrence Livermore National Laboratory, Livermore, CA, USA [2]Simular*(contributions while at LLNL)* [3]University of California, Merced, CA, USA *(contributions while at LLNL)* [4]Lam Research, Livermore, CA, USA *(contributions while at LLNL)*. Correspondence to: Mikel Landajuela <landajuelala1@llnl.gov>.

*Accepted by the Structured Probabilistic Inference & Generative Modeling workshop* of ICML 2024, Vienna, Austria. Copyright 2024 by the author(s).

We hypothesize that a joint discrete-continuous optimization approach (Figure 1(b)) that generates a complete solution based on deep reinforcement learning (RL) (Sutton & Barto, 2018) has significant advantages compared to existing decoupled approaches that employ learning only for the discrete skeleton (Figure 1(a)). In terms of efficiency, a joint approach only requires one evaluation of the objective function for each solution candidate, whereas the decoupled approach based on common nonlinear black-box optimization methods such as BFGS (Fletcher, 2000), simulated annealing (Xiang et al., 1997), or evolutionary algorithms (Storn & Price, 1997) requires a significant number of function evaluations to optimize each discrete skeleton. Furthermore, joint exploration and learning on the full discrete-continuous solution space has the potential to escape from local optima and use information from prior samples to guide the subsequent search.

We explore the benefits of generative joint discrete-continuous optimization in the context of decision tree policy search in RL. Decision trees are a popular choice for modeling policies in control environments due to their interpretability, conciseness, and ease of implementation (Ding et al., 2020; Silva et al., 2020; Custode & Iacca, 2023). In Custode & Iacca (2023), the authors use an evolutionary search to find the best decision tree policy and further optimized the real valued thresholds using a decoupled approach. Relaxation approaches, which relax the discrete part of the problem into a continuous one, have been proposed in works such as Sahoo et al. (2018); Silva et al. (2020); Ding et al. (2020). These approaches suffer from approximations errors and might lead to suboptimal solutions. The problem of decision tree (decision tree) policy search in RL is computationally expensive, as each objective function evaluation involves running the candidate solution on many episodes of a high-dimensional and stochastic physical simulation (Landajuela et al., 2021a), and many evaluations are required to optimize the discrete-continuous solution space. In particular, Bayesian optimization (BO) methods are unsuitable for this problem as they have a computational complexity of $\mathcal{O}(n^3)$ (Shahriari et al., 2015), where $n$ is the number of function evaluations (Lan et al., 2022).

We summarize the main contributions of this paper as follows:

- We propose a novel method for joint discrete-continuous optimization using deep reinforcement learning that is suited for black-box hybrid optimization problems over variable-length search spaces with prefix-dependent positional constraints (Section 2.2). We call the method Discrete-Continuous Deep Symbolic Optimization (DisCo-DSO).
- We present a novel formulation for decision tree policy search in RL as sequential discrete-continuous optimization and propose a method for sequentially finding bounds for parameter ranges in decision nodes (Section 2.2).
- We perform exhaustive empirical evaluation of DisCo-DSO for decision tree policy search in RL. We show that DisCo-DSO outperforms decoupled approaches on all tasks (Section 3).

## 2. Generative Design of Decision Tree Policies

### 2.1. Notation and problem definition

We consider the problem of discovering decision tree policies for RL. The search space $\mathbb{T}$ is the space of univariate decision trees (Silva et al., 2020). Extensions to multivariate decision trees, also known as oblique trees, are possible, but we leave them for future work. Given an RL environment with observations $x_1, \dots, x_n$ and discrete actions $a_1, \dots, a_m$, we consider the library of Boolean expressions and actions given by $\mathcal{L} = \{x_1 < \beta_1, \dots, x_n < \beta_n, a_1, \dots, a_m\}$, where $\beta_1, \dots, \beta_n$ are the values of the observations that are used in the internal nodes of the decision tree. In the following, we use the notation $l$ and $\beta$ to represent the discrete and continuous parts of the tokens, respectively. For a boolean expression $x_i < \beta_i$, $l$ is the token $x_i < \square$ and $\beta$ is the value $\beta_i$. For an action $a_i$, $l$ is the token $a_i$ and $\beta$ is not used. We represent each decision tree as a sequence $\tau = \langle (l_1, \beta_1), \dots, (l_T, \beta_T) \rangle = \langle \tau_1, \dots, \tau_T \rangle \in \mathcal{T}$, where each token $\tau_i$ belongs to a library $\mathcal{L}$ and the length $T$ of the sequence is not fixed *a priori*.

Given a sequence $\tau$, we define the *discrete skeleton* $\tau_\mathrm{d}$ as the sequence obtained by removing the continuous parameters from $\tau$, i.e., $\tau_\mathrm{d} = \langle (l_1, \cdot), \dots, (l_T, \cdot) \rangle$. We introduce the operator `eval` $: \mathcal{T} \to \mathbb{T}$ to represent the semantic interpretation of the sequence $\tau$ as a decision tree in the design space $\mathbb{T}$.

The evaluation operator `eval` $: \mathcal{T} \to \mathbb{T}$ is defined as follows. We treat sequence $\tau$ as the pre-order traversal of a decision tree, where the decision tokens ($x_n < \beta_n$) are treated as binary nodes and the action tokens ($a_n$) are treated as leaf nodes. For evaluating the decision tree, we start from the root node and follow direction

$$D_{x_n < \beta_n}(x) = \begin{cases} \text{left if } x_n < \beta_n \text{ is True,} \\ \text{right if } x_n < \beta_n \text{ is False,} \end{cases}$$

for every decision node encountered until we reach a leaf node. See Figure 2 for an example.

The optimization problem is defined by the reward function $R : \mathbb{T} \to \mathbb{R}$, defined as $R(t) = \mathbb{E}_{r \sim p_R(r|t)}[r]$ where $p_R(r|t)$ is the reward distribution following decision tree policy $t$ in the environment. In practice, we use the average reward over $N$ episodes, i.e., $R(t) = \frac{1}{N} \sum_{i=1}^{N} r_i$, where $r_i$ is the reward obtained in episode $i$. The optimization problem is to find a sequence $\tau^* = \langle \tau_1^*, \ldots, \tau_T^* \rangle = \langle (l_1^*, \beta_1^*), \ldots, (l_T^*, \beta_T^*) \rangle$ (where the length $T$ is not fixed a priori) such that $\tau^* \in \arg\max_{\tau \in \mathcal{T}} R(\tau)$.

The problem of decision tree policy search presents *prefix-dependent positional constraints*, i.e., given a prefix $\tau_{1:(i-1)}$, there exists a possible non-empty set of unfeasible tokens $\mathcal{C}_{\tau_{1:(i-1)}} \subseteq \mathcal{L}$ such that $\texttt{eval}(\tau_{1:(i-1)} \cup \tau_i \cup \tau_{(i+1):T}) \notin \mathbb{T}$ for all $\tau_i \in \mathcal{C}_{\tau_{1:(i-1)}}$ and for all $\tau_j \in \mathcal{L}$ with $i < j \leq T$. For example, threshold values $\beta$ inside a decision node of a tree are constrained by the thresholds of their parent's nodes. See Section 2.2 and Appendix D.1 for more details.

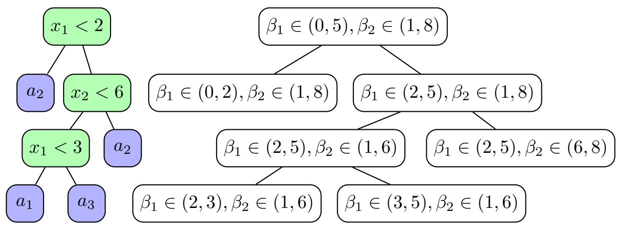

$x_1 < 2$
$a_2$
$x_2 < 6$
$x_1 < 3$
$a_2$
$a_1$
$a_3$
$\beta_1 \in (0, 5), \beta_2 \in (1, 8)$
$\beta_1 \in (0, 2), \beta_2 \in (1, 8)$
$\beta_1 \in (2, 5), \beta_2 \in (1, 8)$
$\beta_1 \in (2, 5), \beta_2 \in (1, 6)$
$\beta_1 \in (2, 5), \beta_2 \in (6, 8)$
$\beta_1 \in (2, 3), \beta_2 \in (1, 6)$
$\beta_1 \in (3, 5), \beta_2 \in (1, 6)$

Figure 2: Left: the decision tree associated with the traversal $\langle x_1 < 2, a_2, x_2 < 6, x_1 < 3, a_1, a_3, a_2 \rangle$. Right: the corresponding bounds for the parameters during the sampling process (suppose the bounds for observations $x_1$ and $x_2$ are respectively [0, 5] and [1, 8]).

## 2.2. Method

**Combinatorial optimization with autoregressive models.** In applications of deep learning to combinatorial optimization (Bello et al., 2016), a probabilistic model $p(\tau)$ is learned over the design space $\mathcal{T}$. The model is trained to gradually allocate more probability mass to high scoring solutions. The training can be done using supervised learning, if problem instances with their corresponding solutions are available, or, more generally, using RL.

In most cases, the model $p(\tau)$ is parameterized by an autoregressive (AR) model with parameters $\theta$. The model is used to generate sequences as follows. At position $i$, the model emits a vector of logits $\psi^{(i)}$ conditioned on the previously generated tokens $\tau_{1:(i-1)}$, i.e., $\psi^{(i)} = \text{AR}(\tau_{1:(i-1)}; \theta)$. The new token $\tau_i$ is sampled from the distribution $p(\tau_i | \tau_{1:(i-1)}, \theta) = \text{softmax}(\psi^{(i)})_{\mathcal{L}(\tau_i)}$, where $\mathcal{L}(\tau_i)$ is the index in $\mathcal{L}$ corresponding to node value $\tau_i$. The new token $\tau_i$ is then added to the sequence $\tau_{1:(i-1)}$ and used to condition the generation of the next token $\tau_{i+1}$. The process continues until a stopping criterion is met.

**Prefix-dependent positional constraints.** Sequential token generation enables flexible configurations and the incorporation of constraints during the search process (Petersen et al., 2021a). Specifically, given a prefix $\tau_{1:(i-1)}$, a prior $\psi_\circ^{(i)} \in \mathbb{R}^{|\mathcal{L}|}$ is computed such that ${\psi_\circ^{(i)}}_{\mathcal{L}(\tau_i)} = -\infty$ for tokens $\tau_i$ in the unfeasible set $\mathcal{C}_{\tau_{1:(i-1)}}$ and zero otherwise. The prior is added to the logits $\psi^{(i)}$ before sampling the token $\tau_i$. See Appendix D for examples.

**Extension to discrete-continuous optimization.** Current deep learning approaches for combinatorial optimization only support discrete tokens, i.e., $\hat{\mathcal{L}} = \emptyset$, (Bello et al., 2016) or completely decouple the discrete and continuous parts of the problem, as in Petersen et al. (2021a); Landajuela et al. (2021a); Mundhenk et al. (2021); da Silva et al. (2023), by sampling first the discrete skeleton $\tau_\text{d}$ and then optimizing its continuous parameters separately (see Figure 1(a)). In this work, we extend these frameworks to support *joint optimization of discrete and continuous tokens*. The model is extended to emit two outputs $\psi^{(i)}$ and $\phi^{(i)}$ for each token $\tau_i = (l_i, \beta_i)$ conditioned on the previously generated tokens, i.e., where we use the notation $(l, \beta)_{1:(i-1)}$ to denote the sequence of tokens $\langle (l_1, \beta_1), \ldots, (l_{i-1}, \beta_{i-1}) \rangle$

$$\left(\psi^{(i)}, \phi^{(i)}\right) = \text{AR}((l, \beta)_{1:(i-1)}; \theta),$$

(see Figure 1(b)). Given tokens $(l, \beta)_{1:(i-1)}$, the $i^{\text{th}}$ token $(l_i, \beta_i)$ is generated by sampling from the following distribution:

$$p((l_i, \beta_i) | (l, \beta)_{1:(i-1)}, \theta) = \begin{cases} \mathcal{U}_{[0,1]}(\beta_i) \text{softmax}(\psi^{(i)})_{\mathcal{L}(l_i)} & \text{if } l_i \in \bar{\mathcal{L}} \\ \mathcal{D}(\beta_i | l_i, \phi^{(i)}) \text{softmax}(\psi^{(i)})_{\mathcal{L}(l_i)} & \text{if } l_i \in \hat{\mathcal{L}} \end{cases},$$

where $\mathcal{D}(\beta_i | l_i, \phi^{(i)})$ is the probability density function of the distribution $\mathcal{D}$ that is used to sample $\beta_i$ from $\phi^{(i)}$. Note that the choice of $\beta_i$ is conditioned on the choice of discrete token $l_i$. We assume that the support of $\mathcal{D}(\beta | l, \phi)$ is a subset of $\mathcal{A}(l)$ for all $l \in \hat{\mathcal{L}}$. Additional priors of the form $(\psi_\circ^{(i)}, 0)$ can be added to the logits before sampling the token $\tau_i$. See Appendix D for more details.

**Training.** The parameters $\theta$ of the model are learned by maximizing the expected reward $J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta)}[R(\tau)]$ or, alternatively, the quantile-conditioned expected reward

$$J_\varepsilon(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta)}[R(\tau) | R(\tau) \geq R_\varepsilon(\theta)],$$

where $R_\varepsilon(\theta)$ represents the $(1 - \varepsilon)$-quantile of the reward distribution $R(\tau)$ sampled from the trajectory distribution $p(\tau|\theta)$. It is worth noting that both objectives, $J(\theta)$ and $J_\varepsilon(\theta)$, serve as *relaxations* of the original $\arg\max R(\tau)$ optimization problem described above. Empirical evidence from Petersen et al. (2021a) demonstrates that, in practice, the $J_\varepsilon(\theta)$ objective tends to be more effective than $J(\theta)$ since it encourages the model to prioritize *best case* performance over *average case* performance.

To optimize the objective $J_\varepsilon(\theta)$, we extend the risk-seeking policy gradient of Petersen et al. (2021a) to the discrete-continuous setting. The gradient of $J_\varepsilon(\theta)$ reads as

$$\nabla_\theta J_\varepsilon(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta)} \left[ A(\tau, \varepsilon, \theta) \, S((l, \beta)_{1:T}) \mid A(\tau, \varepsilon, \theta) > 0 \right],$$

where $A(\tau, \varepsilon, \theta) = R(\tau) - R_\varepsilon(\theta)$ and

$$S((l,\beta)_{1:T}) = \sum_{i=1}^{T} \begin{cases} \nabla_\theta \log p(l_i|(l,\beta)_{1:(i-1)}, \theta) & \text{if } l_i \in \bar{\mathcal{L}}, \\ \nabla_\theta \log p(l_i|(l,\beta)_{1:(i-1)}, \theta) \\ \quad + \nabla_\theta \log p(\beta_i|l_{1:i}, \beta_{1:i-1}, \theta) & \text{if } l_i \in \hat{\mathcal{L}}. \end{cases}$$

We provide pseudocode for DisCo-DSO in Appendix A. See also Appendix B for a derivation of the risk-seeking policy gradient and additional details of the learning procedure.

**Sampling decision nodes in decision trees.** To efficiently sample decision nodes in decision trees, we employ truncated normal distributions to select parameters $\beta_i$ within permissible ranges. Note that this is a feature of the problem we are approaching. Many RL environments place boundaries on observations, and use of the truncated normal distribution guarantees that parameters will only be sampled within those boundaries. Additionally, a decision node which is a child of another decision node cannot naively select parameters from the environment-enforced boundaries. This is because the thresholding performed at a decision node changes the range of values which will be observed at subsequent decision nodes. In this way, a previous decision node ”dictates” the bounds on a current decision node. For instance, consider the decision tree displayed in Figure 2. Assume that the observation $x_1$ falls within the interval $[0, 5]$ (note that in practice the RL environment provided bounds are used to determine the interval), and the tree commences with the node $x_1 < 2$. In the left child node, as $x_1 < 2$ is true, there is no need to evaluate whether $x_1$ is less than 4 (or any number between 2 and 5), as that is already guaranteed. Consequently, we should sample a parameter $\beta_1$ within the range (0, 2). Simultaneously, since we do not assess the Boolean expression regarding $x_2$, the bounds on $\beta_2$ remain consistent with those at the parent node. The parameter bounds for the remaining nodes are illustrated in Figure 2. The procedure for determining these maximum and minimum values is outlined in Algorithm 3 in Appendix D.1.

## 3. Experiments

For evaluation, we follow other works in the field (Silva et al., 2020; Ding et al., 2020; Custode & Iacca, 2023) and use the OpenAI Gym's (Brockman et al., 2016) environments MountainCar-v0, CartPole-v1, Acrobot-v1, and LunarLander-v2. We investigate the sample-efficiency of DisCo-DSO on the decision tree policy task when compared to the following decoupled baselines :

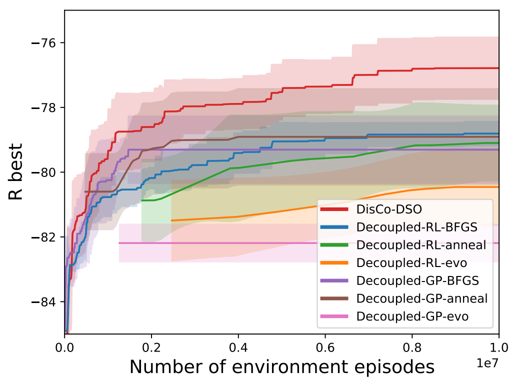
- **Decoupled-RL-**$\{$**BFGS**, **anneal**, **evo**$\}$: This baseline trains a generative model with reinforcement learning to produce a discrete skeleton (Petersen et al., 2021a), which is then optimized by a downstream nonlinear solver for the continuous parameters. The objective value at the optimized solution is the reward, which is used to update the generative model using the same policy gradient approach and architecture as DisCo-DSO. The continuous optimizer is either L-BFGS-B (BFGS), simulated annealing (anneal) (Xiang et al., 1997), or differential evolution (evo) (Storn & Price, 1997), using the SciPy implementation (Virtanen et al., 2020).
- **Decoupled-GP-**$\{$**BFGS**, **anneal**, **evo**$\}$: This baseline uses genetic programming (GP) (Koza, 1990) to produce a discrete skeleton, which is then optimized by a downstream nonlinear solver for the continuous parameters.

All experiments involving RL and DisCo-DSO use a RNN with a single hidden layer of 32 units as the generative model. The GP baselines use the “Distributed Evolutionary Algorithms in Python” software[1] (Fortin et al., 2012). Details on hyperparameters are provided in Appendix C.2.

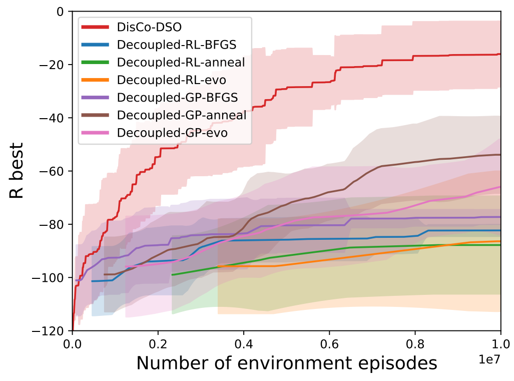
**Results.** In Figure 3, we report the mean and standard deviation of the best reward found by each algorithm versus number of environment episodes. These results show that DisCo-DSO dominates the baselines in terms of sample-efficiency. The trend is consistent across all environments, and is more pronounced in the more complex environments. The efficient use of evaluations by DisCo-DSO (each sample is a complete well-defined decision tree) versus the decoupled approaches, where each sample is a discrete skeleton that requires many evaluations to get a single complete solution, becomes a significant advantage in the RL environments where each evaluation involves running the environment for $N$ episodes.

**Literature comparisons.** We conduct a performance comparison of DisCo-DSO against various literature baselines, namely the evolutionary decision trees as detailed in (Custode & Iacca, 2023), cascading decision trees introduced in (Ding et al., 2020), and interpretable differentiable decision trees (DDTs) introduced in (Silva et al., 2020). In addition, we provide results with a Bayesian Optimization baseline, where the structure of the decision tree is fixed to a binary tree of depth 4 without prefix-dependent positional constraints. Whenever a method supplies a tree structure for a specific environment, we utilize the provided structure and assess it locally. In cases where the method's implementation is missing, we address this by leveraging open-source code. This approach allows us to train a tree on the absent environments, ensuring that we obtain a comprehensive result set for all evaluated methods across all environments. The decision trees found by DisCo-DSO are shown in Figure 4. The comparisons are shown in Table 1. Methods we trained locally are marked with an asterisk (*). Critically, we ensure consistent evaluation across baselines by assess-

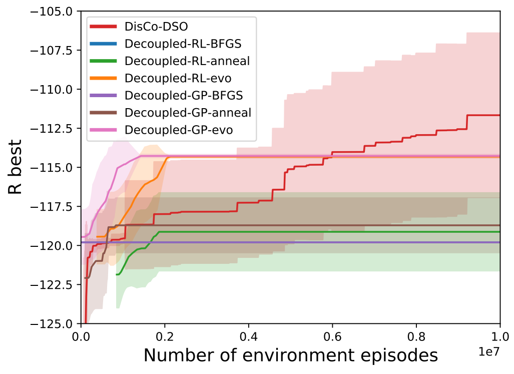
[1]https://github.com/DEAP/deap. LGPL-3.0 license.



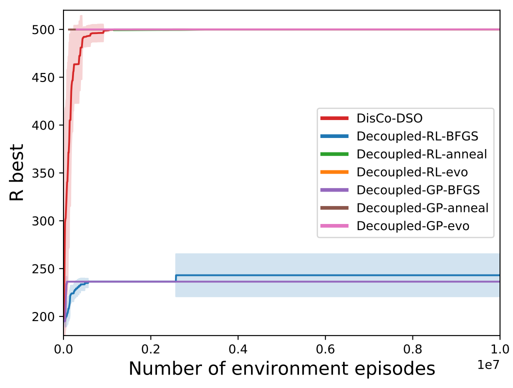
(a) Acrobot-v1



(b) LunarLander-v2



(c) MountainCar-v0



(d) CartPole-v1

Figure 3: Reward of the best solution versus number of function evaluations on the decision tree policy task, for Acrobot-v1, LunarLander-v2, MountainCar-v0, and CartPole-v1. We show the mean and standard deviation of the best reward over 10 random seeds versus the number of environment episodes.

ing each decision tree policy on an identical set of 1,000 random seeds per environment.

In Table 1 we also show the complexity of the discovered decision tree as measured by the number of parameters in the tree. We count every (internal or leaf) node of univariate decision trees (produced by all methods except for Cascading decision trees) as one parameter. For Cascading decision trees, the trees contain feature learning trees and decision making trees. The latter is just univariate decision trees, so the same complexity measurement is used. For the leaf nodes of feature learning trees, the number of parameters is number of observations times number of intermediate features. From Table 1, we observe that the univariate decision trees found by DisCo-DSO have the best performance on all environments at a comparable or lower complexity than the other literature baselines.

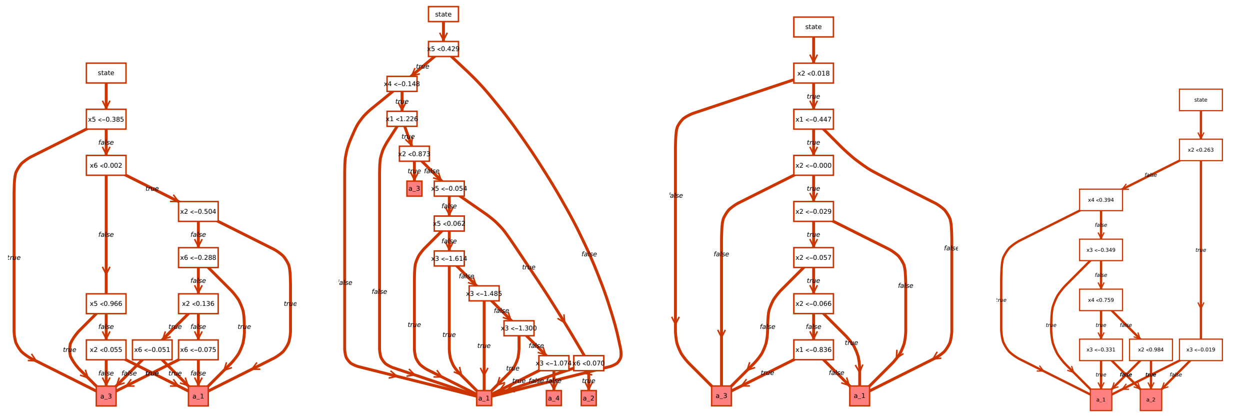
## 4. Conclusion

We explore generative modeling in hybrid discrete-continuous spaces for the problem of decision tree policy search in interpretable RL. We propose DisCo-DSO (Discrete-Continuous Deep Symbolic Optimization), a novel approach for optimization in hybrid discrete-continuous spaces. In contrast to standard decoupled approaches, in which the discrete skeleton is sampled first, and then the continuous variables are optimized separately, our joint optimization approach samples both discrete and continuous variables simultaneously. This leads to more efficient use of objective function evaluations, as the discrete and continuous dimensions of the design space can "communicate" with each other and guide the search. We have demonstrated the benefits of DisCo-DSO on decision tree optimization where DisCo-DSO outperforms the state-of-the-art on univariate decision tree policy optimization for RL.

As for the limitations of DisCo-DSO, it is important to

| Algorithm | Acrobot-v1 | | CartPole-v1 | | LunarLander-v2 | | MountainCar-v0 | |
|---|---|---|---|---|---|---|---|---|
| | MR | PC | MR | PC | MR | PC | MR | PC |
| DisCo-DSO | **-76.58** | 18 | **500.00** | 14 | **99.24** | 23 | **-100.97** | 15 |
| Evolutionary DTs | -97.12* | 5 | 499.58 | 5 | -87.62* | 17 | -104.93 | 13 |
| Cascading DTs | -82.14* | 58 | 496.63 | 22 | -227.02 | 29 | -200.00 | 10 |
| Interpretable DDTs | -497.86* | 15 | 389.79 | 11 | -120.38 | 19 | -172.21* | 15 |
| Bayesian Optimization[†] | -90.99* | 7 | 85.47* | 7 | -112.14* | 7 | -200.0* | 7 |

Table 1: Evaluation of the best univariate decision trees found by DisCo-DSO and other baselines on the decision tree policy task. Here, MR is the mean reward earned in evaluation over a set of 1,000 random seeds, while PC represents the parameter count in each tree. For models trained in-house (*), the figures indicate the parameter count after the discretization process. [†]The topology of the tree is fixed for BO.



(a) Acrobot-v1 (b) LunarLander-v2 (c) MountainCar-v0 (d) CartPole-v1

Figure 4: Best decision trees found by DisCo-DSO on the decision tree policy task for Acrobot-v1, LunarLander-v2, MountainCar-v0, and CartPole-v1.

highlight that the method relies on domain-specific information to define the ranges of continuous variables. In cases where this information is unavailable and estimations are necessary, the performance of DisCo-DSO could potentially be impacted. Furthermore, in our RL experiments, we constrain the search space to univariate decision trees. Exploring more complex search spaces, such as multivariate or "oblique" decision trees, remains an avenue for future research.

## Acknowledgements

We thank Livermore Computing at Lawrence Livermore National Laboratory (LLNL) for the computational resources that enabled this work. Funding was provided by the LLNL Laboratory Directed Research and Development project 21-SI-001. We thank the Computational Engineering Directorate and the Data Science Institute at LLNL for additional support. This work was performed under the auspices of the U.S. Department of Energy by LLNL under contract DE-AC52-07NA27344. LLNL-CONF-865693.

## References

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A., and Parascandolo, G. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pp. 936–945. PMLR, 2021.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Custode, L. L. and Iacca, G. Evolutionary learning of interpretable decision trees. *IEEE Access*, 11:6169–6184, 2023. URL https://ieeexplore.ieee.org/document/10015004.

da Silva, F. L., Goncalves, A., Nguyen, S., Vashchenko, D., Glatt, R., Desautels, T., Landajuela, M., Faissol, D., and Petersen, B. Language model-accelerated deep symbolic optimization. *Neural Computing and Applications*, pp. 1–17, 2023. URL https://link.springer.com/article/10.1007/s00521-023-08802-8.

Ding, Z., Hernandez-Leal, P., Weiguang Ding, G., Li, C., and Huang, R. Cdt: Cascading decision trees for explainable reinforcement learning. *arXiv preprint: arXiv:2011.07553v2*, 2020.

Fletcher, R. *Practical methods of optimization*. John Wiley & Sons, 2000.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A. G., Parizeau, M., and Gagné, C. Deap: Evolutionary algorithms made easy. *The Journal of Machine Learning Research*, 13(1):2171–2175, 2012. URL https://jmlr.org/papers/v13/fortin12a.html.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. URL https://ieeexplore.ieee.org/abstract/document/6795963.

Kamienny, P.-A., d'Ascoli, S., Lample, G., and Charton, F. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.10532*, 2022.

Kim, J. T., Larma, M. L., and Petersen, B. K. Distilling wikipedia mathematical knowledge into neural network models. *CoRR*, abs/2104.05930, 2021. URL https://arxiv.org/abs/2104.05930.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Koza, J. R. *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*, volume 34. Stanford University, Department of Computer Science Stanford, CA, 1990.

Lan, G., Tomczak, J. M., Roijers, D. M., and Eiben, A. Time efficiency in optimization with a bayesian-evolutionary algorithm. *Swarm and Evolutionary Computation*, 69: 100970, 2022.

Landajuela, M., Petersen, B. K., Kim, S., Santiago, C. P., Glatt, R., Mundhenk, N., Pettit, J. F., and Faissol, D. Discovering symbolic policies with deep reinforcement learning. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5979–5989. PMLR, 18–24 Jul 2021a. URL https://proceedings.mlr.press/v139/landajuela21a.html.

Landajuela, M., Petersen, B. K., Kim, S. K., Santiago, C. P., Glatt, R., Mundhenk, T. N., Pettit, J. F., and Faissol, D. M. Improving exploration in policy gradient search: Application to symbolic optimization. In *1st Mathematical Reasoning in General Artificial Intelligence Workshop, ICLR 2021*. arXiv, 2021b. doi: 10.48550/ARXIV.2107.09158. URL https://arxiv.org/abs/2107.09158.

Mundhenk, T., Landajuela, M., Glatt, R., Santiago, C. P., faissol, D., and Petersen, B. K. Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 24912–24923. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/d073bb8d0c47f317dd39de9c9f004e9d-Paper.pdf.

Petersen, B. K., Landajuela, M., Mundhenk, T. N., Santiago, C. P., Kim, S., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021a. URL https://openreview.net/forum?id=m5Qsh0kBQG.

Petersen, B. K., Santiago, C., and Landajuela, M. Incorporating domain knowledge into neural-guided search via in situ priors and constraints. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021b. URL https://openreview.net/forum?id=yAis5yB9MQ.

Popova, M., Shvets, M., Oliva, J., and Isayev, O. Molecularrnn: Generating realistic molecular graphs with optimized properties. *arXiv preprint arXiv:1905.13372*, 2019. URL https://arxiv.org/abs/1905.13372.

Sahoo, S., Lampert, C., and Martius, G. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pp. 4442–4450. PMLR, 2018. URL http://proceedings.mlr.press/v80/sahoo18a.html.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104 (1):148–175, 2015.

Silva, A., Gombolay, M., Killian, T., Jimenez, I., and Son, S.-H. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *International conference on artificial intelligence and statistics*, pp. 1855–1865. PMLR, 2020.

Storn, R. and Price, K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11:341–359, 1997. URL https://link.springer.com/article/10.1023/A:1008202821328.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Tamar, A., Glassner, Y., and Mannor, S. Policy gradients beyond expectations: Conditional value-at-risk. *arXiv preprint arXiv:1404.3862*, 2014.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Xiang, Y., Sun, D., Fan, W., and Gong, X. Generalized simulated annealing algorithm and its application to the thomson model. *Physics Letters A*, 233(3):216–220, 1997. URL https://www.sciencedirect.com/science/article/abs/pii/S037596019700474X.

## A. Pseudocode for DisCo-DSO

In this section we present pseudocode for the DisCo-DSO algorithm. The algorithm is presented in Algorithm 1. We also provide pseudocode for the discrete-continuous sampling procedure in Algorithm 2. Algorithm 2 is called multiple times to form a batch in Algorithm 1. Note that for decision tree policies for reinforcement learning, the distribution, $\mathcal{D}$, used for sampling the next continuous token in Algorithm 2 is only truncated with bounds produced with Algorithm 3 at a decision tree node. Otherwise, the distribution is unbounded.

**Algorithm 1** Discrete-Continuous Deep Symbolic Optimization

```
input  batch size $N$, reward function $R$, risk factor $\epsilon$, policy gradient function $S$, entropy coefficient $\lambda_{\mathcal{H}}$, learning rate $\alpha$
output  Best fitting design $\tau^\star = \langle (l_1, \beta_1), \ldots, (l_T, \beta_T) \rangle^\star$
 1: Initialize AR with parameters $\theta$, defining distribution over designs $p(\cdot|\theta)$
 2: repeat
 3:     $\mathcal{T} \leftarrow \{\tau^{(i)} = (l, \beta)^{(i)}_{1:T_i} \sim p(\cdot|\theta)\}_{i=1}^N$ Sample batch of $N$ discrete-continous designs (Algorithm 2)
 4:     $\mathcal{R} \leftarrow \{R(\tau^{(i)})\}_{i=1}^N$ Compute rewards
 5:     $R_\epsilon \leftarrow (1-\epsilon)$-quantile of $\mathcal{R}$ Compute reward threshold
 6:     $\mathcal{T} \leftarrow \{\tau^{(i)} : R(\tau^{(i)}) \geq R_\epsilon\}$ Select subset of expressions above threshold
 7:     $\mathcal{R} \leftarrow \{R(\tau^{(i)}) : R(\tau^{(i)}) \geq R_\epsilon\}$ Select corresponding subset of rewards
 8:     $\hat{g_1} \leftarrow$ ReduceMean$((\mathcal{R} - R_\epsilon) S(\mathcal{T}, \theta))$ Compute risk-seeking policy gradient
 9:     $\hat{g_2} \leftarrow$ ReduceMean$(-\lambda_{\mathcal{H}} \nabla_\theta \mathcal{H}(\mathcal{T}|\theta))$ Compute entropy gradient
10:     $\theta \leftarrow \theta + \alpha(\hat{g_1} + \hat{g_2})$ Apply gradients
11:     if $\max \mathcal{R} > R(\tau^\star)$ then $\tau^\star \leftarrow \tau^{(\arg\max \mathcal{R})}$ Update best discrete-continuous design
12: return $\tau^\star$
```

**Algorithm 2** Discrete-continuous sampling

```
input  parameters of the given AR $\theta$, token library $\mathcal{L}$
output  sequence $\tau = \langle (l_1, \beta_1), \ldots, (l_T, \beta_T) \rangle$ sampled from the AR
 1: $\tau = \tau_{1:0} \leftarrow [\cdot]$ Initialize empty sequence
 2: for $i = 1, 2, \ldots$ do
 3:     $(\psi^{(i)}, \phi^{(i)}) \leftarrow$ AR$(\tau_{1:(i-1)} = (l, \beta)_{1:(i-1)}; \theta)$ Emit two outputs for each token
 4:     Compute $\mathcal{C}_{\tau_{1:(i-1)}} \subseteq \mathcal{L}$ Compute unfeasible tokens
 5:     $\psi^{(i)}_{\mathcal{C}_{\tau_{1:(i-1)}}} \leftarrow -\infty$ Set unfeasible tokens to $-\infty$
 6:     $l_i \leftarrow$ Categorical$(\psi^{(i)})$ Sample the next discrete token
 7:     if $l_i \in \bar{\mathcal{L}}$ then
 8:         $\beta_i \leftarrow \mathcal{U}_{[0,1]}(\cdot)$ Sample the next continuous token
 9:     else if $l_i \in \hat{\mathcal{L}}$ then
10:         $\beta_i \leftarrow \mathcal{D}(\cdot | l_i, \phi^{(i)})$ Sample the next continuous token
11:     end if
12:     $\tau_i \leftarrow (l_i, \beta_i)$ Joint discrete and continuous token
13:     $\tau \leftarrow \tau \parallel \tau_i$ Append token to traversal
```

## B. Additional algorithm details

**Risk-seeking policy gradient for hybrid discrete-continuous action space.** The derivation of the risk-seeking policy gradient for the hybrid discrete-continuous action space follows closely the derivation in Petersen et al. (2021a) (see also Tamar et al. (2014)). The risk-seeking policy gradient for a univariate sequence $\tau$ is given by

$$\nabla_\theta J_\varepsilon(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta)} \left[ (R(\tau) - R_\varepsilon(\theta)) \nabla_\theta \log p(\tau|\theta) \mid R(\tau) \geq R_\varepsilon(\theta) \right].$$

In the hybrid discrete-continuous action space case, we have $\tau = \langle (l_1, \beta_1), \ldots, (l_T, \beta_T) \rangle$ and

$$p(\tau|\theta) = \prod_{i=1}^{|\tau|} p(\tau_i|\tau_{1:(i-1)}, \theta) = \prod_{i=1}^{|\tau|} p((l_i, \beta_i)|(l, \beta)_{1:(i-1)}, \theta). \tag{1}$$

Thus, using the convenient notation $A(\tau, \varepsilon, \theta) = R(\tau) - R_\varepsilon(\theta)$, the risk-seeking policy gradient for the hybrid discrete-continuous action space is given by

$$\begin{aligned}
&\nabla_\theta J_\varepsilon(\theta) = \mathbb{E}_{\tau\sim p(\tau|\theta)} \left[ A(\tau, \varepsilon, \theta) \nabla_\theta \log p(\tau|\theta) \mid A(\tau, \varepsilon, \theta) \geq 0 \right] \\
&= \mathbb{E}_{\tau\sim p(\tau|\theta)} \left[ A(\tau, \varepsilon, \theta) \nabla_\theta \log \prod_{i=1}^{|\tau|} p((l_i, \beta_i)|(l, \beta)_{1:(i-1)}, \theta) \mid A(\tau, \varepsilon, \theta) \geq 0 \right] \\
&= \mathbb{E}_{\tau\sim p(\tau|\theta)} \left[ A(\tau, \varepsilon, \theta) \sum_{i=1}^{|\tau|} \nabla_\theta \log p((l_i, \beta_i)|(l, \beta)_{1:(i-1)}, \theta) \mid A(\tau, \varepsilon, \theta) \geq 0 \right] \\
&= \mathbb{E}_{\tau\sim p(\tau|\theta)} \left[ A(\tau, \varepsilon, \theta) \sum_{i=1}^{|\tau|} \left( \nabla_\theta \log p(l_i|(l, \beta)_{1:(i-1)}, \theta) + \nabla_\theta \log p(\beta_i|l_i, (l, \beta)_{1:(i-1)}, \theta) \right) \mid A(\tau, \varepsilon, \theta) \geq 0 \right] \\
&= \mathbb{E}_{\tau\sim p(\tau|\theta)} \left[ A(\tau, \varepsilon, \theta) \sum_{i=1}^{|\tau|} \begin{cases} \nabla_\theta \log p(l_i|(l, \beta)_{1:(i-1)}, \theta), & \text{if } l_i \in \bar{\mathcal{L}} \\ \nabla_\theta \log p(\beta_i|l_i, (l, \beta)_{1:(i-1)}, \theta) + \\ \qquad \nabla_\theta \log p(l_i|(l, \beta)_{1:(i-1)}, \theta), & \text{if } l_i \in \hat{\mathcal{L}} \end{cases} \mid A(\tau, \varepsilon, \theta) \geq 0 \right].
\end{aligned}$$

In practice, we use the following estimator for the risk-seeking policy gradient:

$$\nabla_\theta J_\varepsilon(\theta) \approx \frac{1}{M} \sum_{i=1}^{M} \tilde{A}(\tau^{(i)}, \varepsilon, \theta) \sum_{j=1}^{|\tau^{(i)}|} \begin{cases} \nabla_\theta \log p(l_j^{(i)}|(l, \beta)_{1:(j-1)}^{(i)}, \theta), & \text{if } l_j^{(i)} \in \bar{\mathcal{L}} \\ \nabla_\theta \log p(\beta_j^{(i)}|l_j^{(i)}, (l, \beta)_{1:(j-1)}^{(i)}, \theta) + \\ \qquad \nabla_\theta \log p(l_j^{(i)}|(l, \beta)_{1:(j-1)}^{(i)}, \theta), & \text{if } l_j^{(i)} \in \hat{\mathcal{L}} \end{cases} \cdot \mathbb{I}(\tilde{A}(\tau^{(i)}, \varepsilon, \theta) \geq 0),$$

where $\tau^{(i)} = \langle (l_1^{(i)}, \beta_1^{(i)}), \ldots, (l_T^{(i)}, \beta_T^{(i)}) \rangle$ is the $i$-th trajectory sampled from $p(\tau|\theta)$, $M$ is the number of trajectories used in the estimator, and $A(\tau^{(i)}, \varepsilon, \theta) \approx \tilde{A}(\tau^{(i)}, \varepsilon, \theta) = R(\tau^{(i)}) - \tilde{R}_\varepsilon(\theta)$, with $\tilde{R}_\varepsilon(\theta)$ being an estimate of $R_\varepsilon(\theta)$.

**Entropy derivation in the hybrid discrete-continuous action space.** As in Petersen et al. (2021a), we add entropy to the loss function as a bonus. Since there is a continuous component in the library, the entropy for the distribution of $\tau_i$ in the sequence is

$$\mathcal{H}_i = \sum_{l \in \hat{\mathcal{L}}} p(l|(l, \beta)_{1:(i-1)}, \theta) \mathcal{H}_{\beta\sim\mathcal{D}(\beta|l, \phi_l)}(\beta|l) + \mathcal{H}_{l\sim p(l|(l,\beta)_{1:(i-1)}, \theta)}(l).$$

For the derivation, recall that, for a distribution $\mathcal{D}$, the entropy is defined as $\mathcal{H}(\mathcal{D}) = -\int_{-\infty}^{\infty} \mathcal{D}(x) \log \mathcal{D}(x)\, dx$, and that, in DisCo-DSO, we add an entropy regularization term for each distribution $p((l_i, \beta_i)|(l, \beta)_{1:(i-1)}, \theta)$ encountered during the rollout. Thus, we have

$$
\begin{aligned}
\mathcal{H}_i &= -\sum_{l_i \in \hat{\mathcal{L}}} \int_{-\infty}^{\infty} p((l_i, \beta_i)|(l, \beta)_{1:(i-1)}, \theta) \log p((l_i, \beta_i)|(l, \beta)_{1:(i-1)}, \theta)\, d\beta_i \\
&\qquad\qquad\qquad - \sum_{l_i \in \bar{\mathcal{L}}} p(l_i|(l, \beta)_{1:(i-1)}, \theta) \log p(l_i|(l, \beta)_{1:(i-1)}, \theta) \\
&= -\sum_{l_i \in \hat{\mathcal{L}}} \int_{-\infty}^{\infty} p(\beta_i|l_i) p(l) \log \left(p(\beta_i|l_i) p(l_i)\right)\, d\beta_i - \sum_{l_i \in \bar{\mathcal{L}}} p(l_i) \log p(l_i) \\
&= -\sum_{l_i \in \hat{\mathcal{L}}} \int_{-\infty}^{\infty} p(\beta_i|l_i) p(l_i) \log p(\beta_i|l_i)\, d\beta_i - \sum_{l_i \in \hat{\mathcal{L}}} \int_{-\infty}^{\infty} p(\beta_i|l_i) p(l_i) \log p(l_i)\, d\beta_i - \sum_{l_i \in \bar{\mathcal{L}}} p(l_i) \log p(l_i) \\
&= -\sum_{l_i \in \hat{\mathcal{L}}} p(l_i) \int_{-\infty}^{\infty} p(\beta_i|l_i) \log p(\beta_i|l_i)\, d\beta_i - \sum_{l_i \in \hat{\mathcal{L}}} p(l_i) \log p(l_i) \int_{-\infty}^{\infty} p(\beta_i|l_i)\, d\beta_i - \sum_{l_i \in \bar{\mathcal{L}}} p(l_i) \log p(l_i) \\
&= \sum_{l_i \in \hat{\mathcal{L}}} p(l_i|(l, \beta)_{1:(i-1)}, \theta) \mathcal{H}_{\beta_i \sim \mathcal{D}(\beta_i|l_i, \phi_{l_i})}(\beta_i|l_i) - \sum_{l_i \in \mathcal{L}} p(l_i|(l, \beta)_{1:(i-1)}, \theta) \log p(l_i|(l, \beta)_{1:(i-1)}, \theta).
\end{aligned}
$$

Note that we have removed the conditioning elements $\theta$ and $(l, \beta)_{1:(i-1)}$ in some terms in the above derivation for brevity.

## C. Additional experimental results

### C.1. Decision tree policies for reinforcement learning

**Note on "oblique" decision trees.** (Custode & Iacca, 2023) consider multivariate, or "oblique" decision trees. These are trees where the left-hand side of a decision node is composed of an expression, while the right-hand side is still a boolean decision parameter $\beta_n$. While these trees perform well on more complex environments such as LunarLander-v2 (published results report an average test score of 213.09), we do not compare against them here as the search space is drastically different.

**Long run example in LunarLander-v2.** In Figure 5, we show a long run example of DisCo-DSO on LunarLander-v2. The search progress is shown for $4 \cdot 10^7$ function evaluations. We provide a sample of the best traversals found by DisCo-DSO through the process. Note that reward in Figure 5 is the training reward, while reward in Table 1 is the test reward.

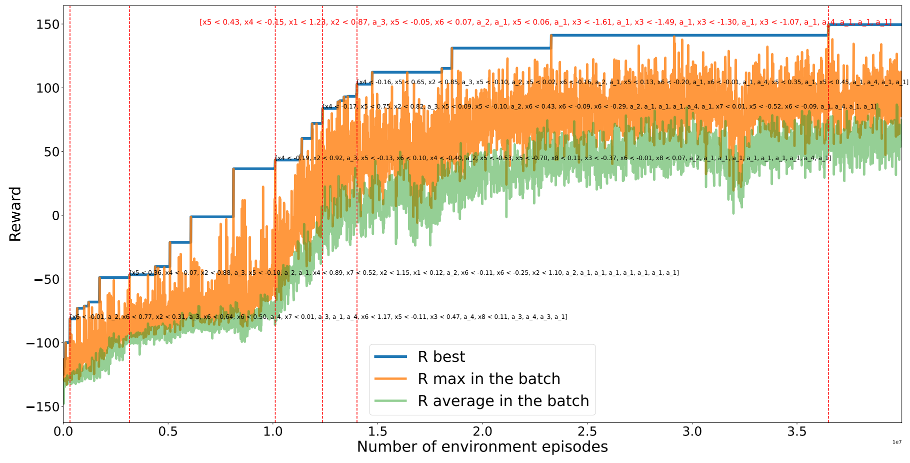


Figure 5: Search progress of DisCo-DSO on LunarLander-v2 for $4 \cdot 10^7$ function evaluations.

### C.2. Hyperparameters

In Table 2, we provide the common hyperparameters used for the RL-based generative methods (DisCo-DSO and Decoupled-RL). The hyperparameters for the GP-based method are provided in Table 3. DisCo-DSO's specific hyperparameters, linked to modeling of the distribution $\mathcal{D}$, are provided in Table 4. For the DT policies for reinforcement learning task, the parameter $N$ (number of episodes to average over to compute a single reward $R(\tau)$) is set to 100.

## D. Prefix-dependent positional constraints

The autoregressive sampling used by DisCo-DSO allows for the incorporation of task-dependent constraints. These constraints are applied *in situ*, i.e., during the sampling process. These ideas have been used by several works using similar autoregressive sampling procedures (Popova et al., 2019; Petersen et al., 2021a;b; Landajuela et al., 2021b; Mundhenk et al., 2021; Kim et al., 2021). In this work, we include three novel constraints that are specific to the decision tree generation task, one on the continuous parameters and two on the discrete tokens.

### D.1. Constraints for decision tree generation

**Constraint on parameter range.** By using the truncated normal distribution, upper/lower bounds are imposed on the parameters of decision trees (i.e., $\beta_n$ in the Boolean expression tokens $x_n < \beta_n$) to prevent meaningless internal nodes from being sampled. In Algorithm 3, we provide the detailed procedure for determining the upper/lower bounds at each position of the traversal. The resolution $h > 0$ is a hyperparameter that controls the distance between the parameters $\beta_p$ at the parent node and the corresponding bounds at the children nodes. This guarantees that the sampled decision trees must have finite depth if the environment-enforced bounds on the features of the optimization problem are finite. Moreover, it also prevents the upper/lower bounds from being too close, which can lead to numerical instability in the truncated normal distribution.

| Parameter | Value |
|---|---|
| Optimizer | Adam (Kingma & Ba, 2017) |
| Number of layers | 1 |
| Number of hidden units | 32 |
| RNN type | LSTM (Hochreiter & Schmidhuber, 1997) |
| Learning rate ($\alpha$) | 0.001 |
| Entropy coefficient ($\lambda_{\mathcal{H}}$) | 0.01 |
| Moving average coefficient ($\beta$) | 0.5 |
| Risk factor ($\epsilon$) | 0.2 |

Table 2: DisCo-DSO and Decoupled-RL hyperparameters

| Parameter | Value |
|---|---|
| Population size | 1,000 |
| Generations | 1,000 |
| Fitness function | NRMSE |
| Initialization method | Full |
| Selection type | Tournament |
| Tournament size ($k$) | 5 |
| Crossover probability | 0.5 |
| Mutation probability | 0.5 |
| Minimum subtree depth ($d_{\text{min}}$) | 0 |
| Maximum subtree depth ($d_{\text{max}}$) | 2 |

Table 3: Decoupled-GP hyperparameters

| Parameter | Value |
|---|---|
| Parameter shift | 0.0 |
| Parameter generating distribution scale ($\sigma$) | 0.5 |
| Learn parameter generating distribution scale | False |
| Parameter generating distribution type | Normal |

Table 4: DisCo-DSO specific hyperparameters

**Algorithm 3** Finding bounds for parameters in decision trees

```
input: parent token $l_p(\beta_p)$, bounds for the parameters of the parent token $\beta_p^{\max}$, $\beta_p^{\min}$
Parameters: Resolution $h > 0$
output: bounds for the parameters of the next token $\beta_i^{\max}$, $\beta_i^{\min}$
$\beta_i^{\max}, \beta_i^{\min} \leftarrow \beta_p^{\max}, \beta_p^{\min}$ Inherit parameter bounds from parent
if the next token is a right child of $l_p(\beta_p)$ then
  $(\beta_i^{\min})_{l_p} \leftarrow \beta_p + h$ {Adjust the lower bound corresponding to $l_p$}
  if $(\beta_i^{\max})_{l_p} - (\beta_i^{\min})_{l_p} < h$ then
    $(\beta_i^{\min})_{l_p} \leftarrow (\beta_i^{\max})_{l_p} - h/2$ Maintain a minimal distance between bounds
  end if
else
  $(\beta_i^{\max})_{l_p} \leftarrow \beta_p - h$ Adjust the upper bound corresponding to $l_p$
  if $(\beta_i^{\max})_{l_p} - (\beta_i^{\min})_{l_p} < h$ then
    $(\beta_i^{\max})_{l_p} \leftarrow (\beta_i^{\min})_{l_p} + h/2$ Maintain a minimal distance between bounds
  end if
end if
Return: $\beta_i^{\max}, \beta_i^{\min}$
```

**Constraint on Boolean expression tokens.** Depending on the values of the parameter bounds, we also impose constraints on the discrete tokens $x_n < \beta_n$. Specifically, when the upper/lower bounds $\beta_n^{\max}$ and $\beta_n^{\min}$ for the parameter of the $n$-th Boolean expression token $x_n < \beta_n$ are too close, oftentimes there is not much value to split the $n$-th feature space further. Therefore, if $\beta_n^{\max} - \beta_n^{\min} < h$, where $h$ is the resolution hyperparameter in Algorithm 3, then $x_n < \beta_n$ are constrained from being sampled.

**Constraint on discrete action tokens.** If the left child and right child of a Boolean expression token $x_n < \beta_n$ are the same discrete action token $a_j$, the subtree will just be equivalent to a single leaf node containing $a_j$. We add a constraint that if the left child of $x_n < \beta_n$ is $a_j$, then the right child cannot be $a_j$.