Discretizing Continuous Action Spaces for Optimal Decision Trees

Verifiable Policies for Continuous Environments in Reinforcement Learning

Mart van der Kuil

Supervisor(s): Anna Lukina, Daniël Vos

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Mart van der Kuil
Final project course: CSE3000 Research Project
Thesis committee: Anna Lukina, Daniël Vos, Luciano Cavalcante Siebert

Abstract

Complex reinforcement learning (RL) models that receive high rewards in their environments are often hard to understand. To this end, more interpretable models can be used, such as decision trees. To be able to deploy these models in safety-critical environments, they need to be high-performing and verifiable. Optimal decision trees can fulfill both these goal. While some methods already exist to find optimal decision trees, none have applied them to RL environments with continuous state and action spaces [4; 13]. Broccoli is a state-of-the-art approach to synthesizing decision tree policies for black-box environments [3]. Given a discretisation of the continuous state space, it can find the optimal tree in discrete action environments.

This research will focus on discretising action spaces of continuous environments. In doing so, discrete action spaces are created for which Broccoli can find optimal decision trees. Additionally, a goal is to show that it is possible to discretise continuous action spaces and compute corresponding optimal decision trees in feasible time. Furthermore, this paper proposes informed discretisation techniques that result in better-performing decision tree policies.

1 Introduction

The practicality of Reinforcement Learning (RL) applications relies on the ability to generate well-performing policies. However, most of these well-performing policies are considered black-boxes as these models cannot be easily understood. This lack of interpretability and verifiability poses a problem in safety-critical systems, such as autonomous driving or robotic behaviour, where interpretability is necessary [5; 8].

Decision trees are intrinsically interpretable and therefore verifiable [6]. Because of these properties, there is an increasing amount of research into decision trees for RL and finding competitive models. The Broccoli algorithm finds optimal decision trees for RL environments with continuous state spaces [3; 4]. Broccoli achieves promising results in discrete action environments. However, real-life applications often are situated in continuous action environments.

This research paper focuses on discretising continuous action spaces for the Broccoli algorithm. This will be done such that optimal decision trees can be found for continuous environments. The main research question the paper will address is:

How do different discretisation techniques of continuous action spaces affect the performance of the Broccoli algorithm and its decision trees within deterministic black-box environments?

In this study, the first experiment will be done to determine the effect of the number of actions on the Broccoli algorithm. The second experiment will evaluate the effect of skewed discretisations on the performance of the optimal and runtime of the algorithm. After which, methods for discretisation will be introduced that are evaluated on the return of the optimal tree and runtime.

There will be a focus on comparing different discretisation techniques and their settings. In the following chapter, the related work will be highlighted. After that, the third chapter will explain some technical details of the Broccoli algorithm. The fourth chapter will describe the methodology used. The fifth chapter will contain the experiments and their results. Following this, the sixth chapter will address some ethical aspects of this research. Finally, the seventh chapter concludes the main research question, lists the limitations of this study, and provides possible paths for future work.

2 Related Work

Optimal Decision Trees Decision trees have gained significant traction in research for explainable AI. In a decision tree, each decision is represented by an inner node, and the actions by a leaf node. It is due to this logical structure that decision trees are interpretable. The structure makes it easy to follow the reasoning behind a decision taken. One of the challenges of decision trees is dealing with continuous state and action spaces, and finding the optimal tree is known to be NP-hard [9].

The Broccoli algorithm provides a solution for finding optimal continuous-state decision trees through exhaustive search [3]. The algorithm uses a pruning technique based on traces that assumes no prior knowledge of the environment. One of the shortcomings of the algorithm is that it currently only works on discrete action environments. There are also other papers that focus on finding optimal decision trees.

In MurTree [2], they also apply a search algorithm for finding depth-limited trees, though these trees are classification trees. In OMDT [13], they show that an optimal decision tree with limited depth can be found, given a Markov Decision Process. One limitation of this method is that the environment's Markov Decision Process is needed, which is not possible in the case of a black-box environment. Currently, there is minimal research regarding optimal decision trees in continuous environments, considering black-box environments.

Continuous action discretisation In various applications of reinforcement learning, there is a need for continuous control. Whether in robotics or game-playing, continuous control results in an infeasible action space. This is due to the fact that continuous spaces are infinite. To compute the optimal decision tree, you would have to evaluate all actions, which is not feasible for continuous spaces. To combat this, an abstraction or discretization can be used to reduce the action space.

In poker-playing agents for no-limit hold 'em, agents have an (almost) infinite number of choices for the size of each bet. Some use a geometric progression of the current pot size to reduce the action space for the agents to a more tractable size [10]. While it works in these agents, this has not yet been applied to decision trees. As for reinforcement learning for motor skills, AQuaDem finds useful discretisations from a dataset of human demonstrations [1]. This technique proves useful but is unfit for the classic control environments.

Research Gap There is a lack of research on optimal decision trees in continuous action environments. In addition to this, there are almost no methods that find useful action space discretisations for decision trees, while they exist for other applications. To address these issues, this paper aims to find methods that are able to successfully discretise action spaces in continuous environments. By using the Broccoli algorithm, these action discretisations can be used to find optimal decision trees in continuous environments [3].

3 The Broccoli Algorithm

As this research builds upon the Broccoli algorithm, some background knowledge is necessary. This chapter will include the specific background knowledge of the *In Search of Trees* [4] paper needed for this paper. The first section will start with some preliminaries taken from the paper. Then the second section will describe the naive search algorithm. After this, the pruning technique specific to Broccoli will be laid out in the third section. That section will also elaborate on why it matters for the discretisation.

3.1 Preliminaries

Let $S = (s_1, s_2, ..., s_d)^T \in \mathcal{S}$ be a *state* which has d dimensions and where $\mathcal{S} \subseteq \mathbb{R}^d$. Let *action* $a \in A \subseteq \mathbb{R}$. Let an *environment* be a function $E : \mathcal{S} \times A \to \mathcal{S}$. The environment takes an action and a state as arguments, and will return the resulting state, $S' = E(S, a)$. A *policy* is a function given a state that it returns an action, $\pi : \mathcal{S} \to A$. $\mathcal{C}_A$ denotes the set of all policies.

In the paper, the focus will be on decision tree policies. This decision tree is a binary tree where the leaf nodes represent specific actions and inner nodes are *predicates*. The predicate nodes are boolean functions in the form of $\mathcal{S} \to \mathbb{B}$, where given the input state, the node will return either *true* or *false*. The decision tree policy works as follows. At a given state, the policy starts at the root node, it then evaluates the predicate function and goes to its left child if it is evaluated to *true*, else it will go to its right child. If this node is also a predicate node, it repeats this process until an action node is encountered and returns its action.

Using a policy π, and environment E, an initial state S_0, and a limited amount of total timesteps $k \in \mathbb{N}$, a trace is formed by the states the policy will traverse in the environment. Formally this can be written as $\tau = S_0, S_1, S_2, ..., S_k$ where S_i is given as $S_i = E(S_{i-1}, \pi(S_{i-1}))$ for $i = 1, ..., k$. $\mathcal{T}$ is written to denote the set of all traces.

A fitness function is assumed, as the goal is to find the optimal decision tree policy given an environment and limited depth. $\succeq$: $\mathcal{T} \times \mathcal{T}$ is the partial ordering of two traces. It is said that if a trace finds a goal state in an environment, it always precedes a trace that does not find one of these states. The return of a trace is the sum of rewards at each state S_i along the trace. τ_1 is said to precede τ_2 if τ_1 has a higher return than τ_2. From this ordering follows a logical ordering for policies, $\succeq$: $\mathcal{C}_a \times \mathcal{C}_a$ where $\pi_1 \succ \pi_2$ if either one of the following conditions holds. The corresponding trace of policy π_1 is strictly better than that of π_2. Or both traces have the same fitness, but π_1 is a smaller tree.

3.2 Exhaustive search

The final tree is ensured to be optimal by enumerating all depth-limited decision trees and keeping track of the best one so far. However, the search space of these decision trees is infinitely large. Discretising the predicate nodes makes the search space finite for discrete action environments. The action space can also be discretized to make the search space for continuous action environments finite. As mentioned before, the optimal tree will be found by keeping track of the best-performing tree in exhaustive search. This method is still exponential in time as the search space grows exponentially. Trace-based pruning was used to eliminate certain decision trees from consideration. This technique ensures that some sub-optimal trees will not be explored.

3.3 Trace-based pruning

To reduce the runtime and the number of trees evaluated, Broccoli uses a pruning algorithm based on the traces the tree policies generate in the environment. The reasoning is that a predicate should only be considered if it would change the trace. In the search, the minimal value for which a predicate is *true* is tracked. If the next threshold value considered is less than the lowest observed value, it will not change the trace if selected. If the threshold value were selected, the following policy would produce a duplicate trace, which is unnecessary.

A practical example is shown in Figure 1, which was taken from the original paper [4]. As shown in the figure, the first two predicates do not create different traces. This means that they are identical for all predicates for s_1 that are less than 2.3. After the evaluation of the first policy, it is found that the s_1 does not fall below 2.3, so it is possible to not consider the second tree as it is identical to the first tree.

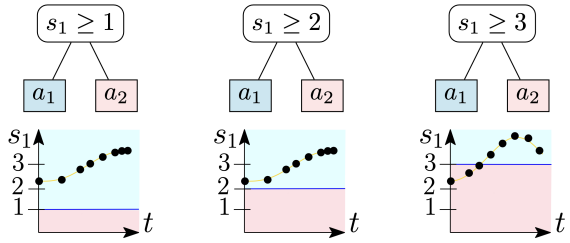
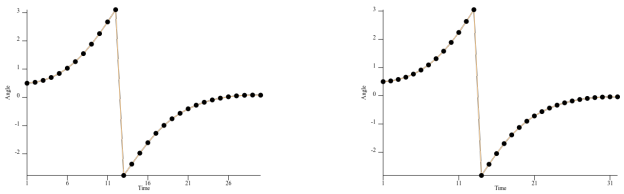


Figure 1: Trace-based pruning applied visually, taken from the *In Search of Trees* paper [4].

Action discretisation implications Despite the pruning not exploiting actions directly, the choice of actions does affect the technique's effectiveness. The pruning technique prunes between two values if the distance is large enough. Take, for example, the traces depicted in Figure 2. In the trace with the larger actions shown in Figure 2a, the policy only needs to take 29 steps to reach the goal state. In Figure 2b, the policy has to take 31 steps to reach the terminal state. Despite both policies using a similar path, the larger-action policy takes fewer steps. Using fewer steps means the space between state values is larger for the policy with larger action values. These larger values possibly result in additional pruning for searches with action sets that contain larger action values.



(a) Pendulum trace with discretised actions -2, 0, 2. 29 actions taken

(b) Pendulum trace with discretised actions -1, 0, 1. 31 actions taken

Figure 2: Graphs showing different traces reaching the same goal, using the same path, in a different number of steps. These indicate that the distance between steps is larger in sub-figure (a).

4 Continuous Action Discretisation

This chapter will outline the motivation behind the discretisation techniques and the main research questions this paper aims to answer. The first two sections will cover the first research sub-question: *How does discretisation of continuous action spaces influence the performance of the Broccoli algorithm?*. After that, the third section will address the second research sub-question: *How can other optimal trees help find a good search space for the final optimal tree?* The quantitative measure of a search space is the total return of the optimal decision tree it contains. The third section will describe how state-of-the-art RL models can help optimal decision tree search. *How can teacher models contribute to the synthesis of optimal decision trees?*

4.1 Simple discretisation

The search space for decision trees is $\mathcal{O}(|P|^n |A|^{n+1})$ where $|P|$ is the number of discretised predicates, $|A|$ the number of discretised

actions, and n is the number of predicate nodes. Note that the number of possible predicate nodes scales exponentially with the depth of the tree. To successfully discretise continuous action spaces, it is essential that the trace-based pruning keeps performing well under an increasing number of actions. An experiment will be conducted where the Broccoli algorithm will be run with varying uniformly discretised actions, both with and without trace-based pruning. Uniformly distributed actions will be used as this approximates the continuous space accurately as the number of actions grow.

4.2 Other static discretisations

In the second experiment, the idea is to use different distributions of geometric progressions and to use these values to generate a more skewed distribution. This discretisation has been used successfully in poker agents to reduce the continuous betting space [10]. These geometric discretisations will be compared to the uniformly distributed baseline used in experiment 1.

There is a discretisation with skewness towards the upper end of the value range, as you can see in Figure 3. A geometric recession will be used to create an identical skewness towards zero. This discretisation is similar to the geometric progression, where the differences decrease instead of increasing. These skews make it possible to see if larger action values influence the trace-based pruning.

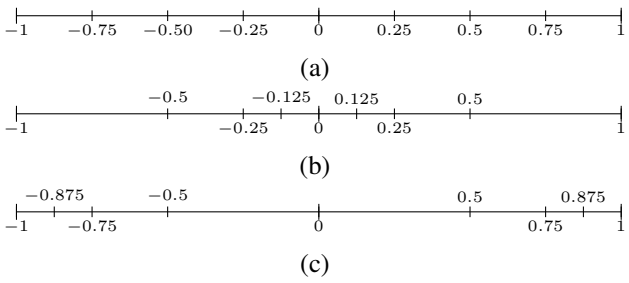


Figure 3: Different static discretisations used. From top to bottom, Uniform (a), Geometric Progression (b), and the Geometric Recession (c).

These distributions of actions do have some drawbacks as they are static and do not differ a lot from each other. While the distributions are asymmetrical on each side of the zero line, the total action distribution is still symmetrical.

4.3 Informed discretisation

As the previously mentioned discretisations are static, they do not take advantage of the environment. It is not possible to try to exploit the environment directly without violating the black-box constraint. Therefore, other policies are generated without assuming any information about the environment. The resulting models are then simulated in the environment, and actions are tracked. By sampling from these actions, representative actions are found which can be used as a discretisation. This is more effective than directly using a more granular discretisation as the search space scales exponentially with the number of actions. Even though the sampling of actions scales better, the practicality of this policy sampling still depends on the time it takes to generate the policy and sample from the environment.

4.3.1 Cheap optimal trees

The first informed discretisation will use cheaper, optimal trees found by the Broccoli algorithm. This method will generate random actions and filter them through the Broccoli algorithm by searching through cheaper runs. Cheaper runs either use fewer depth, fewer predicates, or fewer actions. The goal of these reductions is to shrink the solution space. The formal algorithm for this sampling is given in Algorithm 1. Note that the algorithm presented is the algorithm for cheaper trees by reduced action set sizes. One could employ a similar algorithm to use cheaper searches with either fewer predicates or shallower trees.

Algorithm 1 Cheap Optimal Tree Sampling

Input: number of actions n, predicates $\mathcal{P}$, maximum depth d, number of tree samples k, number of top trees m
Output: Optimal decision tree t^* that produces the highest return

$T = \varnothing$
for i **in** $1,\ ...,\ k$ **do**
  $A_1 \leftarrow \{X_1, ..., X_{\lceil n/2 \rceil}\}$, $X_i \sim U(0, max\_action)$
  $A_2 \leftarrow \{Y_1, ..., Y_{\lfloor n/2 \rfloor}\}$, $Y_i \sim U(min\_action, 0)$
  $\mathcal{A} \leftarrow A_1 \cup A_2$
  $t_i \leftarrow \text{Broccoli}(\mathcal{P}, \mathcal{A}, d)$
  $T \leftarrow T \cup \{t_i\}$
end for
$T_{list} \leftarrow \text{Sort}(T,\ Descending)$
$\mathcal{A}' \leftarrow \varnothing$
for i **in** $1, ..., m$ **do**
  $a_i \leftarrow \text{GetActions}(t_i)$
  $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{a_i\}$
end for
$t^* = \text{Broccoli}(\mathcal{P}, \mathcal{A}', d)$
return t^*

This experiment will focus on reducing the number of actions. The advantage of this method is that the expensive, final run has informed action selection. Of the cheaper trees, only the better-performing trees' actions will be used. The cheaper runs' actions will be drawn from the uniform distribution over the action range. By only selecting the best trees, the chance of sampling from anomalously bad trees is reduced.

Another benefit of this approach is that there is a lower bound on the return of the final optimal tree. All actions from the cheaper trees get included, and the final tree search algorithm will search for at least the depth of the cheaper trees. Therefore, each smaller tree is included in the search space, and the maximum return of these trees is the lower bound for the final, optimal tree (Theorem 1).

Theorem 1. *Given predicates $\{P_1, ..., P_n\} \subseteq \mathcal{P}$, action set $\mathcal{A} = \{A_1, ..., A_n\}$ where P_i and A_i, are the predicates and actions respectively of the sampled decision trees $t \in T$, with maximum depth k, and n is the number of trees in T. Let r_t be the return t produces in a given environment. Let t_{max} denote the tree with the greatest return such that $t_{max} = \arg\max_{t \in T} r_t$. The Broccoli algorithm, given action set $\mathcal{A}$, predicate set $\mathcal{P}$, maximum depth m where $m \geq k$, returns an optimal tree t^* where $r_{t^*} \geq r_{t_{max}}$*

Proof. Let $S^m_{P,A}$ be the search space of all decision trees with predicates $p \subseteq \mathcal{P}$, actions $a \subseteq \mathcal{A}$, and depth up until m By definition of the Broccoli algorithm, given predicates $\mathcal{P}$, actions $\mathcal{A}$, maximum depth m, it finds the optimal tree t^* such that $t^* = \arg\max_{t \in S^m_{P,A}} r_t$. Given that $\forall i : S^k_{P_i,A_i} \subseteq S^m_{P,A}$ where $k \leq m$. It follows that $\forall i : \{\forall t \in S^k_{P_i,A_i} : r_{t^*} \geq r_t\}$. Therefore, with $t_{max} = \arg\max_{\forall t \in T} r_t$, it can be concluded that $r_{t^*} \geq r_{t_{max}}$ QED

As seen in Theorem 1, the actions used in multiple decision trees can be given in one superset as input to the Broccoli algorithm. By

using the same predicates and depth, a lower bound is guaranteed on the return of the optimal tree found. This is given by the greatest return of the decision trees that sampled the actions.

4.3.2 Twin-Delayed Deep Deterministic Policy Gradient (TD3)

The second heuristic discretisation is using the actions of a teacher model. Here, the sampling is done through the Twin-Delayed Deep Deterministic Policy Gradient (TD3) model, as this is a widely accepted working model [7; 15; 14]. The agent will be trained in the same environment, and then the environment and the agent will be used to generate actions.

The sampling is done by running the agent on the environment and finding which actions it has taken. Then the sampling is repeated, but for another random initial state, such that the resulting actions will differ. Finally, a subset of actions is randomly chosen from each simulation. The action set is then either accepted or rejected if its mean and standard deviation are representative of the total distribution. The goal is to find an action set that represents the overall action distribution of a well-performing policy. As such, these values are taken as approximations of the distribution to evaluate if the given subset is representative enough.

5 Experimental Setup and Results

This chapter will give the details and setup of the experiments.

5.1 Experiment #1: Simple Discretisation

This section will detail how the first experiment was conducted and which considerations were made.

First, there needs to be an idea of how many discretised actions can be used to find the influence of discretized actions on the Broccoli algorithm. The more actions that can be used, the more accurately the continuous space will be approximated. The standard Broccoli algorithm was developed for discrete action environments that have a constant number of actions. This means that the influence of the number of actions on the algorithm is still unknown.

To find how Broccoli behaves under an increasing number of actions, the idea is to run the search algorithm with and without trace-based pruning. For both variants, multiple uniform action sets of increasing size will be used. The metrics that will be looked at are the runtime, number of environment calls, and number of explicitly considered trees. Both variants are guaranteed to find the same tree.

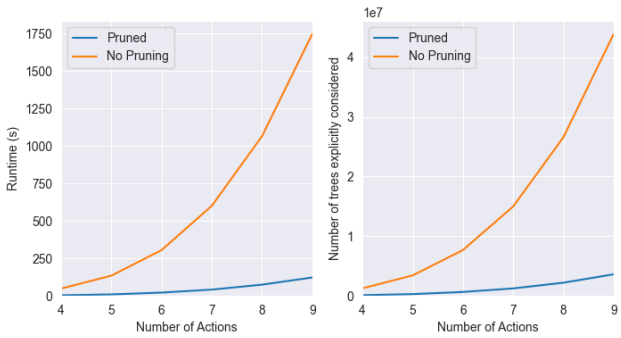


Figure 4: Trace-based pruning vs. Naive exhaustive search for depth two decision trees for the Mountain Car Continuous environment. Both algorithms found the same tree.

As can be seen in Figure 4, the pruning is relatively effective, even for larger numbers of actions. Note that the runtime and number of considered trees still grow exponentially as the number of actions grows. This means that computation will become infeasible for larger numbers of actions.

5.2 Experiment #2: Geometric discretisations

As the goal is to compare the geometric discretisations to the uniform discretisation, function parameters will be the same as in experiment #1. The difference is in the actions chosen. While it is the main difference, what can be noticed in Figure 3 is that the discretisations overlap. As you can see, the actions $[-1, -0.5, 0, 0.5, 1]$ are shared between all three. The only actions which are unique to one discretisation are ± 0.875 and ± 0.125

Table 1 shows that the runtime and trees considered differ significantly for some of the discretisations. There are multiple possible explanations. For the Cart Pole environment, the algorithm could quickly find a tree that reaches the maximum number of iterations. This was also found in the *In Search of Trees* paper as an explanation for significantly reduced running time in the Cart Pole environment [4].

Table 1: Performance of the Broccoli algorithm regarding uniform and geometric action discretisations

Discretisation	Score	# of Trees Considered	Runtime (s)	Episode Length
Mountain Car Continuous				
Uniform	97.5	3.60×10^6	122.0	559
Geo. Pro	**98.6**	$\mathbf{3.40 \times 10^6}$	122.6	657
Geo. Rec.	95.9	3.54×10^6	**119.1**	**404**
Pendulum				
Uniform	**-58.8**	1.11×10^9	4920	**30**
Geo. Pro	**-58.8**	1.21×10^9	5506	**30**
Geo. Rec.	-58.9	$\mathbf{1.02 \times 10^9}$	**4624**	**30**
Cart Pole Continuous				
Uniform	$\mathbf{10^4}$	$\mathbf{3.07 \times 10^6}$	**3.93**	$\mathbf{10^4}$
Geo. Pro	$\mathbf{10^4}$	10.08×10^6	15.26	$\mathbf{10^4}$
Geo. Rec.	$\mathbf{10^4}$	3.79×10^6	5.44	$\mathbf{10^4}$

Another reason could be that the trace-based pruning cannot be applied as much as with specific actions, as mentioned in 3.3. There is no clear discretisation for which Broccoli consistently considers fewer trees. This indicates that the underlying dynamic between action values and trace-based pruning is non-trivial.

Based on the results, no single discretisation performs better than the others in every environment, for any of the metrics. This means that the discretisations are mainly dependent on the specific environment they are used for. What can be noticed is that the uniform discretisation never performs the worst. This is intuitive as it caters decently to environments that reward smaller or larger action values instead of only one of those. Furthermore, the uniform discretisation approximates the continuous action space the most accurately of the three options.

5.3 Experiment #3: Cheap optimal trees

In experiment 3, random action sampling is used to compute optimal decision trees. This method allows us to generate random actions and filter them through the Broccoli algorithm by searching through cheaper solution spaces. Cheaper runs either use a lower maximum depth, fewer predicates, or fewer actions. This is done to shrink the solution space.

This experiment will focus on selecting fewer actions. This is done as the structure of the trees is preserved if the same maximum

depth is used. If the depth were limited, structural change in the tree could induce different tree dynamics for which certain actions perform well. A difference in depth between sample searches and the final search may lead to actions that are only optimal at the previous depth and prove to be sub-optimal at the final depth. To prevent this, the sample runs will use the same maximum depth as the final run.

There will be 8 sample runs with uniformly distributed actions. Three actions will be sampled from [-1, 0] and three actions from [0, 1]. For the Pendulum environment, the respective ranges will be [-2, 0] and [0, 2]. Symmetry was kept such that the total action distribution would not be too dissimilar to the static discretisations.

These runs will use a depth of two, which means the trees will have a maximum of four actions. By taking the three best-performing trees, an action set is formed from their actions. This action set is then used for the final run. This run uses the same predicates and depth as the sample runs. The top sample trees found were not complete trees. This means that the trees used fewer than the 12 possible total actions.

Table 2: Performance of the Broccoli algorithm for action sampling from cheaper decision trees

Mountain Car Continuous			
	Sample Mean	Sample Max (Min)	Optimal
Score	96.96	98.66	**98.79**
Runtime(s)	27.17	**(23.29)**	230.86
Pendulum			
Score	-61.52	**−59.57**	**−59.57**
Runtime(s)	946.35	**(880.42)**	5488.92
Cart Pole Continuous			
Score	$\mathbf{10^4}$	$\mathbf{10^4}$	$\mathbf{10^4}$
Runtime(s)	2.563	(**0.026**)	0.73

As can be seen in Table 2, the scores of the optimal trees are not that different from the maximum scores found in the smaller trees. In the Pendulum environment and the Cart-Pole environment, they find trees with the same score. This means that optimal action selection does not necessarily result in a score increase. If one compares these results to those of the static distributions, it is found that the results do not differ a lot. The static discretisations even perform better for the pendulum environment.

Another explanation could be that the trees used are too optimal. This is meant in the same sense as a learning model that is too exploitative in the learning process. A point of interest for future research could be using sub-optimal trees to explore the action space more. One could include optimal trees of a more limited depth to create structural differences between the trees.

Table 3: Amount of actions considered in the 8 sample searches vs. the final search

Environment	**Unique actions**	**Actions in final search**	**Space Reduction**
Mountain Car Continuous	42	11	73.8%
Cart Pole Continuous	45	9	80.0%
Pendulum	44	9	79.5%

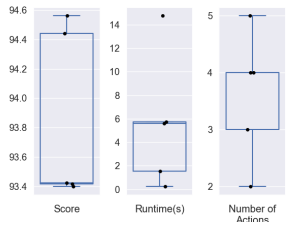
An interesting result is found in Table 3. It can be seen that while a lot of unique actions were considered, the final action space is still feasible to compute. This shows that cheaper trees can be used to filter out sub-optimal action combinations. A drawback is that optimal actions might be filtered out as they were contained in different action sets. By not being in the same action set, their combination was not considered, although it may be optimal. Despite this drawback, this procedure circumvents the exponential search space increase.

5.4 Experiment #4: Twin Delayed DDPG

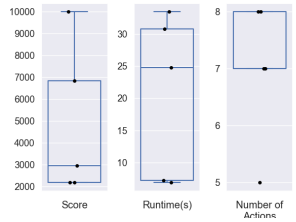
In this experiment, TD3 will be the model that will be sampled from. To get well-performing models to sample from, a model was trained for each of the environments. Mountain Car Continuous was an exception for this. While a model was trained, it did not converge to a policy that could find the goal state. For this reason, a pre-trained model was used to simulate from. One of the limitations of this was that the model was trained on an environment with slightly different physics than the environment that the Broccoli algorithm runs on.

As for the other two environments, the models that were trained performed well. Hyperparameters for training the Pendulum environment were taken from RL baselines3 Zoo [12]. Similar hyperparameters were used for the Cart Pole environment, such that it converged to a policy that would reach maximum iterations.

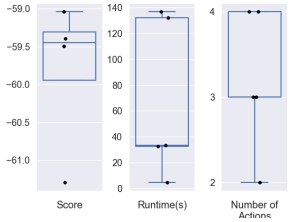
After training, the policy generated a trace from an initial state in the environment. The actions were tracked up until a maximum number of iterations or until the goal state was reached. This was done for 10 random initial states for each of the environments. All the actions of the 10 traces were put together to form an overall distribution. Then, for each environment, 5 random sets of 8 actions were taken from the respective action distribution. If a set did not have a representative mean and standard deviation, the sampling would be repeated for that set. This was done such that the actual actions used were representative of the total distribution. Actions that were within 0.01 were deemed duplicates, and only one of these actions was kept. These action sets were then used to find optimal decision trees.



(a) Mountain Car Continuous, force = 0.0015



(b) Cart Pole Continuous



(c) Pendulum. There are only four scores, as one search could not find a tree that reaches the goal state.

Figure 5: Policy results of TD3-informed action sets with Broccoli.

It can be seen in Figure 5 that overall, this method did not perform

that well compared to the other experiments. Of the environments, the results are most promising for the pendulum environment. The top-performing tree had a return close to the greatest return of a tree in the paper. Especially when one considers the runtime of the algorithm. The training of the TD3 model (1015 seconds) and all the samples' searches ($\sim$339 seconds) combined were faster than any of the search times for the static discretisations as seen in Table 1. Despite the reasonable performance, one search did not find any tree that reached the goal state. This was most likely due to the highly asymmetrical action space, as it only contained negatively valued actions.

As for the Cart Pole environment, this discretisation does not work adequately. Most searches did not find a tree that would max out the iterations, while other discretisations did this consequently. After further inspection, it was concluded that the TD3 was too efficient in this environment. After a few initial actions, the pole was almost perfectly balanced on the cart, needing only actions in the order of 10^{-6}. The resulting action distribution would consist mostly of zeros. When sampled from this distribution, it would return mostly zeros. The effect of this was reduced by using only 1000 iterations of sampling, but as one can see, to no success.

The Mountain Car Continuous environment was modified as the TD3 model was trained on an environment with different physics. The difference was the force that each action was multiplied by. The Broccoli algorithm uses an environment with a force equal to 0.001, while the other environment uses 0.0015. Despite this difference, it would be expected that the environment with the higher force would have a higher return. This is since it could use lower valued actions than the other environment to reach the same trace. Specifically, the fraction of these forces is $\frac{0.0010}{0.0015} = \frac{1}{1.5} = \frac{2}{3}$. This means that one could use actions $\frac{2}{3}$ of the value of the actions in the environment with force equal to 0.0010. The reward function in both environments punishes higher valued actions as seen in Equation 1.

$$reward = \begin{cases} 100 & \text{if state is terminal} \\ -0.1 \times action^2 & \text{else} \end{cases} \tag{1}$$

With this information, it can be concluded that the TD3 environment should perform better with the same actions. However, one can see that the converse is true. Even though the trees in the TD3 environment should perform better, the performance is still lacking. It can be concluded that also for the Mountain Car Environment, this is not a well-performing discretisation compared to the others.

6 Responsible Research

This chapter will describe the considerations for this paper regarding the reproducibility of the study and the ethical aspects. Section one will outline the reproducibility, while section two will focus on highlighting certain ethical implications of the study.

6.1 Reproducibility

To ease the effort in reproducing this study, a codebase is made available in which the experiments were conducted [1]. Furthermore, the Python classes/notebooks that were used for analysis are contained in this codebase. As such, the exact executions that ran are described in a README.md file, and further usage of the codebase is explained briefly, such that follow-up research can be done more easily.

As this paper focuses on a deterministic algorithm, reproducibility can be exact, and no random seeds are needed. What has to be considered is that for the third and fourth experiments, the actions that were used as input for the algorithm were generated by a random process.

In experiment three, the sample method described is trivial and can be easily replicated. As multiple samples were taken, the chance that all the samples are anomalies is much smaller than that one sample was taken. Contradictory, since only the top trees' actions were taken, the results have an increased chance of being anomalous.

In experiment 4, the actions were taken from a model trained on the environment, which is a non-deterministic function. To aid in reproducing the experiment, the model file used as policy will be made publicly available such that its performance can be easily compared to benchmarks. The comparison was not done as part of the study, as this was not included in the scope of the research.

Further considerations that were taken include comparing the mean and standard deviation of sampled actions to the mean and standard deviation of the whole action population. This was done such that the samples taken are a representative set of the model. This also aids in reproducibility as there are certain constraints to the actions used.

6.2 Ethical Considerations

While this study has so far focused on the technical parts of optimal decision trees, reinforcement learning, and black-box environments, it is also important to discuss the ethical implications of this research.

Small decision trees As small decision trees are more interpretable than larger trees, using smaller trees is beneficial for explainability and verifiability [11]. While these smaller models often lack performance, larger models can be much harder to interpret. As such, one should balance a trade-off between result and interpretability.

Optimality Guarantee While factually correct, the claim of optimality can be misleading. This is due to the fact that the decision tree that is computed is optimal *given* the input parameters. Note that this does not mean that of all possible decision trees, it is the most optimal one. Certain constraints are used to decrease the search space. Most notable is the depth limitation. While this does aid in the explainability of the resulting model, it does not help in the case of optimality, as there is an infinite number of decision trees left unchecked. The same holds for the action and state spaces of the environments. As they are continuous and thus infinite, only some values are checked, and approximations of those spaces are used.

Use of Artificial Intelligence In this research, Artificial Intelligence (AI) was used. To be able to claim that this research was done with due diligence, the directly used answers from chatbots can be found in the appendix with a link to the conversation. This way anyone can view the chat and the tasks that were delegated to those chatbots. Exact rewordings of text was also used to help with grammar and sentence flow. All of these instructions were asked in the form of *"Can you give me 5 ways to write this differently? {QUOTE}"*. None of the answers were taken directly, but rather were used in either forming an entirely new sentence or adapting the human-written one. As rewordings were used multiple times, examples will be shown. Stating all conversations is deemed unnecessary and infeasible.

7 Conclusion

This paper demonstrated the ability of discretising action spaces to find optimal decision trees for continuous action environments with the Broccoli algorithm [4].

[1] https://github.com/mmjvanderkuil/discretising-continuous-action-spaces

The first method of discretisation was the uniform approximation. The decision trees were found in a feasible time. However, an increase in action set size still resulted in exponential growth of runtime and search space.

The uniform discretisation was additionally compared to geometric discretisations. There was no clear winner among these discretisations. However, the uniform discretisation never performed the worst and thus could serve as a solid baseline. It was furthermore found that action values had a complex relationship with the trace-based pruning of the Broccoli algorithm [3].

One method used was sampling optimal decision trees of smaller search spaces, which were found to inform which actions the final search should consider. The usefulness of this method was demonstrated by a proof of a lower bound on the return of the final decision tree. Additionally, the reduction in action spaces was proven to be significant, while remaining competitive in performance. Furthermore, while the performance of this method was shown to be superior to that of the static discretisations, it was not considerably greater.

In another method, state-of-the-art teacher models were used to find appropriate actions in the action space. However, this was not proven to be as successful as the other discretisations. For only one of the environments, this discretisation was performed on par with the other discretisations. Despite these results, the discretisation still has a use. The trees found in the Pendulum environment were as performant as the other discretisations' trees, while reducing sampling and search time.

Limitations

As almost all actions were taken to be symmetrical around the zero point. This was done to reduce incredibly asymmetric action sets, as the action space is inherently symmetric. This could prove to be a sub-optimal approach. While the TD3 model was trained in this experiment, it did use finetuned hyperparameters, which breaks the black-box constraint. This was done to be able to sample from competitive models and not influence the results too much. The Mountain Car Continuous environment differed slightly from the gymnasium implementation as the force of the normal Mountain Car environment was used. Though this should not discredit the results, as this reduced force would prove to be negatively influencing the return (See 5.4). The TD3 model was trained on the actual gymnasium version, and its corresponding Broccoli runs also used the correct version.

Future Work

To objectively measure the performance of the Broccoli algorithm, the predicates were untouched for each of the experiments. Possible future work could look into how one can balance the number of predicates and actions to find possibly deeper trees or a more accurate approximation of the continuous action space.

Another area of interest is regarding different tree dynamics for sampling cheaper trees. This research focused on a simple case where the predicates and maximum depth were kept the same. A constraint was given that these trees were also the same depth as the final tree. This prevents structural differences in the tree sample, which could deceptively return sub-optimal actions for the final depth, while optimal at the reduced depth.

New research could explore the usefulness of using other trees synthesisation techniques to generate decision trees. These trees can then be used in an experiment similar to experiment 3. This method could use multiple trees that are already well-performing but not yet proven to be optimal. The resulting tree would either be one of the given trees or a trees which was unknown. Either way, a tree is proven to be optimal.

A Appendix

A.1 ChatGPT Conversations

Conversation #1

Task given: To help in adapting the TD3 algorithm for the continuous environments and speed up the experiment set-up time.

Link to conversation: https://chatgpt.com/share/683c68b4-5c98-8009-8d83-9530bdc247cb

Used result: Started from the file ChatGPT gave and fixed code bugs until it worked. Changed certain physics parameters for the CartPoleContinuous environment

A.2 Github Repository

The codebase used, experiments ran, and results found in this paper are located at: https://github.com/mmjvanderkuil/discretising-continuous-action-spaces.

References

[1] Robert Dadashi, Léonard Hussenot, Damien Vincent, Sertan Girgin, Anton Raichuk, Matthieu Geist, and Olivier Pietquin. Continuous control with action quantization from demonstrations. (arXiv:2110.10149), June 2022. arXiv:2110.10149 [cs].

[2] Emir Demirović, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J. Stuckey. Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research*, 23(26):1–47, 2022.

[3] Emir Demirović, Christian Schilling, and Anna Lukina. Broccoli, January 2025.

[4] Emir Demirović, Christian Schilling, and Anna Lukina. In search of trees: Decision-tree policy synthesis for black-box systems via search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(26):27250–27257, April 2025. arXiv:2409.03260 [cs].

[5] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. (arXiv:1702.08608), March 2017. arXiv:1702.08608 [stat].

[6] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Commun. ACM*, 63(1):68–77, December 2019.

[7] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. (arXiv:1802.09477), October 2018. arXiv:1802.09477 [cs].

[8] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, September 2017. arXiv:1606.08813 [stat].

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

[10] Christian Kroer and Tuomas Sandholm. Discretization of continuous action spaces in extensive-form games. In *Proceedings of the 2015 international conference on autonomous agents and multiagent systems*, pages 47–56, 2015.

[11] Mitja Luštrek, Matjaž Gams, Sanda Martinčić-Ipšić, et al. What makes classification trees comprehensible? *Expert Systems with Applications*, 62:333–346, 2016.

[12] Antonin Raffin. Rl baselines3 zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[13] Daniël Vos and Sicco Verwer. Optimal decision tree policies for markov decision processes. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, page 5457–5465. International Joint Conferences on Artificial Intelligence Organization, August 2023.

[14] Jiaolv Wu, QM Jonathan Wu, Shuyue Chen, Farhad Pourpanah, and Detian Huang. A-td3: An adaptive asynchronous twin delayed deep deterministic for continuous action spaces. *IEEE Access*, 10:128077–128089, 2022.

[15] Jianhao Zhou, Siwu Xue, Yuan Xue, Yuhui Liao, Jun Liu, and Wanzhong Zhao. A novel energy management strategy of hybrid electric vehicle via an improved td3 deep reinforcement learning. *Energy*, 224:120118, 2021.