Optimal Decision Tree Policies for Markov Decision Processes

Vos, Daniël; Verwer, Sicco

Publication date
2023

Document Version
Final published version

Published in
Proceedings of the 32nd International Joint Conference on Artificial Intelligence, IJCAI 2023

Optimal Decision Tree Policies for Markov Decision Processes

Daniël Vos , Sicco Verwer
Delft University of Technology
{d.a.vos, s.e.verwer}@tudelft.nl

Abstract

Interpretability of reinforcement learning policies is essential for many real-world tasks but learning such interpretable policies is a hard problem. Particularly rule-based policies such as decision trees and rules lists are difficult to optimize due to their non-differentiability. While existing techniques can learn verifiable decision tree policies there is no guarantee that the learners generate a decision that performs optimally. In this work, we study the optimization of size-limited decision trees for Markov Decision Processes (MPDs) and propose OMDTs: Optimal MDP Decision Trees. Given a user-defined size limit and MDP formulation OMDT directly maximizes the expected discounted return for the decision tree using Mixed-Integer Linear Programming. By training optimal decision tree policies for different MDPs we empirically study the optimality gap for existing imitation learning techniques and find that they perform sub-optimally. We show that this is due to an inherent shortcoming of imitation learning, namely that complex policies cannot be represented using size-limited trees. In such cases, it is better to directly optimize the tree for expected return. While there is generally a trade-off between the performance and interpretability of machine learning models, we find that OMDTs limited to a depth of 3 often perform close to the optimal limit.

1 Introduction

Advances in reinforcement learning using function approximation have allowed us to train powerful agents for complex problems such as the games of Go and Atari [Schrittwieser *et al.*, 2020]. Policies learned using function approximation often use neural networks, making them impossible for humans to understand. Therefore reinforcement learning is severely limited for applications with high-stakes decisions where the user has to trust the learned policy.

Recent work has focused on *explaining* opaque models such as neural networks by attributing prediction importance to the input features [Ribeiro *et al.*, 2016; Lundberg and Lee, 2017]. However, these explanation methods cannot capture the full complexity of their models, which can mislead users when attempting to understand the predictions [Rudin, 2019]. Concurrently, there has been much work on *interpretable* machine learning in which the model learned is limited in complexity to the extent that humans can understand the complete model. Particularly decision trees have received much attention as they are simple models that are capable of modeling non-linear behavior [Lipton, 2018].

Decision trees are difficult to optimize as they are non-differentiable and discontinuous. Previous works have used different strategies to overcome the hardness of optimizing trees: using assumptions or relaxations to make the trees differentiable [Gupta *et al.*, 2015; Silva *et al.*, 2020; Likmeta *et al.*, 2020], reformulating the MDP into a meta-MDP that exclusively models decision tree policies [Topin *et al.*, 2021] or extracting trees from a complex teacher [Bastani *et al.*, 2018]. While these methods are increasingly successful in training performant trees they do not offer guarantees on this performance.

Our work takes a first step at bridging the gap between the fields of optimal decision trees and reinforcement learning. Existing formulations for optimal decision trees assume a fixed training set with independent samples. This cannot be used in a dynamic setting where actions taken in one state influence the best actions in others. Instead, we formulate the problem of solving a Markov Decision Process (MDP) using a policy represented by a size-limited decision tree (see Figure 1) in a single MILP. We link the predictions of the decision tree policy to the state-action frequencies in the dual linear program for solving MDPs. The dual allows us to reason over policies explicitly, which results in a more efficient formulation. Our formulation for Optimal MDP Decision Trees, OMDTs, optimizes a decision tree policy for a given MDP and a tree size limit. OMDT produces increasingly performant tree policies as runtime progresses and eventually proves the optimality of its policy.

Existing methods for training size-limited decision trees in reinforcement learning such as VIPER [Bastani *et al.*, 2018] make use of imitation learning where a student tries to learn from a powerful teacher policy. We compare the perfomance of OMDT and VIPER on a variety of MDPs. Interestingly, we show that when training interpretable size-limited trees imitation learning performs significantly worse as capacity of the learned decision tree is wasted on parts of the state

space that are never reached by the policy. Moreover, VIPER cannot prove optimality even if it identifies the optimal solution. Regarding the performance-interpretability trade-off we show that decision trees of 7 decision nodes are enough to perform close to optimally in 8 out of 13 environments. Such trees are orders of magnitude smaller than the trees created by methods that exactly replicate the optimal policy using size-unrestricted trees such as dtcontrol [Ashok *et al.*, 2020].

2 Background

2.1 Decision Trees

Decision trees [Breiman *et al.*, 1984; Quinlan, 1986] are simple models that execute a series of comparisons between a feature value and a threshold in the nodes to arrive at a leaf node that contains the prediction value. Due to their simple descriptions, size-limited decision trees are easy to understand for humans [Molnar, 2020]. Particularly, size-limited decision trees admit *simulatability* [Lipton, 2018]: humans can use the model to make predictions by hand in reasonable time and *decomposability*: humans can understand each individual aspect of the model. The method proposed in this paper, OMDT, also offers *algorithmic transparency*: we can trust the learning algorithm to produce models that fulfill certain qualities such as global optimality. Therefore decision trees are an attractive model class when interpretable policies are required.

2.2 Markov Decision Processes

Markov Decision Processes (MDPs) [Bellman, 1957] are the processes underlying reinforcement learning problems. An MDP models a decision-making process in a stochastic environment where the agent has some control over the state transitions. An MDP can be described by a tuple $\langle S, A, P, R \rangle$ where S contains all states, A the set of actions an agent can take, $P_{s,s',a}$ the probabilities of transitioning from state s to state s' when taking action a, and $R_{s,s',a}$ the reward the agent receives when going from state s to state s' under action a. When solving an MDP we want to find a policy $\pi : S \to A$ such that when executing its actions the expected sum of rewards (the return) is maximized. In this work we define policies (w.l.o.g.) as a mapping from states and actions to an indicator for whether or not action a is taken in state s: $\pi : S \times A \to \{0, 1\}$.

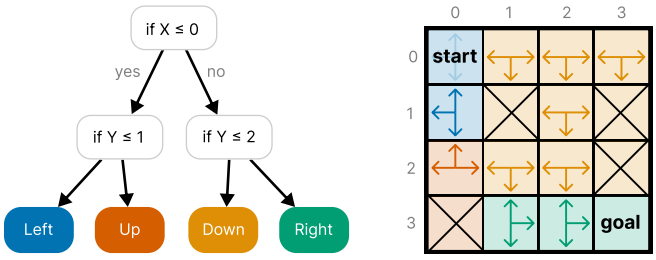


Figure 1: Depth 2 OMDT on the stochastic Frozenlake 4x4 environment. OMDT proves that no better depth 2 decision tree policy exists (discounted return 0.37 with $\gamma = 0.99$).

Value Iteration

When solving MDPs we generally discount future rewards in each step by a user-defined value of $0 < \gamma < 1$ to ensure that the optimal policy will generate a finite return. The most common approach for optimally solving MDPs is by using one of many dynamic programming variants. In this work, we focus on value iteration. Value iteration finds a value V_s for each state s that holds the expected discounted return for taking optimal greedy actions starting from that state. These values can be found by iteratively updating V_s until the Bellman equation [Bellman, 1957] is approximately satisfied. We will refer to this optimal solution found with value iteration as the unrestricted optimal solution as the computed policy can be arbitrarily complex.

3 Related Work

3.1 Learning Decision Tree Policies

Decision trees have appeared at various parts of the reinforcement learning pipeline [Glanois *et al.*, 2021] for example in modeling the Q-value function or in modeling the policy. In this work, we are interested in modeling the policy with a decision tree as this gives us an interpretable model that we can directly use at test time.

Decision trees make predictions using hard comparisons between a single feature value and a threshold the learned models can be discontinuous and non-differentiable which makes optimization with gradients challenging. One line of research focuses on overcoming the non-differentiability of decision trees to allow for the use of gradient-based optimization methods. Gupta et al. [Gupta *et al.*, 2015] train decision trees that contain linear models in their leaves that can be optimized with gradients but this results in models that are hard to interpret. Silva et al. [Silva *et al.*, 2020] first relax the constraints that decision nodes select one feature, that leaves predict one value, and that thresholds are hard step functions. Such relaxed trees are differentiable and can be trained with policy gradient methods. By discretizing the relaxed tree they end up with an interpretable model that approximates the relaxed model. However, the relaxed tree can get stuck in local minima and the discretized tree offers no performance guarantees. Likmeta et al. [Likmeta *et al.*, 2020] consider decision tree policies for autonomous driving tasks. To make the decision tree parameters easily optimizable they fix the structure of the tree along with the features used in the decision nodes. By learning a differentiable hyper-policy over decision tree policies they are then able to approximately optimize the models with gradient descent.

With Iterative Bounding MDPs [Topin *et al.*, 2021] the authors reformulate the underlying MDP into one where the agent implicitly learns a decision tree policy. The method can be thought of as a tree agent learning to take actions that gather information and a leaf agent learning to take actions that work well given the gathered information of the tree agent. By reformulating the MDP its decision tree policy can be optimized using differentiable function approximators and gradient-based optimizers.

In a separate line of work the goal is to represent a specific, usually optimal, policy as a decision tree that is unbounded

in size. These techniques have been developed for policies with a single goal state [Brázdil *et al.*, 2015] and as a tool for general controllers [Ashok *et al.*, 2020]: dtcontrol.

Imitation Learning (VIPER)

Instead of directly optimizing a decision tree one can also try to extract a decision tree policy from a more complex teacher policy using imitation learning. These imitation learning algorithms turn reinforcement learning into a supervised learning problem for which we have successful decision tree learning algorithms [Breiman *et al.*, 1984; Quinlan, 1986]. DAGGER [Ross *et al.*, 2011] (dataset aggregation) is an algorithm that iteratively collects traces from the environment using its current policy and trains a supervised model on the union of the current and previous traces. Since DAGGER only uses information on the predicted action of the teacher policy it ignores extra information on Q-values that modern Q-learning algorithms provide. VIPER [Bastani *et al.*, 2018] focuses on learning decision trees and improves on DAGGER by including Q-value information into the supervised learning objective. While VIPER generates significantly smaller decision trees than DAGGER we will show that these trees are not yet optimal with respect to the trade-off in size and performance.

3.2 Optimal Decision Trees

The standard algorithms for training decision trees in supervised learning are greedy heuristics and can learn trees that perform arbitrarily poorly [Kearns, 1996]. Therefore in recent years there has been increasing interest in the design of algorithms that train decision trees to perform optimally. Early works formulated training decision trees for classification and regression and used methods such as dynamic programming to find optimal decision trees [Nijssen and Fromont, 2007]. Mixed-Integer Linear Programming based formulations [Bertsimas and Dunn, 2017; Verwer and Zhang, 2017] have since become popular. These methods are flexible and have been extended to optimize performance under fairness constraints [Aghaei *et al.*, 2019] or directly optimize adversarial robustness [Vos and Verwer, 2022]. Generally, the size of the tree is limited to provide regularization and aid interpretability, then the solver is tasked with finding a decision tree that maximizes training performance given the size limits. The field has since worked on increasingly efficient optimization techniques using a variety of methods such as MILP [Verwer and Zhang, 2019], dynamic programming [Demirović *et al.*, 2020; Lin *et al.*, 2020], constraint programming [Verhaeghe *et al.*, 2020], branch-and-bound search [Aglin *et al.*, 2020; Aglin *et al.*, 2021] and boolean (maximum) satisfiability [Narodytska *et al.*, 2018; Hu *et al.*, 2020; Schidler and Szeider, 2021].

4 OMDT: Optimal MDP Decision Trees

In an attempt to bridge the gap between optimal decision trees for supervised and reinforcement learning we introduce OMDTs: Optimal MDP Decision Trees. OMDT is a Mixed-Integer Linear Programming formulation that encodes the problem of identifying a decision tree policy that achieves maximum discounted return given a user-defined MDP and size limit. Our formulation can be solved using one of many available solvers, in this work we use the state-of-the-art solver Gurobi[1].

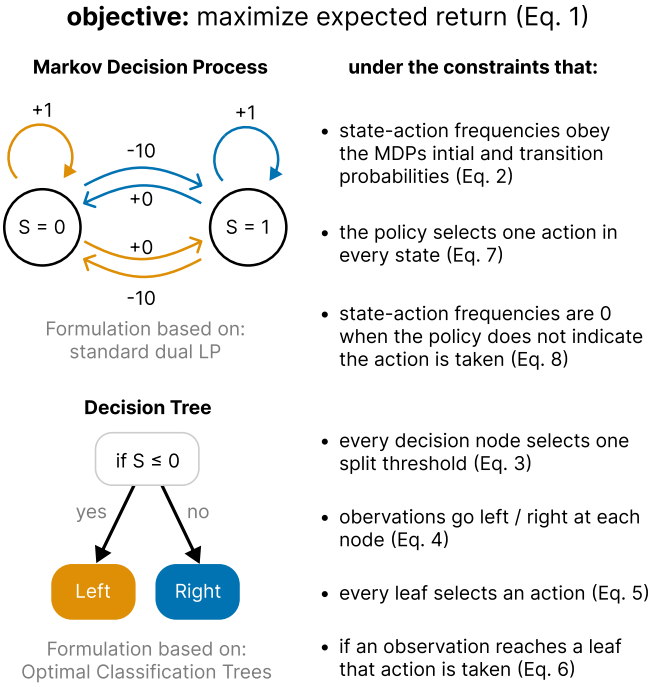


Figure 2: Overview of OMDT's formulation. We maximize the discounted return in an MDP under the constraint that the policy is represented by a size-limited decision tree.

Intuitively the OMDT formulation consists of two parts: a (dual) linear programming formulation for solving MDPs and a set of constraints that limits the set of feasible policies to decision trees. Figure 2 summarizes OMDT's formulation in natural language. All the notation used in OMDT is summarized in Table 1.

4.1 Constraints

It is well known that MDPs can be solved using linear programming, the standard linear program is [Puterman, 2014]:

$$\text{min.} \quad \sum_s p_0(s) V_s$$
$$\text{s.t.} \quad V_s - \sum_{s'} \gamma P_{s,s',a} V_{s'} \geq \sum_{s'} R_{s,s',a}, \quad \forall s, a$$

It is not easy to efficiently add constraints to this formulation to enforce the policies to be size-limited decision trees because it reasons abstractly over policies, i.e. by reasoning over the policy's state values. To create a formulation for decision tree policies we resort to the standard dual program:

$$\text{max.} \quad \sum_s \sum_a x_{s,a} \sum_{s'} P_{s,s',a} R_{s,s',a} \tag{1}$$
$$\text{s.t.} \quad \sum_a x_{s,a} - \sum_{s'} \sum_a \gamma P_{s',s,a} x_{s',a} = p_0(s), \quad \forall s \tag{2}$$

[1] https://www.gurobi.com/

This program uses a measure $x_{s,a}$ of how often the agent takes action a in state s, this allows us to add efficient constraints that control the policy of the agent. Intuitively the program maximizes the rewards $\sum_{s'} R_{s,s',a}$ weighted by this $x_{s,a}$. The constraints enforce that the frequency by which a state is exited is equal to the frequency that the agent is initialized in the state or returns to it following the discounted transition probabilities $\gamma P_{s,s',a}$.

To enforce the policy to be a size-limited decision tree we will later constrain the $x_{s,a}$ values to only be non-zero when the agent is supposed to take action a in state s according to a tree policy. We will first introduce the variables and constraints required to model the decision tree constraints.

Modeling Decision Nodes

Our decision tree formulation is roughly based on OCT [Bertsimas and Dunn, 2017] and ROCT [Vos and Verwer, 2022], MILP formulations for optimal (OCT) and robust (ROCT) classification trees. In these formulations, the shape of the decision tree is fixed. Like ROCT, we describe a decision node m by binary threshold variables $b_{m,j,k}$, indicating whether the kth threshold of feature j is chosen.[2] Unlike ROCT, we only allow one of these variables to be true over all features and possible thresholds:

$$\sum_j \sum_k b_{m,j,k} = 1, \quad \forall m \tag{3}$$

We follow paths through the tree to map observations to leaves. In each node m we decide the direction $d_{s,m}$ that the observation of state s takes (left=0 or right=1 of the threshold k). ROCT uses two variables per state-node pair to model directions $d_{s,m}$ to account for perturbations in the observations. Since we are optimizing for an MDP without uncertainty in the observations we only require one variable $d_{s,m}$ per state-node pair.

We further improve over ROCT by determining $d_{s,m}$ using only one constraint per state-node pair instead of a separate constraint per state-node-feature triple. For this, we pre-compute a function side(s, j, k) which indicates for each feature-threshold pair (j, k) and every observation s the side of k that s is on for feature j (left=0 or right=1), i.e. whether $X_{sj} > k$ holds. This formulation is not limited to the predicates '$\leq$' or '$>$' however and can be easily extended to other predicates in the pre-computation of side(s, j, k). The following then forces the direction $d_{s,m}$ to be equal to the direction of the indicated threshold:

$$d_{s,m} = \sum_j \sum_k \text{side}(s, j, k)\ b_{m,j,k}, \quad \forall s, m \tag{4}$$

The variables $d_{s,m}$ represent the direction of an observation's path at a decision node. Together, the d variables allow us to follow an observation's path through the tree which we use to identify the leaf that it reaches. Important in this formulation, compared to existing binary encodings, is that it requires no big-M constraints to describe these paths. This makes the relaxation stronger and therefore the solver give much better bounds than using the big-M style formulations from ROCT.

[2] In practice, the possible values for threshold k depends on the chosen feature j. We do not model this dependence for convenience of notation.

Modeling Policy Actions

Decision leaves only have one set of binary decision variables: $c_{t,a}$ encoding whether or not leaf t predicts action a. We want each leaf to select exactly one action:

$$\sum_a c_{t,a} = 1, \quad \forall t \tag{5}$$

As mentioned before we can follow an observation's path through the tree by using their $d_{s,m}$ path variables. One can linearize an implication of a conjunction of binary variables as follows:

$$x_1 \wedge x_2 \wedge ... \wedge x_n \implies y$$
$$\equiv x_1 + x_2 + ... + x_n - n + 1 \leq y$$

If an observation reaches leaf t and the leaf predicts action a then we want to force the policy $\pi_{s,a}$ to take that action in the associated state s. Using the aforementioned equivalence we add the constraint:

$$\sum_{m \in A_l(t)} (1 - d_{s,m}) + \sum_{m \in A_r(t)} d_{s,m} + c_{t,a} - |A(t)| \leq \pi_{s,a}, \quad \forall s, a, t \tag{6}$$

This constraint forces the agent to take the action indicated by the leaf. To prevent the agent from taking other actions that were not indicated we force it to only take a single action in each state (giving a deterministic policy):

$$\sum_a \pi_{s,a} = 1, \quad \forall s \tag{7}$$

Now we have indicators $\pi_{s,a}$ that mark what action is taken by the agent. To link this back to the MDP linear programming formulation that we use to optimize the policy, we set the $x_{s,a}$ variables. We need to set $x_{s,a} = 0$ if $\pi_{s,a} = 0$, else $x_{s,a}$ should be unbounded. We encode this using a big-M formulation:

$$x_{s,a} \leq M \pi_{s,a}, \quad \forall s, a \tag{8}$$

M should be chosen as small as possible, but larger or equal to the largest value that $x_{s,a}$ can take. We use the fact that we are optimizing the MDP using discount factor γ to compute an upper bound on $x_{s,a}$ and set $M = \frac{1}{1-\gamma}$, proof is given in the appendix.

4.2 Complete Formulation

The runtime of MILP solvers grows worst-case exponentially with respect to formulation size therefore it is important to limit the scale of the formulation. The number of variables in our formulation grows with $\mathcal{O}(|S||J||\mathcal{T}_D| + |A||\mathcal{T}_L| + |S||A|)$ which follows from their indices in Table 1. The number of constraints grows with the order $\mathcal{O}(|S||\mathcal{T}_D| + |S||A||\mathcal{T}_L|)$ as it is dominated by the constraints that determine $d_{s,m}$ at each node (Equation 4) and constraints that force $\pi_{s,a}$ according to the tree (Equation 6). OMDT's complete MILP formulation is summarized below:

$$\max \quad \sum_s \sum_a x_{s,a} \sum_{s'} P_{s,s',a} R_{s,s',a} \tag{1}$$

Name	Kind	Description
$b_{m,j,k}$	bin.	Tree uses feat. j and threshold k in node m
$c_{t,a}$	bin.	Tree selects action a in leaf t
$d_{s,m}$	bin.	Observation of s goes left / right in node m
$\pi_{s,a}$	bin.	Policy takes action a in state s
$x_{s,a}$	cont.	Frequency of action a taken in state s
$P_{s,s',a}$	const.	Probability of transition $s \to s'$ with action a
$R_{s,s',a}$	const.	Reward for transition $s \to s'$ with action a
$p_0(s)$	const.	Probability of starting in state s
γ	const.	Discount factor
X_{ij}	const.	Feature j's value of observation i
side(s,j,k)	const.	Side state s is on for thresh. k and feat. j
$a \in A$	set	Set of actions in MDP
$s \in S$	set	Set of states in MDP
i=1..$\|S\|$	set	Observation and state indices
$j \in J$	set	Set of feature indices
$k = 1..K$	set	Indices of all possible feature thresholds
$m \in \mathcal{T}_D$	set	Set of decision nodes in the tree
$t \in \mathcal{T}_L$	set	Set of leaves in the tree
$A(t)$	set	Set of ancestors of leaf t
$A_l(t)$	set	... that have t in their left path
$A_r(t)$	set	... that have t in their right path

Table 1: Summary of notation used in OMDT.

s.t.

$$\sum_a x_{s,a} - \sum_{s'} \sum_a \gamma P_{s',s,a} x_{s',a} = p_0(s), \quad \forall s \tag{2}$$

$$\sum_j \sum_k b_{m,j,k} = 1, \quad \forall m \tag{3}$$

$$d_{s,m} = \sum_j \sum_k \text{side}(s,j,k)\ b_{m,j,k}, \quad \forall s, m \tag{4}$$

$$\sum_a c_{t,a} = 1, \quad \forall t \tag{5}$$

$$\sum_{m \in A_l(t)} (1 - d_{s,m}) + \sum_{m \in A_r(t)} d_{s,m} + c_{t,a} - |A(t)| \leq \pi_{s,a}, \quad \forall s, a, t \tag{6}$$

$$\sum_a \pi_{s,a} = 1, \quad \forall s \tag{7}$$

$$x_{s,a} \leq M \pi_{s,a}, \quad \forall s, a \tag{8}$$

5 Results

We present experiments comparing the performance of OMDTs with VIPER and dtcontrol. Viper uses imitation learning to extract a size-limited decision tree from a teacher policy and dtcontrol learns an unrestricted tree that exactly copies the teacher's behavior. To provide a fair comparison we have trained VIPER and dtcontrol with an optimal teacher by first solving the MDP with value iteration and then extracting all Q values, both methods ran with default parameters. We also implemented and ran our experiments on interpretable Differentiable Decision Trees [Silva *et al.*, 2020] but excluded these models from our analysis as they did not outperform a random policy. The full code for OMDT and our experiments can be found on GitHub[3]. All of our experiments ran on a Linux machine with 16 Intel Xeon CPU cores and 72 GB of RAM total and used Gurobi 10.0.0 with default parameters. Each method ran on a single CPU core.

5.1 Environments

For comparison we implemented 13 environments based on well known MDPs from the literature, the sizes of these MDPs are given in Table 2. All MDPs were pre-processed such that states that are unreachable from the initial states are removed. We briefly describe the environments below but refer to the appendix for complete descriptions.

In *3d_navigation* the agent controls a robots in a 5x5x5 world and attempts to reach from one corner to the other end with each voxel having a chance to make the robot disappear. *blackjack* is a simplified version of the famous casino game where we assume an infinite sized-deck and only the actions 'skip' or 'hit'. *frozenlake* is a grid world where the agent attempts to go from the start state to the end state without falling into holes, actions are stochastic so the agent will not always move in the intended direction (e.g. the action 'up' will only surely not send the agent 'down'). *inventory management* models a company with limited inventory size that has decide how many items to order to meet its customer's demand while minimizing cost. *system_administrator* refers to a computer network where computers randomly crash and an administrator has to decide which computer to spend time on rebooting, a crashed computer has an increased probability of crashing a neighboring computer. *tictactoe_vs_random* is the well known game of tic-tac-toe when played against an opponent that makes random moves. In *tiger_vs_antelope* the tiger attempts to catch the antelope that randomly jumps away from the tiger in a grid world. *traffic_intersection* describes a perpendicular intersection where traffic flows in at different rates and the operator decides when to switch the traffic lights. *xor* is an MDP constructed such that the states randomly lie on a plain, the agent gets 1 reward for taking the action according to an XOR function and -1 for a mistake. The XOR problem is notoriously difficult for greedy decision tree learning algorithms.

5.2 Performance-Interpretability Trade-off

It is often assumed that there is a trade-off in the performance and interpretability of machine learning models [Gunning and Aha, 2019] since interpretable models necessarily lack complexity but this assumption is not always true [Rudin, 2019]. We aim to answer whether the performance-interpretability trade-off occurs in a variety of MDPs by training size-limited decision trees and comparing their performance to the optimal solutions that were not restricted in complexity. We visualize the normalized return of depth 3 OMDTs and unrestricted dtcontrol trees in Figure 3. Returns were normalized such that 0 corresponds to the return of a random policy and 1 to an optimal one. Since small deterministic decision tree policies are limited in the number of distinct actions that they take an optimal tree can perform worse than a random policy. Experiments were repeated 3 times and runs were limited to 2

[3]https://github.com/tudelft-cda-lab/OMDT

MDP	$\|S\|$	$\|A\|$	normalized return VIPER	OMDT 5 mins.	OMDT 2 hrs.	MILP vars.	constrs.	trees	runtime (s) VIPER	OMDT optimal
3d_navigation	125	6	**1.00** ±.00	.81 ±.10	**1.00** ±.00	2,528	7,890	10^{14}	2,090 ±55	315 ±89
blackjack	533	2	**1.00** ±.00	**1.00** ±.00	**1.00** ±.00	6,187	14,406	10^{14}	2,248 ±27	408 ±85
frozenlake_4x4	16	4	.67 ±.00	**.96** ±.00	**.96** ±.00	328	735	10^{10}	74 ±3	2 ±0
frozenlake_8x8	64	4	.83 ±.06	**.95** ±.00	**.95** ±.00	1,104	2,895	10^{13}	178 ±5	98 ±30
frozenlake_12x12	144	4	.19 ±.09	.63 ±.03	**.68** ±.04	2,360	6,495	10^{14}	196 ±38	timeout
inv. management	101	100	**1.00** ±.00	.37 ±.37	**1.00** ±.00	22,414	91,824	10^{30}	2,254 ±86	2,533 ±540
sysadmin_1	256	9	.88 ±.01	.85 ±.01	**.92** ±.00	6,584	23,055	10^{14}	2,265 ±37	timeout
sysadmin_2	256	9	**.59** ±.00	.23 ±.06	.58 ±.01	6,584	23,055	10^{14}	2,257 ±7	timeout
sysadmin_tree	128	8	.57 ±.04	.48 ±.07	**.70** ±.09	3,106	10,383	10^{13}	2,136 ±72	timeout
tictactoe_vs_random	2,424	9	**.80** ±.01	-.06 ±.00	.43 ±.18	61,239	218,175	10^{20}	21 ±3	timeout
tiger_vs_antelope	626	5	-.10 ±.02	-.17 ±.19	**.52** ±.03	10,850	33,819	10^{15}	490 ±243	timeout
traffic_intersection	361	2	.98 ±.00	.99 ±.00	**1.00** ±.00	4,127	9,762	10^{11}	2,188 ±121	1,219 ±177
xor	200	2	.34 ±.06	**1.00** ±.00	**1.00** ±.00	5,016	5,415	10^{21}	1,999 ±123	50 ±0

Table 2: Comparison of depth 3 trees trained with VIPER and OMDT on 13 MDPs, experiments were repeated 3 times and runs were limited to 2 hours of runtime. OMDT solves some MDPs in 5 minutes but significantly improves when given 2 hours of runtime. While 2 hours are enough for OMDT to achieve greater or equal scores to VIPER in most MDPs, OMDT needs more runtime to outperform VIPER on the large tictactoe instance. OMDT was able to identify the optimal size-limited tree and prove its optimality in 7 instances.

hours. We consider an OMDT optimal when the relative gap between its objective and bound is proven to be smaller than 0.01%.

While it is debatable what the precise size limits are for decision trees to be interpretable [Lipton, 2018] we use trees of depth 3 which implies that a tree has at most 8 leaves. We find that in all environments, OMDTs of depth 3 improve on the performance of random policies, and in 8 out of 13 environments the policy gets close to optimal. Decision trees trained with dtcontrol always achieve the optimal normalized return of 1 since they exactly mimic the optimal policy. However, dtcontrol produces large trees that are not interpretable to humans. When run on 3d_navigation for example, dtcontrol produces a tree of 68 decision nodes which is very complex for humans to understand. OMDT produces a tree of 7 decision nodes which performs equally well.

Overall, our results demonstrate that for many environments there is no performance-interpretability trade-off: simple policies represented by size-limited trees perform approximately as well as the unrestricted optimal policy.

5.3 Direct Optimization versus Imitation Learning

The above conclusion holds when the policy is learned to optimality under the constraint that it has to be a small tree, e.g., using OMDT. Frequently used techniques such as VIPER do not directly enforce this constraint but aim to imitate the unrestricted optimal policy. We now show that this comes at a cost when the unrestricted policy is too complex to be represented using a small tree.

VIPER trains its decision trees by imitating high Q values of the optimal policies while OMDT directly maximizes expected return. In Table 2 we list the normalized return (0 for random policies, 1 for optimal policies) for VIPER and OMDT with respectively 5 minutes and 2 hours of runtime. After 5 minutes OMDT improves performance over random

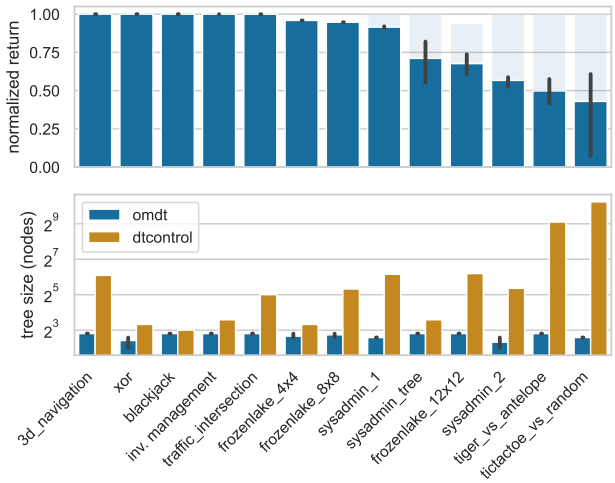
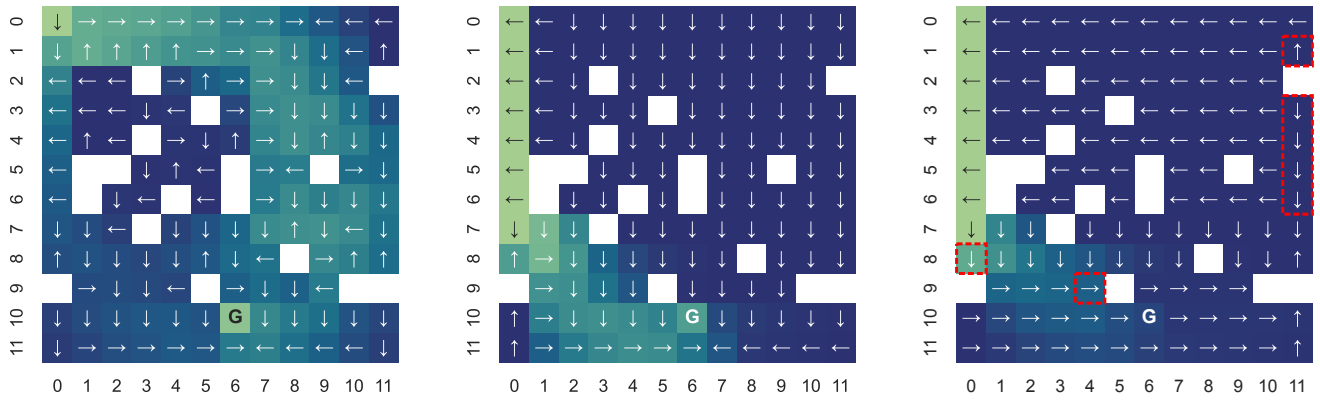


Figure 3: (top) Normalized return and bounds for OMDT trees of depth 3, optimal policies score 1 while uniform random policies score 0. (bottom) Log of tree sizes for OMDT (maximum depth 3) and dtcontrol. Dtcontrol always produces an optimal policy but the trees are orders of magnitude larger than OMDT.

policies but often needs more time to improve over VIPER. After 2 hours OMDT's policies win on 11 out 13 environments. For instances with large state space such as tictactoe_vs_random OMDT needs more than 2 hours to improve over VIPER.

Shortcomings of Imitation Learning

Overall, given sufficient run-time, OMDT produces better policies than VIPER. This cannot be easily solved by giving VIPER more run-time but is an inherent problem of imitation learning. To illustrate this, we investigate the results on the



(a) Optimal: 92% success (b) OMDT (depth 3): 66% success (c) VIPER (depth 3): 11% success

Figure 4: Paths taken on 10,000 Frozenlake_12x12 runs. The agent starts at (0, 0) and attempts to reach the goal tile 'G' while avoiding holes. Actions are indicated by arrows and are somewhat stochastic, i.e. an action of 'up' will send the agent 'left', 'up' or 'right' (but never down) with equal probability. VIPER fails to produce a good policy because it spends capacity of its tree mimicking parts of the complex teacher policy that its simple student policy will never reach. OMDT achieves a greater success rate by directly optimizing a simple policy.

frozenlake environments. Table 2 demonstrates that imitation learning (VIPER) can perform far from optimal on these environments, while direct optimization (OMDT) performs closer to the unrestricted optimal solution. It is insightful to understand why this happens.

In theory, imitation learning performs optimally in the limit [Ross *et al.*, 2011] but this result requires the student policy to be as expressive as the teacher policy. This is not the case for size-limited decision trees. When VIPER learns a policy for frozenlake_12x12 it tries to imitate a complex policy using a small tree that cannot represent all teacher policy actions. This results in VIPER spending capacity of its decision tree on parts of the state space that will never be reached under its student policy. In Figure 4 we visualize the paths that the agents took on 10,000 runs and indicate the policies with arrows. VIPER creates leaves that control action in the right section of the grid world (indicated in red). The optimal teacher policy often visits this section but the simple student does not. By directly optimizing a decision tree policy using OMDT, the policy spends its capacity on parts of the state space that it actually reaches. As a result, VIPER cannot prevent actions that send its agent into holes on the left part of the grid world (indicated in red). OMDT actively avoids these.

5.4 Runtime

Runtime for solving Mixed-Integer Linear Programming formulations scales worst-case exponentially which makes it important to understand how solvers operate on complex formulations such as OMDT. We solved OMDTs for a depth of 3 for a maximum of 2 hours and display the results in Table 2. The table compares the runtimes of VIPER and solving OMDT to optimality. If the solver does not prove optimality within 2 hours we denote it as 'timeout'. We also denote the number of possible decision tree policies computed as: $|\mathcal{T}_B|^{\text{possible splits}} \times |\mathcal{T}_L|^{|A|}$. It provides an estimate of how many decision tree policies are possible and shows that enumerating trees with brute force is intractable.

OMDT solves a simple environment such as Frozenlake 4x4 (16 states, 4 actions) to optimality within 2 seconds but runtime grows for larger environments such as inventory management (101 states, 100 actions) which took on average 2533 seconds. VIPER needs roughly 2250 seconds of runtime for every MDP and runs significantly faster on some MDPs. This is because VIPER spends much time evaluating policies on the environment and some environments quickly reach terminal states which results in short episodes. While OMDT was able to prove optimality on only 7 out of 13 environments within 2 hours, OMDT finds good policies before this time on 12 out of 13 environments.

6 Conclusion

We propose OMDT, a Mixed-Integer Linear Programming formulation for training optimal size-limited decision trees for Markov Decision Processes. Our results show that for simple environments such as blackjack we do not have to trade off interpretability for performance: OMDTs of depth 3 achieve near-optimal performance. On Frozenlake 12x12, OMDT outperforms VIPER by more than 100%.

OMDT sets a foundation for extending supervised optimal decision tree learning techniques for to the reinforcement learning setting. Still, OMDT requires a full specification of the Markov Decision Process. Imitation learning techniques such as VIPER can instead also learn from a simulation environment. Therefore, future work should focus on closing the gap between the theoretical bound supplied by OMDT and the practical performance achieved by algorithms that require only simulation access to optimize interpretable decision tree policies in reinforcement learning. Additionally, future work can incorporate factored MDPs into OMDT's formulation to scale up to larger state spaces.

References

[Aghaei *et al.*, 2019] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning optimal and fair decision trees for non-discriminative decision-making. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1418–1426, 2019.

[Aglin *et al.*, 2020] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3146–3153, 2020.

[Aglin *et al.*, 2021] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Pydl8. 5: a library for learning optimal decision trees. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 5222–5224, 2021.

[Ashok *et al.*, 2020] Pranav Ashok, Mathias Jackermeier, Pushpak Jagtap, Jan Křetínskỳ, Maximilian Weininger, and Majid Zamani. dtcontrol: Decision tree learning algorithms for controller representation. In *Proceedings of the 23rd international conference on hybrid systems: Computation and control*, pages 1–7, 2020.

[Bastani *et al.*, 2018] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018.

[Bellman, 1957] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[Bertsimas and Dunn, 2017] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

[Brázdil *et al.*, 2015] Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Andreas Fellner, and Jan Křetínskỳ. Counterexample explanation by learning small strategies in markov decision processes. In *International Conference on Computer Aided Verification*, pages 158–177. Springer, 2015.

[Breiman *et al.*, 1984] L Breiman, JH Friedman, R Olshen, and CJ Stone. Classification and regression trees. 1984.

[Demirović *et al.*, 2020] Emir Demirović, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J Stuckey. Murtree: optimal classification trees via dynamic programming and search. *arXiv preprint arXiv:2007.12652*, 2020.

[Glanois *et al.*, 2021] Claire Glanois, Paul Weng, Matthieu Zimmer, Dong Li, Tianpei Yang, Jianye Hao, and Wulong Liu. A survey on interpretable reinforcement learning. *arXiv preprint arXiv:2112.13112*, 2021.

[Gunning and Aha, 2019] David Gunning and David Aha. Darpa's explainable artificial intelligence (xai) program. *AI magazine*, 40(2):44–58, 2019.

[Gupta *et al.*, 2015] Ujjwal Das Gupta, Erik Talvitie, and Michael Bowling. Policy tree: Adaptive representation for policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[Hu *et al.*, 2020] Hao Hu, Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. Learning optimal decision trees with maxsat and its integration in adaboost. In *IJCAI-PRICAI 2020, 29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence*, 2020.

[Kearns, 1996] Michael Kearns. Boosting theory towards practice: Recent developments in decision tree induction and the weak learning framework. In *Proceedings of the national conference on artificial intelligence*, pages 1337–1339, 1996.

[Likmeta *et al.*, 2020] Amarildo Likmeta, Alberto Maria Metelli, Andrea Tirinzoni, Riccardo Giol, Marcello Restelli, and Danilo Romano. Combining reinforcement learning with rule-based controllers for transparent and general decision-making in autonomous driving. *Robotics and Autonomous Systems*, 131:103568, 2020.

[Lin *et al.*, 2020] Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning*, pages 6150–6160. PMLR, 2020.

[Lipton, 2018] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[Lundberg and Lee, 2017] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[Molnar, 2020] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[Narodytska *et al.*, 2018] Nina Narodytska, Alexey Ignatiev, Filipe Pereira, Joao Marques-Silva, and I Ras. Learning optimal decision trees with sat. In *Ijcai*, pages 1362–1368, 2018.

[Nijssen and Fromont, 2007] Siegfried Nijssen and Elisa Fromont. Mining optimal decision trees from itemset lattices. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 530–539, 2007.

[Puterman, 2014] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[Ribeiro *et al.*, 2016] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[Ross *et al.*, 2011] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

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

[Schidler and Szeider, 2021] André Schidler and Stefan Szeider. Sat-based decision tree learning for large data sets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3904–3912, 2021.

[Schrittwieser *et al.*, 2020] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[Silva *et al.*, 2020] Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *International conference on artificial intelligence and statistics*, pages 1855–1865. PMLR, 2020.

[Tange, 2021] Ole Tange. Gnu parallel 20210622 ('protasevich'), June 2021. GNU Parallel is a general parallelizer to run multiple serial command line programs in parallel without changing them.

[Topin *et al.*, 2021] Nicholay Topin, Stephanie Milani, Fei Fang, and Manuela Veloso. Iterative bounding mdps: Learning interpretable policies via non-interpretable methods. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9923–9931, 2021.

[Verhaeghe *et al.*, 2020] Hélene Verhaeghe, Siegfried Nijssen, Gilles Pesant, Claude-Guy Quimper, and Pierre Schaus. Learning optimal decision trees using constraint programming. *Constraints*, 25(3):226–250, 2020.

[Verwer and Zhang, 2017] Sicco Verwer and Yingqian Zhang. Learning decision trees with flexible constraints and objectives using integer optimization. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 94–103. Springer, 2017.

[Verwer and Zhang, 2019] Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1625–1632, 2019.

[Vos and Verwer, 2022] Daniël Vos and Sicco Verwer. Robust optimal classification trees against adversarial examples. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8520–8528, 2022.

A Environment Descriptions

We describe the environments in additional detail below. All environments in the paper were solved with a discount rate of $\gamma = 0.99$. Such a value ensures that the solvers generate policies that have good performance even for environments that takes hundreds of steps to solve. All MDPs were pre-processed such that states that are unreachable from the initial states are removed.

A.1 3d_navigation

While policies in 2d scenarios are often easy to visualize and interpret policies in higher dimensions are harder to understand. Therefore we extend the 2d_navigation problem from ICAP 2011 IPPC competition[4] to 3 dimensions. The agent controls a robots in a 5x5x5 world and attempts to reach from one corner to the other end with each voxel having a chance to make the robot disappear. The disappearing chances were independently sampled from a uniform distribution on [0, 1].

A.2 blackjack

The version of blackjack that we used[5] is a slightly simplified version of the one that often appears in casinos. Specifically, we make the following assumptions:

- There is no doubling down and splitting action.
- The player does not have knowledge about other players at the board.
- Aces always account for a value of 11.
- Cards are drawn from an infinitely sized deck, i.e. drawn with replacement.

The player always starts in the state where no cards are dealt, the dealer receives a random card, followed by the player receiving a random card. The player can 'Draw' cards until they cross the total of 21 or until they 'Skip'. When that happens the dealer receives cards until their total is over 17. In the end, the reward is computed as follows:

- -1 if the player has a lower total than the dealer or if the player went over 21.
- 0 if the player and dealer end up with the same total.
- +1 if the player has a higher total than the dealer or if the dealer went over 21.
- +1.5 if the player gets to exactly 21

A.3 frozenlake

Frozenlake is a maze-like game played on a 2D grid where the player attempts to move from the start state (S) to the goal state (G) without falling into holes (H). The player walks over frozen tiles (F) that make the actions stochastic:

- The action 'Up' sends a player in the directions left, up, or right with probability $1/3$ each.
- The action 'Right' sends a player in the directions up, right, or down with probability $1/3$ each.
- The action 'Down' sends a player in the directions right, down, or left with probability $1/3$ each.
- The action 'Left' sends a player in the directions down, left, or up with probability $1/3$ each.

We used the default 4x4 and 8x8 maps as defined by the OpenAI gym[6] and defined a new map for 12x12. The maps are given below (in the same format that the environment uses):

4x4

```
"SFFF",
"FHFH",
"FFFH",
"HFFG",
```

8x8

```
"SFFFFFFF",
"FFFFFFFF",
"FFFHFFFF",
"FFFFFHFF",
"FFFHFFFF",
"FHHFFFHF",
"FHFFHFHF",
"FFFHFFFG",
```

12x12

```
"SFFFFFFFFFFF",
"FFFFFFFFFFFF",
"FFFHFFFFFFFH",
"FFFFFHFFFFFF",
"FFFHFFFFFFFF",
"FHHFFFHFFHFF",
"FHFFHFHFFFFF",
"FFFHFFFFFFFF",
"FFFFFFFFHFFF",
"HFFFFHFFFFHH",
"FFFFFFGFFFFF",
"FFFFFFFFFFFF",
```

A.4 inventory management

In inventory management, the player is the owner of a shop that stocks items and sells them to customers. Our environment is adapted from the implementation by Paul Hendricks[7]. Every day the player can choose 'Don't buy' or 'Buy x' items. We used the following parameters for the problem:

- The maximum inventory size is 100 (this creates 101 states since the inventory can have 0 up to and including 100 items)
- Ordering any items comes with a fixed cost of -10
- Ordering items costs -2 per item
- Holding an item in storage costs -1 per item
- Selling an item grants the player +4
- The number of customers follows a Poisson distribution with expectation $\lambda = 15$ (customers per day)

A.5 system administrator

The system administrator task refers to a computer network where computers randomly crash and an administrator has to decide which computer to spend time on rebooting. This environment is based on one of the MDPs of the ICAPS 2011 IPPC competition[4]. When a computer crashes it has an increased probability in following time steps to crash a neighboring computer. The topology of the network can be varied to create MDPs with various properties. The 3 topologies considered in this work are visualized in Figure 5.

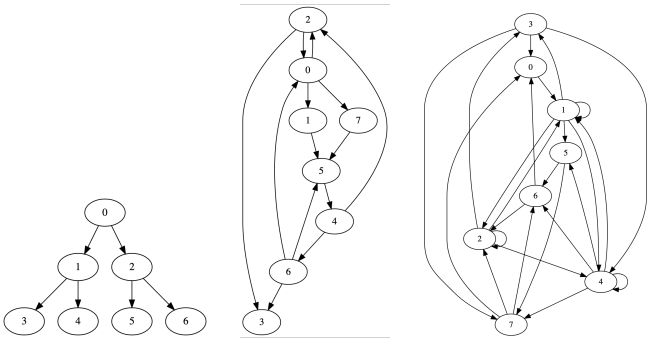
The system administrator is then allowed to reboot one machine at every time step or wait. Rebooting a computer is penalized with a reward of -0.45 but will always fix the crashed computer. At each time step every computer that has

[4] http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/

[5] https://gist.github.com/iiLaurens/ba9c479e71ee4ceef816ad50b87d9ebd

[6] https://github.com/openai/gym/blob/master/gym/envs/toy_text/frozen_lake.py

[7] https://github.com/paulhendricks/gym-inventory/tree/master



(a) tree (b) random 1 (c) random 2

Figure 5: System administrator topologies.

not crashed will reward the system administrator with +1. We refer to a computer being on (1) or crashed (0) as the 'status', the probability of being on in the next step is then given by:

$$\text{ratio\_on\_neighbors} \times (0.05 + 0.9 \times \text{status})$$

A.6 tictactoe_vs_random

The well known tic-tac-toe game is played on a 3x3 grid where players take turns putting a cross / circle in an empty square. The player that manages to put 3 of their symbols in a row / column / diagonal wins the game. While 2 player zero-sum games are not MDPs, they become MDPs when we fix the strategy of one of the players. In this case the opponent plays moves uniformly at random. It is worth noting that this turns the game that is normally deterministic into a stochastic environment. The player gets to start the game, i.e. they play with the crosses.

A.7 tiger_vs_antelope

The tiger vs antelope game is played on a 5x5 grid where the agent controls the tiger and tries to catch an antelope. Every turn the agent can move 'up', 'down', 'left' or 'right' and if the antelope is not caught it responds by jumping in any direction away from the player uniformly at random.

A.8 traffic_intersection

The traffic intersection scenario that we considered describes a perpendicular intersection where traffic flows in at different rates and the operator decides when to switch the traffic lights. At each side of the intersection 5 cars can fit in a row and from the moment the traffic light switches we count up to 5 time steps in which one side of the intersection is waiting. Cars arrive with probability 0.1 per time step on one side of the intersection and 0.5 on the other side.

The operator gets rewarded +1 for each car making it through the intersection and gets penalized for flipping the color of the traffic signs and for waiting cars. Flipping the color of the traffic lights incurs a reward of -2. For each waiting car the traffic operator gets a reward of $-0.1 \times 2^{\text{wait\_time}}$.

A.9 xor

The XOR problem is notoriously difficult for greedy decision tree learning algorithms, we generate an MDP that mimics the structure of the supervised learning problem. Specifically we draw 200 samples in 2 dimensions uniformly at random. The agent gets 1 reward for taking the action a according to an XOR function and -1 for a mistake:

$$R(x, y, a) = \begin{cases} 1 & \text{round}(x) + \text{round}(y) \mod 2 = a \\ -1 & \text{round}(x) + \text{round}(y) \mod 2 = a \end{cases}$$

Each action sends the agent to a state uniformly at random.

B Linearizing Implications of Conjunctions

In the main text, we constructed one of OMDT's constraints using the fact that:

$$x_1 \wedge x_2 \wedge ... \wedge x_n \implies y \equiv x_1 + x_2 + ... + x_n - n + 1 \leq y$$

We give a short proof of this statement. First, we rewrite the logical formula into conjunctive normal form. In this case a single disjunctive clause.

$$\begin{aligned} x_1 \wedge x_2 \wedge ... \wedge x_n &\implies y \\ \equiv \neg(x_1 \wedge x_2 \wedge ... \wedge x_n) &\vee y \\ \equiv \neg x_1 \vee \neg x_2 \vee ... \vee \neg x_n &\vee y \end{aligned}$$

A disjunctive constraint $a \vee b \vee c$ can be trivially expressed as the linear constraint $a + b + c \geq 1$. Intuitively any variable a, b or c needs to be true. Now we rewrite to linear constraints:

$$\begin{aligned} &\neg x_1 \vee \neg x_2 \vee ... \vee \neg x_n \vee y \\ \equiv (1 - x_1) + (1 - x_2) + ... + (1 - x_n) &+ y \geq 1 \\ \equiv x_1 + x_2 + ... + x_n - n + 1 &\leq y \end{aligned}$$

C Big-M Value

For the constraints that force $x_{s,a}$ according to the policy $\pi_{s,a}$ (Equation 8) we used a big-M formulation with $M = \frac{1}{1-\gamma}$ as an upper bound on $x_{s,a}$. To prove this upper bound we construct an MDP in which we maximize $x_{s,a}$ then show that this value equals our bound. Consider an MDP with one state s^* and one action a^* such that the agent always starts in that state ($p_0(s^*) = 1$) and action a^* will always return to s^* with probability $P_{s^*,s^*,a^*} = 1$. Clearly, this maximizes the frequency x_{s^*,a^*}. Substituting into Equation 2 we find:

$$\sum_a x_{s,a} = p_0(s) + \sum_{s'} \sum_a \gamma P_{s',s,a} x_{s',a} \tag{2}$$

$$x_{s^*,a^*} = p_0(s^*) + \gamma x_{s^*,a^*} = \frac{1}{1-\gamma}$$

D Performance at Varying Depths

In the main text we discussed performance of trees at a depth of 3 since these trees are still easily interpretable but more powerful than trees of depth 1 or 2. Normalized returns after 2 hours of runtime for varying depth are presented in Table 3, Table 4 shows the runtime needed to prove optimality of the solutions for varying depths.

E Size-Limited Policies

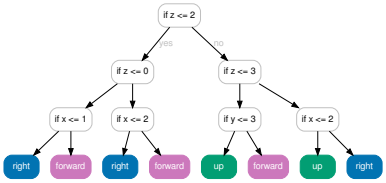
We claim that since our models are interpretable since they are limited in size to an extend that a human can easily follow the predictions of the models [Lipton, 2018]. The decision trees of depth 3 created by OMDT and VIPER are visualized for inspection in Figure 6 and Figure 7 respectively.

MDP	depth 1		depth 2		depth 3		depth 4	
	OMDT	VIPER	OMDT	VIPER	OMDT	VIPER	OMDT	VIPER
3d_navigation	**-.00** ± .00	**-.00** ± .00	**.06** ± .00	.01 ± .00	**1.00** ± .00	**1.00** ± .00	**1.00** ± .00	**1.00** ± .00
blackjack	**.98** ± .00	**.98** ± .00	**1.00** ± .00	.98 ± .00	**1.00** ± .00	**1.00** ± .00	**1.00** ± .00	**1.00** ± .00
frozenlake_4x4	**.19** ± .00	.04 ± .06	**.67** ± .00	.46 ± .00	**.96** ± .00	.67 ± .00	**1.00** ± .00	**1.00** ± .00
frozenlake_8x8	**.74** ± .00	.72 ± .01	**.93** ± .00	.90 ± .02	**.95** ± .00	.83 ± .06	**.98** ± .00	.95 ± .00
frozenlake_12x12	**.06** ± .00	.04 ± .02	**.34** ± .00	.30 ± .00	**.68** ± .04	.19 ± .09	**.81** ± .01	.19 ± .09
inv. management	**.99** ± .00	.98 ± .00	**1.00** ± .00	.99 ± .00	**1.00** ± .00	**1.00** ± .00	**1.00** ± .00	**1.00** ± .00
sysadmin_1	**.84** ± .00	.83 ± .00	**.89** ± .00	.86 ± .03	**.92** ± .00	.88 ± .01	.91 ± .00	**.92** ± .01
sysadmin_2	**.27** ± .00	**.27** ± .00	**.55** ± .00	**.55** ± .00	.57 ± .02	**.59** ± .00	.67 ± .04	**.72** ± .00
sysadmin_tree	**-.03** ± .00	**-.03** ± .00	**.37** ± .00	.26 ± .00	**.71** ± .08	.57 ± .04	**.86** ± .04	**.86** ± .00
tictactoe_vs_random	**-.06** ± .00	**-.06** ± .00	**.61** ± .00	**.61** ± .00	.43 ± .18	**.80** ± .01	.68 ± .10	**.83** ± .00
tiger_vs_antelope	**-.36** ± .00	-.64 ± .06	**.19** ± .05	-.31 ± .17	**.50** ± .05	-.10 ± .02	**.64** ± .03	.23 ± .10
traffic_intersection	**.50** ± .00	.42 ± .00	**.98** ± .00	**.98** ± .00	**1.00** ± .00	.98 ± .00	**1.00** ± .00	**1.00** ± .00
xor	**.15** ± .00	.05 ± .00	**1.00** ± .00	.13 ± .00	**1.00** ± .00	.34 ± .06	**1.00** ± .00	.90 ± .03

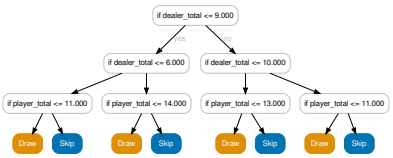
Table 3: Normalized returns of OMDT and VIPER when varying the maximum depth of the decision tree, runtime was limited to 2 hours and experiments repeated with 3 random seeds. For depths until 3 OMDT finds optimal or high quality trees within 2 hours while for depth 4 OMDT will need more runtime to improve on the scores produced by VIPER's trees on 3 environments.

MDP	depth 1	depth 2	depth 3	depth 4
3d_navigation	9 ±0	258 ±27	315 ±89	223 ±25
blackjack	36 ±5	72 ±7	408 ±85	598 ±99
frozenlake_4x4	1 ±0	1 ±0	2 ±0	2 ±0
frozenlake_8x8	2 ±0	9 ±2	98 ±30	3134 ±350
frozenlake_12x12	10 ±2	349 ±57	timeout	timeout
inv. management	453 ±42	6597 ±507	2533 ±540	3548 ±732
sysadmin_1	129 ±10	5059 ±542	timeout	timeout
sysadmin_2	488 ±53	timeout	timeout	timeout
sysadmin_tree	74 ±10	4547 ±767	timeout	timeout
tictactoe_vs_random	2989 ±61	timeout	timeout	timeout
tiger_vs_antelope	1695 ±210	timeout	timeout	timeout
traffic_intersection	33 ±1	104 ±28	1219 ±177	*5595*
xor	113 ±10	43 ±1	50 ±0	98 ±27

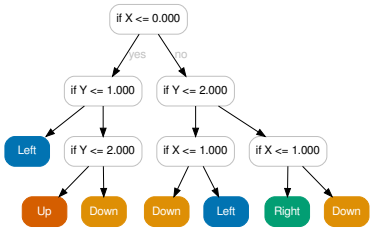
Table 4: Runtimes until OMDT proves optimality when varying the depth of the trees, runtime was limited to 2 hours and experiments repeated with 3 random seeds. Proving optimality for deeper trees usually takes more runtime but since a deeper tree allows the tree's objective value to be closer to the bound runtime can reduce e.g. in *3d_navigation* depth 4. OMDT only proved optimality for *traffic_intersection* depth 4 in one of the runs.



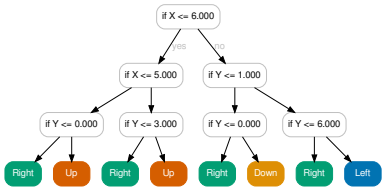
(a) 3d_navigation



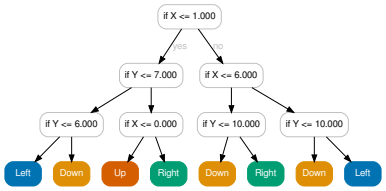
(b) blackjack



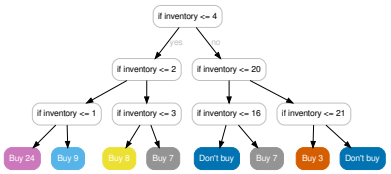
(c) frozenlake_4x4



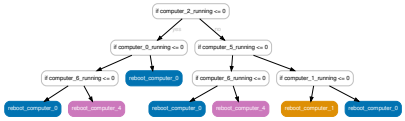
(d) frozenlake_8x8



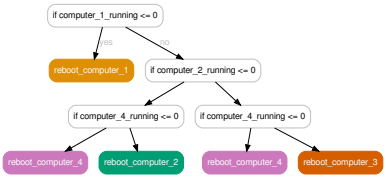
(e) frozenlake_12x12



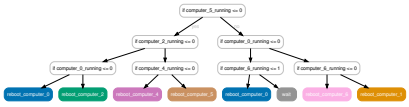
(f) inventory_management



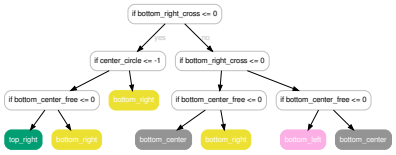
(g) system_administrator_1



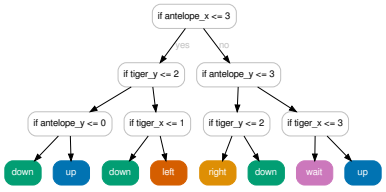
(h) system_administrator_2



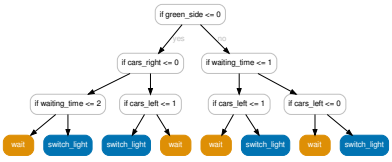
(i) system_administrator_tree



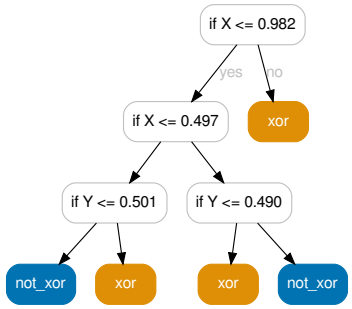
(j) tictactoe_vs_random



(k) tiger_vs_antelope

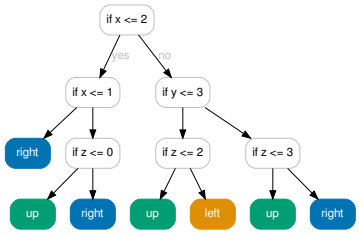


(l) traffic_intersection

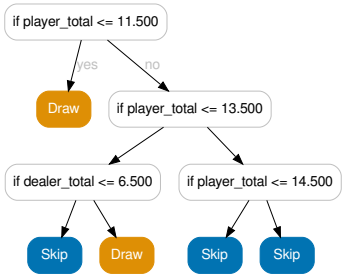


(m) xor

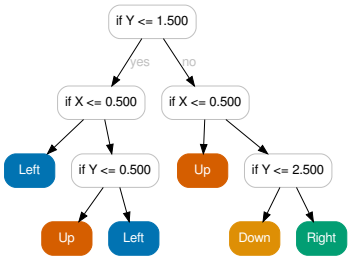
Figure 6: Depth 3 trees produced by OMDT within 2 hours of runtime with one fixed seed.



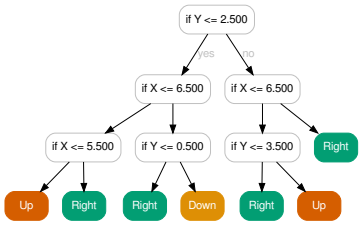
(a) 3d_navigation



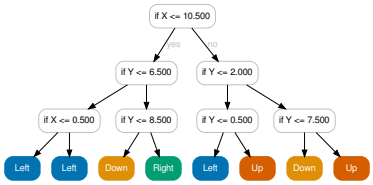
(b) blackjack



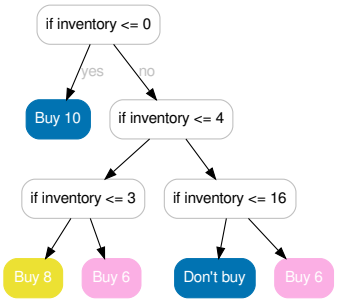
(c) frozenlake_4x4



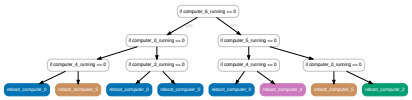
(d) frozenlake_8x8



(e) frozenlake_12x12

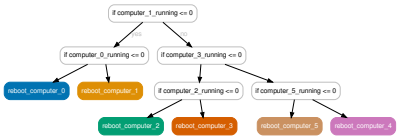


(f) inventory_management

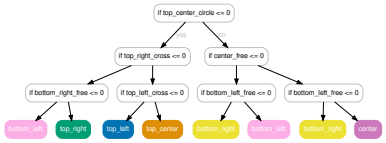


(g) system_administrator_1

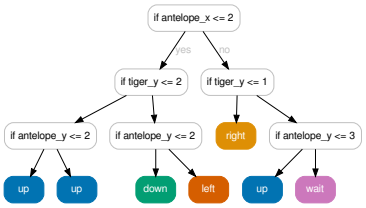
(h) system_administrator_2



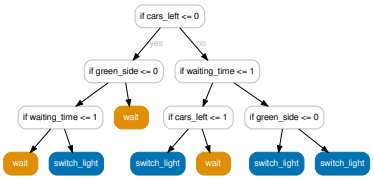
(i) system_administrator_tree



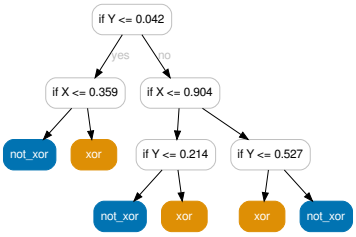
(j) tictactoe_vs_random



(k) tiger_vs_antelope



(l) traffic_intersection



(m) xor

Figure 7: Depth 3 trees produced by VIPER with one fixed seed.