
SPOT: Scalable Policy Optimization with Trees for Markov Decision Processes

Anonymous Author(s)

Affiliation

Address

email

Abstract

Interpretable reinforcement learning policies are essential for high-stakes decision-making, yet optimizing decision tree policies in Markov Decision Processes (MDPs) remains challenging. We propose SPOT, a novel method for computing decision tree policies, which formulates the optimization problem as a mixed-integer linear program (MILP). To enhance efficiency, we employ a space-reduced branch-and-bound approach that decouples the MDP dynamics from tree-structure constraints, enabling efficient parallel search. This significantly improves runtime and scalability compared to previous methods. Our approach ensures that each iteration yields the optimal decision tree. Experimental results on standard benchmarks demonstrate that SPOT achieves substantial speedup and scales to larger MDPs with a significantly higher number of states. The resulting decision tree policies are interpretable and compact, maintaining transparency without compromising performance. These results demonstrate that our approach simultaneously achieves interpretability and scalability, delivering high-quality policies an order of magnitude faster than existing approaches.

1 Introduction

In high-stakes or safety-critical domains, it is crucial that reinforcement learning (RL) policies be understandable by humans. Rather than relying on post-hoc explanations of opaque neural policies (e.g. via LIME or SHAP) which can be incomplete or misleading [25], a more direct approach is to learn inherently interpretable policies. Decision tree policies have attracted significant attention as a suitable interpretable model class: they are simple, rule-based decision structures (threshold tests on state features leading to actions) that can represent non-linear behavior while remaining human-comprehensible. A size-limited decision tree (bounded depth or number of leaves) is simulatable by a person (one can manually follow the decision path) and decomposable (each decision node is an understandable rule).

Optimizing a policy constrained to be a decision tree is notoriously difficult [6]. Unlike differentiable function approximators, decision trees have a discontinuous, non-differentiable structure that precludes straightforward gradient-based training. The space of possible trees grows combinatorially with depth, making brute-force search intractable except for very small cases. In supervised learning, greedy algorithms like CART [8] can find reasonably good trees but are not guaranteed to find the optimal tree and may perform arbitrarily poorly in some cases. In the context of MDPs, an added challenge is that the policy’s decisions in one state can influence the distribution of states encountered elsewhere. This means we cannot simply optimize the tree on a static dataset of state-action examples; we must consider the global dynamics of the MDP when evaluating a tree policy. Overall, finding an optimal or near-optimal decision tree policy in an MDP is a combinatorial optimization problem on top of the usual RL complexities, and is generally NP-hard. These challenges have motivated a variety of research efforts to learn decision tree policies in a tractable way.

38 1.1 Interpretable Policy Learning in RL

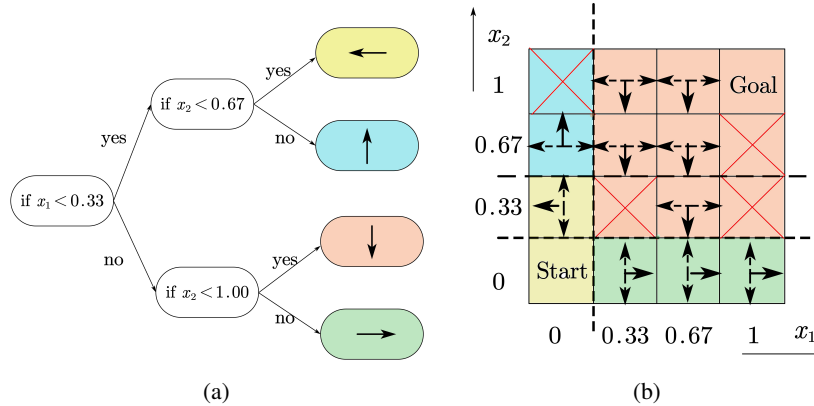


Figure 1: Left: An interpretable policy learned by a decision tree of depth $D = 2$. Right: Illustration of the Frozen Lake 4x4 (Stochastic) Markov Decision Process.

39 A number of prior works have proposed methods to learn interpretable RL policies (decision trees
 40 or other rule-based representations) that balance interpretability and performance. These methods
 41 can be broadly categorized into: (1) relaxation-based or programmatic methods that directly train a
 42 restricted policy, and (2) imitation and distillation methods that extract an interpretable policy from a
 43 complex one. We review key representatives of each approach:

44 One strategy is to soften or relax the tree representation to enable (approximate) gradient-based
 45 optimization. For example, Gupta et al. (2015) [11] introduced a policy tree model that is optimized
 46 via policy gradient, using a smooth parameterization of the splitting criteria. These approaches embed
 47 the non-differentiable tree into a continuous optimization, often achieving decent performance, but
 48 they do not guarantee a globally optimal tree. Another innovative idea is to reformulate the RL
 49 problem itself to enforce a tree policy: Topin et al. (2021) [20] proposed Iterative Bounding MDPs
 50 (IBMDPs), a meta-MDP construction in which any optimal policy corresponds to a decision tree
 51 for the original MDP. By solving the IBMDP with standard deep RL algorithms, they indirectly
 52 obtain a tree-structured policy. This approach allows using powerful function approximation during
 53 training while yielding a discrete tree policy in the end, but the solution quality still depends on
 54 the RL training procedure. Finally, beyond decision trees, researchers have explored programmatic
 55 policy learning. Verma et al. (2018) [22] introduced Programmatically Interpretable RL (PIRL),
 56 using a high-level domain-specific programming language to represent policies. Their method, NDPS
 57 (Neurally Directed Program Search), first trains a neural network policy and then performs a guided
 58 search in the space of programs to find a policy that mimics the neural policy’s decisions.

59 Another framework for obtaining an interpretable policy is to leverage a teacher, typically a high-
 60 performance but complex policy, and train a simpler model to imitate it. Dataset Aggregation
 61 (DAGger) [17] is a classic approach where an agent gradually collects states by following its current
 62 policy and labels them with actions from an expert policy, iteratively improving the learned policy.
 63 VIPER (Verifiable Imitation Policy Extraction) [4] builds on this idea to specifically learn decision
 64 tree policies. VIPER augments DAGger by using the teacher’s Q-value information to focus the
 65 learning on important states. This results in smaller, more accurate decision trees than naive imitation
 66 learning. VIPER demonstrated success in distilling deep RL agents (like DQN policies) into compact
 67 decision trees that can be formally analyzed. In summary, imitation-based methods can produce
 68 high-quality tree policies if the teacher is strong; however, they inherit a fundamental limitation: if
 69 the optimal policy in the MDP is too complex to be represented by a small tree, a student forced to
 70 imitate that complex optimal (or any complex teacher) will struggle. The student might use its limited
 71 capacity to mirror the teacher’s decisions in parts of the state space that are actually unnecessary
 72 to obtain a high reward. In fact, recent work found that when a limited-depth tree is optimal for
 73 the task, directly optimizing that tree can yield better performance than imitating a complex policy
 74 [25]. This highlights an imitation gap: the best interpretable policy may differ from the behavior of a
 75 black-box optimal policy, so chasing the latter via imitation can be counterproductive. In light of this,

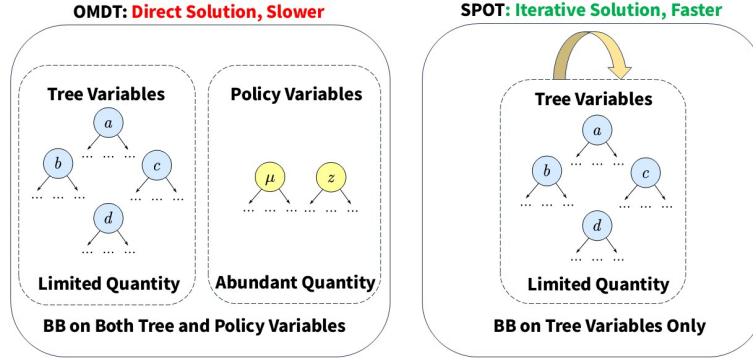


Figure 2: Branching Strategy Comparison between SPOT and OMDT. OMDT uses standard solvers (e.g., Gurobi), requiring branching on all integer variables, thus linearly increasing complexity with the number of states. In contrast, SPOT employs a two-stage reduced-space branch-and-bound approach, branching only on tree variables.

a promising direction is to optimize the decision tree policy directly for the MDP’s returns, rather than relying on a teacher.

1.2 Optimal Decision Tree Policies

To overcome the performance limitations of approximate methods, researchers have begun to tackle the exact optimization of decision-tree policies in MDPs. Over the past decade, several works have formulated the decision tree induction problem as a mixed-integer program (MIP) or other combinatorial optimization problem to find the globally optimal tree.

Notably, Bertsimas & Dunn (2017) [6] and Verwer & Zhang (2017) [23] introduced MIP models for optimal classification trees, and subsequent extensions incorporated various constraints (fairness [1], robustness to adversarial examples [24], etc.) Due to the NP-hard nature of optimal tree induction, a variety of techniques have been explored to improve solver efficiency: dynamic programming for small trees [10], constraint programming and SAT formulations [13, 15, 18, 21], and specialized branch-and-bound search algorithms that cleverly prune the search space [2, 3]. These works demonstrated that for classification/regression tasks with a fixed dataset, one can often compute an optimal decision tree for modestly sized problems, providing a guarantee of maximal accuracy given the tree size. Vos and Verwer [25] brought this exact optimization approach into the RL setting. OMDT (Optimal MDP Decision Trees) formulates finding a decision-tree policy for a given discrete MDP as a single comprehensive MIP. In essence, their formulation combines the standard linear programming formulation of an MDP’s optimal policy with additional integrality constraints that restrict the policy to the structure of a binary decision tree. They link each state’s decision to the traversal of some path in the tree and enforce consistency of actions with the tree’s predictions. Using a MILP solver, OMDT can directly maximize the expected discounted return of a size-limited decision tree policy. However, the exact approach comes with scalability challenges. Solving a large MILP that encodes the entire MDP and a complex policy structure is computationally intensive. In OMDT’s experiments, the approach was feasible for MDPs with on the order of 10^3 – 10^4 states and for trees of depth up to around 5–7. The authors report that in some environments with larger state spaces (e.g. a tic-tac-toe game MDP), OMDT required hours of runtime to surpass the performance of the approximate VIPER method. This has motivated us to improve the scalability of tree policies.

A key open challenge is how to achieve scalability for decision tree policies. In particular, *can we design algorithms that find the tree policy more efficiently than directly solving the MILP, enabling use on problems with significantly more states?*

In contrast to OMDT’s single huge MILP solve, we introduce a decision-tree policy iteration framework that alternates between evaluating the current tree policy and improving it. At each policy improvement step, finding the optimal decision tree (with respect to the current value function) is

formulated as an MILP, but we solve it using a reduced-space branch-and-bound procedure, which leverages the problem’s structure by decoupling the MDP constraints. By distributing our approach across multiple processors, we exploit parallelism to further speed up the search for the optimal tree policy in each iteration. Empirically, in our experiments, this approach outperforms OMDT in both runtime and scalability, finding optimal or near-optimal tree policies in problems where the baseline MILP approach fails or takes prohibitively long.

Our work addresses gaps in interpretable reinforcement learning by developing a scalable and efficient algorithm for optimizing interpretable decision-tree policies in Markov Decision Processes (MDPs). Specifically, our key contributions include:

- Our method produces compact, interpretable policies suitable for deployment in sensitive, safety-critical domains.
- We propose a novel decomposition strategy inspired by policy iteration. Instead of directly solving a monolithic optimization, we iteratively decompose the problem into smaller, more tractable subproblems. We provide theoretical guarantees that each of these subproblems can be solved to global optimality, ensuring overall high-quality solutions.
- By leveraging our decomposition strategy, the optimization problems arising in each iteration become independently solvable and highly parallelizable. This significantly enhances computational efficiency and scalability.
- Extensive numerical experiments on standard benchmarks demonstrate that our approach yields solutions dramatically faster than state-of-the-art methods. We observe significant runtime improvements and demonstrate the capability to efficiently handle larger problem instances.

2 Preliminaries

Markov Decision Processes. A Markov Decision Process (MDP) provides a mathematical framework for modeling sequential decision-making problems under uncertainty. Formally, an MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where \mathcal{S} is a finite set of states, \mathcal{A} is a finite set of actions, $P(i' | i, k)$ represents the transition probability from state i to state i' given action k , $R(i, i', k)$ denotes the immediate reward received for transitioning from state i to i' under action k , and $\gamma \in [0, 1]$ is the discount factor. The goal is to find a policy π that maximizes the expected cumulative discounted reward $\mathbb{E}_\pi [\sum_{t=0}^{\infty} \gamma^t R(i_t, i_{t+1}, k_t)]$.¹

Solving MDPs via Linear Programming and its Dual Formulation. For a finite, discounted MDP, an optimal policy can be computed by formulating and solving a linear programming (LP) problem. Let V_i denote the optimal value function at state i , let $p_0(i)$ represent the probability of starting in state i , and let $P_{i,i',k}$ and $R_{i,i',k}$ denote, respectively, the transition probability and immediate reward when action k is executed in state i transitioning to state i' . The primal LP formulation directly encodes the Bellman optimality conditions in terms of state values:

$$\begin{aligned} \min_V \quad & \sum_{i \in \mathcal{S}} p_0(i) V_i \\ \text{s.t.} \quad & V_i - \gamma \sum_{i' \in \mathcal{S}} P_{i,i',k} V_{i'} \geq \sum_{i' \in \mathcal{S}} P_{i,i',k} R_{i,i',k}, \quad \forall i \in \mathcal{S}, k \in \mathcal{A}. \end{aligned}$$

At optimality, these constraints hold with equality for at least one action per state. Intuitively, the primal LP minimizes each state’s value while satisfying the constraints derived from Bellman’s equation, resulting in the optimal value function. Subsequently, an optimal policy can be derived by selecting the action(s) in each state that yield equality in the corresponding constraints.

Taking the dual of the primal LP yields an alternative formulation where the variables explicitly represent the frequency with which each state-action pair is utilized. Define $\mu_{i,k} \geq 0$ as the *discounted occupancy measure*, representing the expected discounted number of times the agent visits state i and selects action k . The dual LP maximizes the total expected discounted reward subject to linear

¹We use i to represent state and k to represent action, differing from the traditional notation of s and a . This choice is made to distinguish from variables commonly used in decision-tree contexts.

153 *flow-balance* constraints:

$$\begin{aligned}
& \max_{\mu} \quad \sum_{i \in \mathcal{S}} \sum_{k \in \mathcal{A}} \mu_{i,k} \left(\sum_{i' \in \mathcal{S}} P_{i,i',k} R_{i,i',k} \right) \\
& \text{s.t.} \quad \sum_{k \in \mathcal{A}} \mu_{i,k} = p_0(i) + \gamma \sum_{i' \in \mathcal{S}} \sum_{k \in \mathcal{A}} P_{i',i,k} \mu_{i',k}, \quad \forall i \in \mathcal{S}.
\end{aligned} \tag{1}$$

154 In this formulation, the term on the left-hand side of each constraint represents the *flow out* of state i ,
 155 while the right-hand side represents the *flow into* state i , consisting of contributions from the initial
 156 state distribution and transitions from predecessor states. Any feasible solution μ thus corresponds to
 157 a valid policy, and the dual objective directly quantifies the expected return of this policy.

158 3 Scalable Policy Optimization with Trees

159 The primary challenge in solving the full dual optimization problem for tree-structured MDP policies
 160 arises from the coupling introduced by the MDP transition dynamics. Specifically, the system
 161 of equations defined by Eq. (1) creates a fully interconnected set of constraints, wherein each
 162 state’s decision variables depend on transition probabilities that link them to all other states. This
 163 interconnectedness implies that the optimization problem is not decomposable; it cannot be divided
 164 into smaller, independently solvable subproblems because the transition terms inherently couple all
 165 state-action variables together. Consequently, directly solving the complete dual formulation becomes
 166 computationally impractical.

167 To address this issue, we draw inspiration from the classical value iteration algorithm [5, 12]. Rather
 168 than attempting to solve the entire coupled system simultaneously, we adopt an iterative strategy.
 169 We start with an initial estimate of the value function, denoted by V^{old} , and perform a single-step
 170 Bellman update for each state based on this fixed value. For each state i , the single-step Bellman
 171 update can be formulated as:

$$\begin{aligned}
& \min_{V_i} \quad V_i \\
& \text{s.t.} \quad V_i - \gamma \sum_{i' \in \mathcal{S}} P_{i,i',k} V_{i'}^{\text{old}} \geq \sum_{i' \in \mathcal{S}} P_{i,i',k} R_{i,i',k}, \quad \forall k \in \mathcal{A}.
\end{aligned} \tag{2}$$

172 It can be found that iteratively solving Eq. (2) corresponds exactly to performing value iteration. The
 173 critical advantage of adopting this iterative approach in our formulation is its inherent decomposability.
 174 Specifically, in the Bellman backup LP presented above, the constraints are separated by state. Each
 175 state i is associated with its own independent set of inequalities, which eliminates direct coupling and
 176 interdependencies between states.

177 Analyzing the dual form of this single-step LP provides insights into the underlying policy updates.
 178 Specifically, the dual formulation of Eq. (2) for state i can be expressed as

$$\begin{aligned}
& \max_{\mu} \quad \sum_{k \in \mathcal{A}} \mu_{ik} \sum_{i' \in \mathcal{S}} P_{ii',k} (R_{ii',k} + \gamma V_{i'}^{\text{old}}) \\
& \text{s.t.} \quad \sum_{k \in \mathcal{A}} \mu_{ik} = 1.
\end{aligned}$$

179 When extending this policy update formulation jointly across all states \mathcal{S} , we arrive at the following
 180 aggregated optimization problem

$$\max_{\mu} \quad \sum_{i \in \mathcal{S}} \Phi_i^{\pi^{\text{old}}} \sum_{k \in \mathcal{A}} \mu_{ik} \sum_{i' \in \mathcal{S}} P_{ii',k} (R_{ii',k} + \gamma V_{i'}^{\text{old}}) \tag{3a}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{A}} \mu_{ik} = 1, \quad \forall i \in \mathcal{S}. \tag{3b}$$

181 Here, the coefficient $\Phi_i^{\pi^{\text{old}}}$ serves as a weight, indicating the relative importance given to each state i
 182 during the overall policy update. This conceptualization is further clarified in Proposition 1.

Now, we introduce the decision tree policy constraints. Consider that each state i has an associated feature vector $x_i \in \mathbb{R}^F$, with each feature scaled to the unit interval $[0, 1]$. The decision tree consists of nodes indexed by $t = 1, \dots, T$, where $T = 2^{D+1} - 1$ denotes the total number of nodes for a tree of depth D . Following the notation from [14], we let $p(t) = \lfloor t/2 \rfloor$ denote the parent of node t , and define the sets $A_L(t)$ and $A_R(t)$ to include the ancestors of node t where the left or right branch, respectively, is taken along the path from the root to node t . The tree nodes are partitioned into decision nodes $\mathcal{T}_D = \{1, \dots, \lfloor T/2 \rfloor\}$ and leaf nodes $\mathcal{T}_L = \{\lfloor T/2 \rfloor + 1, \dots, T\}$. The resulting constraints for the decision-tree-structured policy optimization problem are as follows:

Tree Constraints

$$\sum_{k \in \mathcal{A}} c_{kt} = 1, \quad \forall t \in \mathcal{T}_L \quad (4a)$$

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad \forall i \in \mathcal{S} \quad (4b)$$

$$a_m^T(x_i + \epsilon_{im} - \epsilon_{\min}) + \epsilon_{\min} \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), \quad \forall t \in \mathcal{T}_L, m \in A_L(t), i \in \mathcal{S} \quad (4c)$$

$$a_m^T x_i \geq b_m - (1 - z_{it}), \quad \forall t \in \mathcal{T}_L, m \in A_R(t), i \in \mathcal{S} \quad (4d)$$

$$\sum_{j=1}^F a_{jt} = d_t, \quad \forall t \in \mathcal{T}_B \quad (4e)$$

$$0 \leq b_t \leq d_t, \quad \forall t \in \mathcal{T}_B \quad (4f)$$

$$d_t \leq d_{p(t)}, \quad \forall t \in \mathcal{T}_B \quad (4g)$$

Binary Constraints

$$a_{jt}, d_t \in \{0, 1\}, \quad \forall t \in \mathcal{T}_B \quad (4h)$$

$$z_{it} \in \{0, 1\}, \quad \forall t \in \mathcal{T}_B \quad (4i)$$

$$c_{kt} \in \{0, 1\}, \quad \forall k \in \mathcal{A}, t \in \mathcal{T}_L \quad (4j)$$

Policy-Tree Coupling Constraint

$$z_{it} + c_{kt} - 1 \leq \mu_{ik}, \quad \forall i \in \mathcal{S}, k \in \mathcal{A}, t \in \mathcal{T}_L \quad (4k)$$

In alignment with the constraint-design framework from [14], the tree structure is encoded through four sets of variables: a, b, c, d . At each decision node $t \in \mathcal{T}_D$, the binary variable d_t indicates whether a node splits. When a split occurs, it is characterized by a binary feature selection vector $a_t = [a_{1t}, \dots, a_{Ft}]^\top \in \{0, 1\}^F$ and a threshold $b_t \in [0, 1]$. At the leaves, c_{kt} specifies the action k assigned to leaf t , and z_{it} records if state i terminates at leaf t . Constants $\epsilon, \epsilon_{\max}, \epsilon_{\min}$ stabilize the mixed-integer big-M calculations as described in [7].

These constraints collectively preserve the logical and structural coherence of the decision tree. Specifically, Constraint (4a) ensures that each leaf node is associated with exactly one action label. Constraint (4b) guarantees that each state is assigned uniquely to one leaf node. Constraints (4c) and (4d) implement the appropriate branching behavior, directing states correctly based on their feature values. Constraints (4e) and (4f) ensure proper handling of non-splitting nodes: when no split occurs at a node (i.e., $d_t = 0$), all states reaching this node follow the right branch, ultimately directing them consistently towards the right-most leaf. Constraint (4g) maintains hierarchical consistency, enforcing that splits at parent nodes precede splits at child nodes, thereby ensuring logical downward propagation of tree branches. Finally, Constraint (4k) explicitly restricts the feasible policy space to policies that can be represented by decision trees.

Next, we explicitly derive $\Phi^{\pi^{\text{old}}}$. This quantity is the *discounted occupancy measure*, representing the expected cumulative number of discounted visits to each state under the current policy π^{old} . Formally, it is computed as:

$$\Phi_i^{\pi^{\text{old}}} = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = i | \pi^{\text{old}})$$

where $\mathbb{P}(s_t = i | \pi^{\text{old}})$ denotes the probability of being in state i at time step t when following policy π^{old} . The complete Algorithm is presented in Algorithm 1.

Algorithm 1: Scalable Policy Optimization with Trees (SPOT)

Input: Number of iterations n , a sequence of threshold $\{\phi_l\}_{l=1}^n$

```
1 Initialize random policy  $\pi_0$  and compute its value function  $V^0$ ;  
2 Set  $V^{\text{old}} \leftarrow V^0$ ;  
3 for  $l = 1$  to  $n$  do  
4   Fix: Select tree parameters as tunable with probability  $\phi_l$ ; fix others;  
5   Solve: With  $V^{\text{old}}$ , solve Problem (3)-(4) via Algorithm 2 to obtain policy  $\pi_l$ ;  
6   Evaluate: Compute  $V_l$  and state distribution  $\Phi^{\pi_l}$  for  $\pi_l$ ;  
7   Update:  $V^{\text{old}} \leftarrow V_l$ ,  $\Phi^{\pi^{\text{old}}} \leftarrow \Phi^{\pi_l}$ ;  
8 end  
Output: The policy in  $\{\pi_1, \dots, \pi_n\}$  with highest expected return
```

212 **Proposition 1.** *Each iteration of SPOT corresponds to performing a one-step policy gradient ascent*
213 *update starting from π^{old} . In particular, the update maximizes the same first-order objective that the*
214 *policy gradient theorem uses, resulting in a new policy that is a greedy improvement on π^{old} .*

215 SPOT differs from traditional policy gradient methods in a key way: instead of taking a small update
216 step in the gradient direction, we fully re-optimize the policy μ to maximize the surrogate linearized
217 objective around π^{old} . This can be interpreted as performing exact policy improvement rather than
218 incremental improvement. Therefore, one iteration of SPOT corresponds to a one-step greedy
219 policy improvement that leverages the structure of the policy gradient objective while optimizing it
220 completely under the current value estimate of π^{old} .

221 **Proposition 2.** *If the class of decision-tree policies is expressive to represent an optimal MDP policy,*
222 *and the weighting vector Φ^{old} is strictly positive, then SPOT is guaranteed to converge to the optimal*
223 *solution.*

224 4 Reduced-Space Branching and Bound Framework

225 The optimization problem described by equations (3)-(4) can be viewed as a two-stage stochastic pro-
226 gramming (TSSP) problem aimed at learning decision-tree policies. Although traditional off-the-shelf
227 solvers are generally effective, directly solving these large-scale problems is often computationally
228 intensive or even intractable in practice.

229 To address this challenge, we utilize a reduced-space branch-and-bound (RSBB) framework specif-
230 ically designed for solving mixed-integer TSSP problems, enhanced with carefully constructed
231 lower and upper bounds [9]. This RSBB procedure initiates with the full feasible region M_0 and
232 iteratively refines the optimality gap by bisecting M_0 into progressively smaller subsets, discarding
233 any subregions that cannot contain the optimal policy. At each iteration, a subregion M is assessed by
234 computing a lower bound $\beta(M)$ and an upper bound $\alpha(M)$. If the gap between these bounds satisfies
235 $\alpha(M) - \beta(M) \leq \varepsilon$, the algorithm terminates. Otherwise, the subregion M is further partitioned
236 according to a defined branching rule, and the process continues iteratively.

237 A significant advantage of this RSBB approach is its proven convergence, which occurs despite
238 branching exclusively on first-stage structural variables, the split-indicator and threshold vectors
239 (i.e., a, b, c, d). It has been demonstrated that even for optimization problems with complex tree-like
240 constraints involving both continuous and binary second-stage variables, the convergence property
241 still holds [14].

242 **Theorem 1.** *RSBB converges to the global optimum, formally expressed as*

$$\lim_{i \rightarrow \infty} \alpha_i = \lim_{i \rightarrow \infty} \beta_i = f \quad (5)$$

243 The finite nature of the feasible region for optimal policies arises naturally from the finite number of
244 distinct tree structures that can result from the discrete selection of thresholds b at each decision node.

245 4.1 Two Stage Stochastic Program

246 Note that variables a, b, c, d describe the structure of the decision tree and are the same for all states,
 247 while policy-related variables z , and μ are state-specific and describe the allocation and reward of
 248 a specific state. Therefore, Problem (3)-(4) with fixed V_{old} can be reformulated as the following
 249 formula:

$$f(M_0) = \max_{m \in M_0} \sum_i Q_i(m), \quad (6)$$

250 where we denote $m = (a, b, c, d)$ as all first-stage variables and $Q_i(\cdot)$ represents the optimal value of
 251 the second-stage problem:

$$Q_i(m) = \max_{z_i, \mu_i} \Phi_i^{\pi^{old}} \sum_k \mu_{ik} \sum_{i'} P_{ii'k} (R_{ii'k} + \gamma V_{i'}^{old}) \quad (7a)$$

$$\text{s.t. Constraint (3a), (4a) } \sim (4k) \quad (7b)$$

252 The closed set $M_0 := [m^l, m^u]$ denotes the region of the first stage variables. $(\cdot)^l, (\cdot)^u$ represent the
 253 lower and upper bound of each variable. In each BB node with a specific partition $M \subseteq M_0$, we
 254 solve the problem $f(M) = \max_{m \in M} \sum_{i \in \mathcal{S}} Q_i(m)$.

255 Leveraging the aforementioned problem reformulation and its inherent decomposable structure, we
 256 can readily derive effective lower and upper bounding methods within our Reduced-Space Branch-
 257 and-Bound (RSBB) framework for solving Problem 6. The lower bounding techniques can directly
 258 utilize existing approaches from the optimal decision tree literature, as detailed in [14], including
 259 scenario grouping, relaxed mixed-integer linear programming (MILP) bound, and trivial primal
 260 bound searching. Different from traditional optimal decision tree policy learning formulation, the
 261 decomposable structure enables a trivial parallelism for a faster convergence of the algorithm.

262 However, to fully leverage the unique structure and decomposability of Problem (6), we introduce
 263 a tailored closed-form decomposable upper bounding strategy specifically designed for this maxi-
 264 mization problem. Additionally, this approach allows us to implement effective bound-tightening
 265 techniques that accelerate convergence within the RSBB framework by pre-determining the optimal
 266 actions for certain states during the branch-and-bound process. The overall structure and procedure
 267 of the RSBB algorithm are summarized in Algorithm 2 (see Supplementary Material).

268 4.2 Closed-Form Upper Bounding Strategy

269 At each branch-and-bound (BB) node, the optimization problem includes an implicit constraint known
 270 as the non-anticipativity constraint within the stochastic programming literature. This constraint
 271 requires that all states share the same first-stage variables (i.e., the decision tree structure). By
 272 relaxing this non-anticipativity constraint, we derive an upper bounding problem defined as

$$\alpha(M) := \min_{m_i \in M} \sum_{i \in \mathcal{S}} Q_i(m_i). \quad (8)$$

273 This relaxed problem naturally decomposes into $|\mathcal{S}|$ independent subproblems: $\alpha_i(M) :=$
 274 $\min_{m \in M} Q_i(m)$ with $\alpha(M) := \sum_{i \in \mathcal{S}} \alpha_i(M)$. The optimal value $\alpha_i(M)$ for each state i can be effi-
 275 ciently computed by enumerating all possible leaf nodes that state i could reach, without explicitly
 276 solving any optimization problems. The computational complexity of this enumeration approach is
 277 $O(|\mathcal{T}_D| + |\mathcal{T}_L|)$. Since decision trees typically maintain a small depth for interpretability, this calcu-
 278 lation can substantially outperform traditional optimization methods. Given a bound $M = [m^l, m^u]$,
 279 the enumeration process exhaustively identifies all feasible leaf nodes for state i , thereby determining
 280 the global optimal value of $\alpha_i(M)$.

281 **State Action Pre-determination** An important benefit of the enumeration process utilized in the
 282 closed-form upper bound calculation is the reduction of feasible leaf nodes for each state i from the
 283 full set \mathcal{T}_L to a smaller set \mathcal{T}_{z_i} . Here, \mathcal{T}_{z_i} denotes the set of leaf nodes reachable by state i given the
 284 current subregion M . By comparing the action selection indicators c_{kt} with their associated range
 285 across each leaf node in \mathcal{T}_{z_i} , it becomes possible to pre-determine the optimal action and reward for
 286 certain states even before solving their corresponding subproblems explicitly. Specifically, the reward

and optimal action for state i under the current branch-and-bound (BB) node can be evaluated by, for any $k \in \mathcal{A}, i \in \mathcal{S}$,

$$R_i = \begin{cases} Q_{ik} & \text{if } \bigwedge_{t \in T_{z_i}} c_{kt}^l = c_{kt}^u = 1 \\ \text{undtm} & \text{otherwise.} \end{cases} \quad (9)$$

Once the optimal action and corresponding value for a specific state i have been determined at a node within the branch-and-bound (BB) algorithm, this state can be excluded from subsequent upper-bound calculations. This effectively reduces the computational load by decreasing the number of states to consider. Importantly, these predetermined state-action pairs remain valid throughout all subsequent descendant nodes in the BB search tree. This proactive state-action determination significantly mitigates computational complexity, especially at deeper levels of the search tree. By avoiding redundant optimization subproblems for previously determined states, the overall efficiency and scalability of the algorithm are markedly enhanced.

5 Experiments

We evaluate our algorithm by comparing it to OMDT [25] on large-scale MDPs.

MDP Preparation We prepare a set of 9 MDP datasets to evaluate our methods: `sys_ad_1`, `sys_ad_2`, `tic_vs_ran`, `tiger_vs_ant`, `csma_2_2`, `csma_2_4`, `firewire`, `wlan0`, and `wlan1`. The first four MDPs are available at <https://github.com/tudelft-cda-lab/OMDT>, while the remaining five can be found at <https://github.com/prismmodelchecker/prism-benchmarks/>. The properties of MDPs are shown in Table 1.

Comparison We evaluate two variants of our method: (1) direct application of SPOT1, and (2) SPOT with a warm start, referred to as SPOT+WS. In SPOT+WS, an initial solution generated by OMDT under a 5-minute time constraint serves as a warm start, subsequently refined by SPOT. All experiments were performed with a uniform 60-minute computational budget. Table 1 reports the performance results for both variants, including the warm-start baselines.

Table 1: Comparison of OMDT and SPOT on Large MDPs with Tree Depth $D = 3$. Values represent normalized returns. The best gain is calculated as the maximum of the SPOT and SPOT+WS gains relative to the absolute value of OMDT (60min). The threshold is set to $\phi_l = 0.5$

MDP	$ \mathcal{S} $	$ \mathcal{A} $	F	OMDT (5m)	OMDT (60m)	SPOT (60m)	SPOT+WS (60m)	Best Gain (%)
<code>sys_ad_1</code>	256	9	8	0.90884	0.94064	0.94260	0.93861	+0.2084
<code>sys_ad_2</code>	256	9	8	0.60547	0.77446	0.80902	0.80902	+4.4625
<code>tic_vs_ran</code>	2424	9	27	-1.0170	-1.0143	-1.0081	-1.0158	+0.6113
<code>tiger_vs_ant</code>	625	5	4	0.47150	0.81333	0.84102	0.65309	+3.4045
<code>csma_2_2</code>	1038	8	11	1.2025e-4	4.7332e-4	2.3898e-4	2.8681e-3	+505.95
<code>csma_2_4</code>	7958	8	11	-1.1582e-2	-1.1582e-2	-1.1699e-2	1.1466e-4	+100.99
<code>firewire</code>	4093	13	10	-9.9350e-3	4.8978e-2	-1.0035e-2	0.63757	+1201.7
<code>wlan0</code>	2954	6	13	0.93399	1.00000	0.99558	1.00000	0.000
<code>wlan1</code>	6825	6	13	-0.76336	0.98730	0.99694	0.99694	+0.9764

Table 1 summarizes the normalized returns for 9 benchmark MDPs. SPOT+WS consistently outperforms both vanilla SPOT and OMDT, achieving notable improvements (e.g., 1201.7% on `firewire` and 505.95% on `csma_2_2`). Even when OMDT already yields strong results (e.g., `wlan0`), SPOT maintains comparable performance.

6 Conclusion

In this paper, we presented a novel framework, SPOT, designed for computing decision tree policies in Markov Decision Processes (MDPs). Our approach addresses the critical computational bottlenecks encountered when optimizing tree-structured policies through policy iteration and a reduced-space branch-and-bound algorithm. Empirical evaluations on benchmark problems consistently show that our approach outperforms existing methods.

Despite these results, our method has some limitations. Its performance still relies on the efficiency of the underlying optimization solver, which may restrict scalability for extremely large or complex MDPs. Future research could explore solver-independent acceleration methods or approximate formulations to extend applicability to very larger-scale decision-making problems.

References

- [1] 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.
- [2] 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.
- [3] 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.
- [4] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable Reinforcement Learning via Policy Extraction, January 2019.
- [5] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.
- [6] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106:1039–1082, 2017.
- [7] Dimitris Bertsimas and Jack Dunn. *Machine Learning Under a Modern Optimization Lens*. Dynamic Ideas LLC, Belmont, MA, 2019.
- [8] Leo Breiman, Jerome Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.
- [9] Yankai Cao and Victor M. Zavala. A scalable global optimization algorithm for stochastic nonlinear programs. *Journal of Global Optimization*, 75(2):393–416, October 2019. Publisher: Springer Science and Business Media LLC.
- [10] 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.
- [11] 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.
- [12] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.
- [13] 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.
- [14] Kaixun Hua, Jiayang Ren, and Yankai Cao. A scalable deterministic global optimization algorithm for training optimal decision tree. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 8347–8359. Curran Associates, Inc., 2022.
- [15] Nina Narodytska, Alexey Ignatiev, Filipe Pereira, and Joao Marques-Silva. Learning optimal decision trees with sat. In *International Joint Conference on Artificial Intelligence 2018*, pages 1362–1368. Association for the Advancement of Artificial Intelligence (AAAI), 2018.
- [16] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994.
- [17] 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.

- 371 [18] André Schidler and Stefan Szeider. Sat-based decision tree learning for large data sets. *Journal*
372 *of Artificial Intelligence Research*, 80:875–918, 2024.
- 373 [19] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient meth-
374 ods for reinforcement learning with function approximation. *Advances in neural information*
375 *processing systems*, 12, 1999.
- 376 [20] Nicholay Topin, Stephanie Milani, Fei Fang, and Manuela Veloso. Iterative bounding mdps:
377 Learning interpretable policies via non-interpretable methods. In *Proceedings of the AAAI*
378 *Conference on Artificial Intelligence*, volume 35, pages 9923–9931, 2021.
- 379 [21] Hélène Verhaeghe, Siegfried Nijssen, Gilles Pesant, Claude-Guy Quimper, and Pierre Schaus.
380 Learning optimal decision trees using constraint programming. *Constraints*, 25:226–250, 2020.
- 381 [22] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri.
382 Programmatically interpretable reinforcement learning. *ArXiv*, abs/1804.02477, 2018.
- 383 [23] Sicco Verwer and Yingqian Zhang. Learning decision trees with flexible constraints and
384 objectives using integer optimization. In *Integration of AI and OR Techniques in Constraint*
385 *Programming: 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017,*
386 *Proceedings 14*, pages 94–103. Springer, 2017.
- 387 [24] Daniël Vos and Sicco Verwer. Efficient training of robust decision trees against adversarial
388 examples. In *International Conference on Machine Learning*, pages 10586–10595. PMLR,
389 2021.
- 390 [25] Daniël Vos and Sicco Verwer. Optimal Decision Tree Policies for Markov Decision Processes. In
391 *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages
392 5457–5465, Macau, SAR China, August 2023. International Joint Conferences on Artificial
393 Intelligence Organization.

394 A Pseudocode for Algorithm 2

Algorithm 2: Reduced-Space Branch-and-Bound for Optimal Tree Policy

Input: M_0 , non-zero tolerance ϵ

- 1 Set iteration index $i = 0$, $\mathbb{M} \leftarrow \{M_0\}$;
- 2 Initial upper and lower bounds $\alpha_i = \alpha(M_0)$, $\beta_i = \beta(M_0)$;
- 3 **repeat**
- 4 **Node Selection**
- 5 Select a set $M \in \mathbb{M}$ satisfying $\beta(M) = \beta_i$;
- 6 $\mathbb{M} \leftarrow \mathbb{M} \setminus \{M\}$;
- 7 $i \leftarrow i + 1$;
- 8 **Branching**
- 9 Partition M into subsets M_1 and M_2 according to the branch strategy in Section D;
- 10 Add each subset to \mathbb{M} to create separated descendent nodes;
- 11 **Bounding**
- 12 Compute $\alpha(M_1)$, $\beta(M_1)$, $\alpha(M_2)$, $\beta(M_2)$;
- 13 If $\beta_s(M_j)$, $j \in \{1, 2\}$ is infeasible for some $s \in \mathcal{S}$, $\mathbb{M} \leftarrow \mathbb{M} \setminus \{M_j\}$;
- 14 $\beta_i \leftarrow \min\{\beta(M') \mid M' \in \mathbb{M}\}$;
- 15 $\alpha_i \leftarrow \min\{\alpha_{i-1}, \alpha(M_1), \alpha(M_2)\}$;
- 16 Remove all M' from \mathbb{M} if $\beta(M') \geq \alpha_i$;
- 17 If $|\beta_i - \alpha_i| \leq \epsilon$, **STOP**;
- 18 **until** $\mathbb{M} = \emptyset$;

395 B Additional Experiment Details

396 B.1 Experiments Details

397 **Details of SPOT Implementation.** During SPOT training, we adopt a time-constrained setup with a
398 total runtime budget of one hour (60 minutes) and a maximum of five minutes allocated per iteration.
399 At each step, we retain the best solution found within the allotted time window. We fix the threshold
400 ϕ_l at 0.5 and, for simplicity, set all coefficients $Phi_i^{\pi_{old}}$ to 1. To enhance exploration, we employ an
401 epsilon-decay strategy: when computing V^{old} , the evaluation is based on a greedy policy with an
402 exploration probability $\epsilon = 1/l$, where l denotes the current iteration index. With probability ϵ , the
403 policy randomly selects alternative actions rather than the greedy one. This mechanism encourages
404 the policy to explore beyond the most immediate choices. In each iteration, we use Algorithm 2 to
405 compute the solution.

406 B.2 Experiments with Tree Depth $D = 4$

407 **Set up.** The MDP setup and method comparison follow those in Section 5. The SPOT implementation
408 details are identical to Section B.1, except that the tree-depth parameter is now set to $D = 4$.

409 **Results.** Table 2 reports the normalized return across all benchmarks. We find that even the vanilla
410 SPOT method surpasses OMDT in seven out of nine instances—for example, it increases the return on
411 `sys_ad_2` from 0.75403 to 0.83337 (+10.76%) and reduces the loss on `tic_vs_ran` from -1.00610
412 to -1.00089. Incorporating warm-start initialization further amplifies these improvements: on
413 `csma_2_2`, SPOT+WS boosts the return from 0.00024 to 0.01485 (+6101.14%), and on `firewire`,
414 from -0.009935 to 0.63699 (+6514.90%). Even in the moderate case of `csma_2_4`, warm starts
415 boost performance by +100.97% (-0.01158 \rightarrow 0.01147). These results confirm that SPOT’s depth
416 tuning yields steady improvements and that warm-start strategies can unlock substantial additional
417 performance gains in more challenging MDPs.

418 B.3 Experiments on Running Time

419 To demonstrate SPOT’s efficiency in solving each iteration problem, we conduct experiments com-
420 paring the runtime of different solution methods on the same optimization task.

Table 2: Comparison of OMDT and Our Methods on Large MDPs with Tree Depth $D = 4$. Values are normalized returns. “Best Gain (%)” is the maximum relative improvement of either “Ours” or “Warms” over OMDT (60 min).

MDP	$ S $	$ A $	P	OMDT (5 min)	OMDT (60 min)	Ours (60 min)	Warms (60 min)	Best Gain (%)
sys_ad_1	256	9	8	0.90823	0.94403	0.95416	0.95027	+1.07
sys_ad_2	256	9	8	0.74003	0.75403	0.83337	0.83519	+10.76
tic_vs_ran	2424	9	27	-1.01700	-1.00610	-1.00089	0.38958	+138.72
tiger_vs_ant	625	5	4	0.53266	0.80464	0.92347	0.81389	+14.77
csma_2_2	1038	8	11	2.3902e-4	2.3944e-4	-1.2252e-2	1.4848e-2	+6101.14
csma_2_4	7958	8	11	-1.1582e-2	-1.1582e-2	1.1466e-4	-1.1582e-2	+100.97
firewire	4093	13	10	-9.9350e-3	-9.9350e-3	0.63699	0.63757	+6514.90
wlan0	2954	6	13	0.42805	1.00000	0.97792	0.97792	-2.21
wlan1	6825	6	13	-0.76973	0.37665	0.99694	0.99694	+164.71

Setup. For each MDP, we initialize the policy by selecting a fixed action across all states. Based on this fixed policy, we compute the corresponding value function V^{old} and set $\Phi^{\pi^{\text{old}}}$ to the softmax of the discounted state-action occupancy measure. We then perform a single iteration of optimization to find a tree with depth $D = 3$ under three different approaches: (i) solving directly with Gurobi, (ii) using RSBB (reduced space branch-and-bound) in serial, and (iii) using RSBB in parallel across 10 CPU cores. The solver is configured with a gap tolerance of 0.01 and a maximum runtime of 14,800 seconds. Each setup is repeated five times, and we report the mean and standard deviation of the runtime.

Results. Figure 3 summarizes the results. For all instances except `tic_vs_ran` and `tiger_vs_ant`, where none of the methods reach optimality within the time limit, parallel RSBB consistently achieves superior efficiency. For example, on the `firewire` benchmark, solving directly with Gurobi requires 11,528.12 s, whereas parallel RSBB completes the task in only 15.91 s. Similarly, on `csma_2_4`, Gurobi exceeds the 14,800 s limit, while parallel RSBB finishes in just 42.568 s. This dramatic reduction in computation time highlights the efficiency of the parallel RSBB approach.

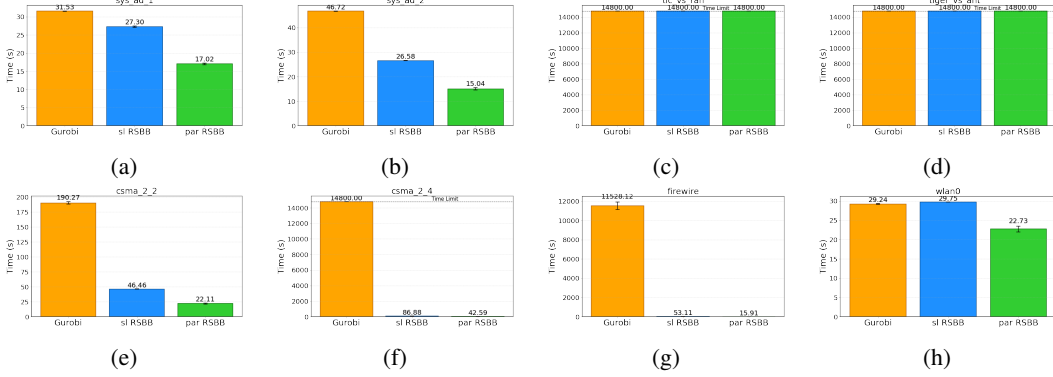


Figure 3: Running time comparisons across different MDPs using Gurobi, serial RSBB, and parallel RSBB. Each bar represents the mean runtime, with error bars showing standard deviation.

C Proofs

C.1 Proof of Proposition 1

Proof. Given the current policy π^{old} , consider the optimization performed by SPOT at the update step:

$$\max_{\mu} \sum_i \Phi_i^{\pi^{\text{old}}} \sum_k \mu_{ik} \sum_{i'} P_{ii'k} (R_{ii'k} + \gamma V_{i'}^{\text{old}}).$$

subject to $\sum_k \mu_{ik} = 1$ for each state i and $\mu_{ik} \geq 0$. Here $P_{ii'k}$ and $R_{ii'k}$ are the transition probability and reward for taking action k in state i and landing in state i' , and $V_{i'}^{\text{old}}$ is the value of state i' under π^{old} . This objective is the expected total discounted return of the new policy evaluated under the state

visitation weights of the old policy. To see this, define the Q -value of π^{old} for state–action pair (i, k) as

$$Q^{\pi^{\text{old}}}(i, k) = \sum_{i'} P_{ii'k} (R_{ii'k} + \gamma V_{i'}^{\text{old}}).$$

Using this definition, we can rewrite the objective in a simpler form:

$$\max_{\mu} \sum_i \Phi_i^{\pi^{\text{old}}} \sum_k \mu_{ik} Q^{\pi^{\text{old}}}(i, k).$$

This expression represents the expected Q -value of the new policy, weighting each state i by its discounted occupancy $\Phi_i^{\pi^{\text{old}}}$ under π^{old} . Because $\Phi^{\pi^{\text{old}}}$ is fixed during this optimization, the objective separates by states. Maximizing it will assign, for each state i , all probability mass μ_{ik} to the action k that has the highest $Q^{\pi^{\text{old}}}(i, k)$. In other words, the optimal μ is the greedy policy π^{new} defined by $\pi^{\text{new}}(k|i) = 1$ for $k = \arg \max_a Q^{\pi^{\text{old}}}(i, a)$. This yields the maximum of the weighted Q -value objective.

To connect this update to the policy gradient, recall the policy gradient theorem [19]. For a differentiable policy π_{θ} , the gradient of the expected return $J(\pi_{\theta})$ can be written as

$$\nabla_{\theta} J(\pi_{\theta}) = \sum_i \Phi_i^{\pi_{\theta}} \sum_k \nabla_{\theta} \pi_{\theta}(k|i) Q^{\pi_{\theta}}(i, k).$$

In particular, at $\theta = \text{old}$ (i.e. at the current policy), the direction of steepest ascent in policy space is determined by the term $\sum_i \Phi_i^{\pi^{\text{old}}} \sum_k Q^{\pi^{\text{old}}}(i, k) \nabla_{\theta} \pi_{\theta}(k|i)$. Intuitively, this means that in each state i , increasing the probability of an action k will increase J at a rate proportional to $\Phi_i^{\pi^{\text{old}}} Q^{\pi^{\text{old}}}(i, k)$. The optimization we performed above uses this same signal: $\Phi_i^{\pi^{\text{old}}} Q^{\pi^{\text{old}}}(i, k)$ appears as the coefficient of μ_{ik} in the objective. Thus, choosing the action that maximizes $Q^{\pi^{\text{old}}}(i, k)$ for each state (i.e. setting $\mu_{ik} = 1$ at the maximizing k) is precisely what the policy gradient would prescribe — it favors actions with the largest positive impact on the objective. In fact, the above μ -update can be seen as solving for the policy that maximizes the first-order expansion of $J(\pi)$ around π^{old} . By taking a greedy, full step in the direction of this gradient (rather than an infinitesimal step), SPOT produces the deterministic policy π^{new} that locally maximizes the expected return improvement.

□

C.2 Proof of Proposition 2

Proof. Assume that the decision tree policy class has enough capacity to represent the optimal policy. This means at each iteration, when SPOT optimizes the tree policy, it can effectively carry out a policy improvement step with respect to the current value function V^{old} . We show that this improvement step will monotonically increase the policy’s performance and reach optimality.

More specifically, for a fixed V^{old} , the objective follows:

$$\max_{\mu} \sum_i \Phi_i^{\pi^{\text{old}}} \sum_k \mu_{ik} \sum_{i'} P_{ii'k} (R_{ii'k} + \gamma V_{i'}^{\text{old}}). \quad (10)$$

Because the maximization separates by state (note that μ_{ik} for each state i appears only in the term for state i), we can optimize each state’s decision independently. Let us define the Q -value of taking action k in state i under the old policy as

$$Q^{\pi^{\text{old}}}(i, k) = \sum_{i'} P_{ii'k} (R_{ii'k} + \gamma V_{i'}^{\text{old}}).$$

Maximizing the inner sum over actions for a given state i will simply pick the action that maximizes this Q -value. In other words, for each state i ,

$$\max_{\mu_i} \sum_k \mu_{ik} Q^{\pi^{\text{old}}}(i, k) = \max_k Q^{\pi^{\text{old}}}(i, k),$$

since the optimal choice is to put all the state’s probability mass on the single action k that yields the highest $Q^{\pi^{\text{old}}}(i, k)$ (this corresponds to choosing that action deterministically in state i). Therefore, the entire objective decouples over states and can be written as:

$$\sum_i \Phi_i^{\pi^{\text{old}}} \max_k Q^{\pi^{\text{old}}}(i, k).$$

473 The policy π^{new} that achieves this maximum is precisely the greedy policy improvement over π^{old} .
 474 for each state i , $\pi^{\text{new}}(i) = \arg \max_k Q^{\pi^{\text{old}}}(i, k)$. By construction, this new policy π^{new} is a decision
 475 tree (deterministic) policy, assuming the tree function approximator is flexible enough to represent
 476 that state-to-action mapping.

477 Crucially, because $\Phi_i^{\pi^{\text{old}}} > 0$ for every state i , no state is ignored in this optimization. The strictly
 478 positive weights ensure that improving the Q -value in any state will increase the overall objective.
 479 In other words, the weighted objective preserves the true ordering of policy performance: if one
 480 policy yields higher value at some state without lowering it elsewhere, the objective (with all-positive
 481 weights) will also be higher for that policy. This guarantees that the greedy step indeed aligns with
 482 improving the actual value function of the policy.

483 The described greedy update is exactly the policy improvement step from dynamic programming. By
 484 the policy improvement theorem [16], the new policy π^{new} is guaranteed to be no worse than π^{old} in
 485 terms of its value for all states, and in fact strictly better for at least one state if π^{old} was not already
 486 optimal courses.cs.washington.edu. In other words, $V^{\pi^{\text{new}}}(i) \geq V^{\pi^{\text{old}}}(i)$ for all states i , with a strict
 487 inequality for some state as long as π^{old} is not optimal. Because our decision-tree function class can
 488 represent the policy exactly and SPOT guarantees solving the optimal policy in each step, it finds the
 489 optimal policy after convergence.

490 □

491 C.3 Proof of Theorem 1

492 The proof of Theorem 1 follows naturally from observing that the set of feasible solutions is finite.
 493 Specifically, since each decision node in the tree can only take a finite set of meaningful split values b ,
 494 the number of distinct decision tree structures generated from these splits is also finite. Consequently,
 495 a finite enumeration of solutions is inherently guaranteed, making the convergence of the proposed
 496 RSBB algorithm straightforward.

497 Additionally, Problem 6 can be viewed as a specialized instance of a two-stage stochastic programming
 498 problem. Consequently, the convergence analysis for our RSBB algorithm can leverage established
 499 methodologies presented in earlier literature, particularly the frameworks described by [9] and [14].
 500 Although our problem features a distinct cost function formulation tailored for reinforcement learning
 501 policy optimization, the fundamental structure of the branching process in SPOT closely mirrors
 502 that of the classification tree optimization problems studied in [14]. Specifically, the choice of
 503 branching variables in each iteration—such as the split indicator variables (a) and threshold variables
 504 (b)—remains identical to those in previous optimal decision tree frameworks. Therefore, despite
 505 variable b being continuous rather than discrete, the convergence properties derived in earlier works
 506 can also be apply in current problem. This continuity does not compromise convergence guarantees,
 507 as the underlying finite combinatorial structure induced by discrete structural variables ensures
 508 that the algorithm systematically explores a finite solution space and thus converges to the global
 509 optimum.

510 D Branching Strategy for Algorithm 2

511 Algorithm 2’s branching strategy prioritizes variables according to their structural significance within
 512 the optimal decision tree. The binary variables d , which directly encode whether individual decision
 513 nodes perform splits, naturally form the foundational structure of the tree. Therefore, branching
 514 decisions first focus on these split-indicator variables.

515 For the remaining variables (a, b, c), we employ a heuristic procedure as follows: we first compute a
 516 branching threshold defined as $\tau = 1 - \frac{1}{2} \|b^u - b^l\|_\infty$. We then compare τ to a uniformly generated

517 random number in the range $[0, 1]$. If this random number exceeds τ , branching occurs on the discrete
518 variable a , which specifies the splitting feature at the decision node. Should all components of
519 a already be fixed, we instead branch on the variable c , controlling the action assignment at leaf
520 nodes. Alternatively, if the random number is less than or equal to τ , branching is conducted on the
521 continuous variable b , selecting the midpoint between its current bounds as the branching point.

522 Within each category of variables (a, b, c), we systematically prioritize branching based on ascending
523 node indices; for example, if two feature-selection variables $a_{j,t}$ and $a_{j,t+1}$ remain undetermined,
524 branching will occur first on the variable associated with the lower-indexed node, i.e., $a_{j,t}$.