SPOT: Scalable Policy Optimization with Trees for Markov Decision Processes

Xuyuan Xiong[1] **Pedro Chumpitaz-Flores**[2] **Kaixun Hua**[*2] **Cheng Hua**[*1]
[1]Shanghai Jiao Tong University [2]University of South Florida

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 reduced-space 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 [27], 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).

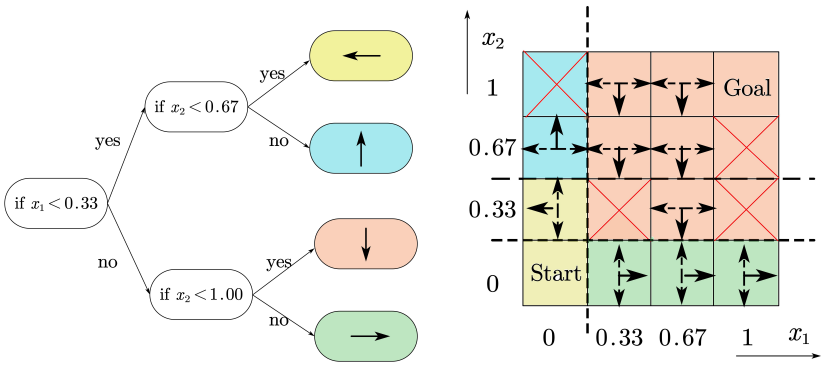
Optimizing a policy constrained to be a decision tree is notoriously difficult [6]. Rudin (2019) [19] identified optimizing sparse logical models such as decision trees as the foremost grand challenge in interpretable machine learning, emphasizing the distinction between inherently interpretable models and post-hoc explanation methods. 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

[*]Corresponding Authors

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

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.

1.1 Interpretable Policy Learning in RL



(a) (b)

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

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

One strategy is to soften or relax the tree representation to enable (approximate) gradient-based optimization. For example, Gupta et al. (2015) [12] introduced a policy tree model that is optimized via policy gradient, using a smooth parameterization of the splitting criteria. These approaches embed the non-differentiable tree into a continuous optimization, often achieving decent performance, but they do not guarantee a globally optimal tree. Another innovative idea is to reformulate the RL problem itself to enforce a tree policy: Topin et al. (2021) [22] proposed Iterative Bounding MDPs (IBMDPs), a meta-MDP construction in which any optimal policy corresponds to a decision tree for the original MDP. By solving the IBMDP with standard deep RL algorithms, they indirectly obtain a tree-structured policy. This approach allows using powerful function approximation during training while yielding a discrete tree policy in the end, but the solution quality still depends on the RL training procedure. Finally, beyond decision trees, researchers have explored programmatic policy learning. Verma et al. (2018) [24] introduced Programmatically Interpretable RL (PIRL), using a high-level domain-specific programming language to represent policies. Their method, NDPS (Neurally Directed Program Search), first trains a neural network policy and then performs a guided search in the space of programs to find a policy that mimics the neural policy's decisions.

Another framework for obtaining an interpretable policy is to leverage a teacher, typically a high-performance but complex policy, and train a simpler model to imitate it. Dataset Aggregation (DAgger) [18] is a classic approach where an agent gradually collects states by following its current policy and labels them with actions from an expert policy, iteratively improving the learned policy. VIPER (Verifiable Imitation Policy Extraction) [4] builds on this idea to specifically learn decision tree policies. VIPER augments DAgger by using the teacher's Q-value information to focus the learning on important states. This results in smaller, more accurate decision trees than naive imitation learning. VIPER demonstrated success in distilling deep RL agents (like DQN policies) into compact decision trees that can be formally analyzed. In summary, imitation-based methods can produce high-quality tree policies if the teacher is strong; however, they inherit a fundamental limitation: if the optimal policy in the MDP is too complex to be represented by a small tree, a student forced to imitate that complex optimal (or any complex teacher) will struggle. The student might use its limited capacity to mirror the teacher's decisions in parts of the state space that are actually unnecessary to obtain a high reward. In fact, recent work found that when a limited-depth tree is optimal for

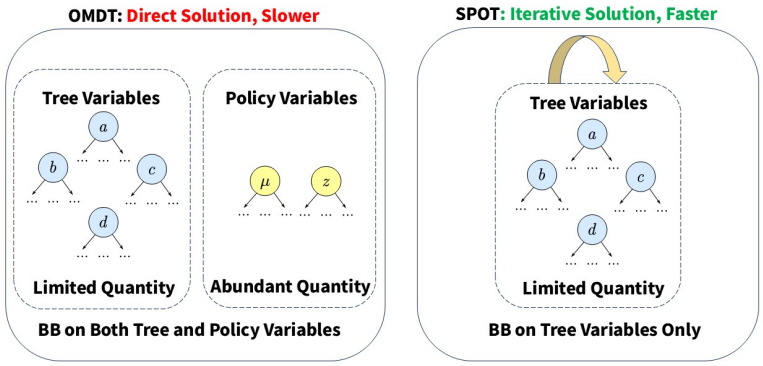


Figure 2: Branching Strategy Comparison between SPOT and OMDT. OMDT uses standard solvers (e.g., Gurobi), requiring branching on all integer variables, thus linearly increasing complexity with the number of states. In contrast, SPOT employs a two-stage reduced-space branch-and-bound approach, branching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

the task, directly optimizing that tree can yield better performance than imitating a complex policy [27]. This highlights an imitation gap: the best interpretable policy may differ from the behavior of a black-box optimal policy, so chasing the latter via imitation can be counterproductive. In light of this, 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) [25] introduced MIP models for optimal classification trees, and subsequent extensions incorporated various constraints (fairness [1], robustness to adversarial examples [26], 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 [14, 16, 20, 23], 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 [27] 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 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). The key contributions of this work are summarized as follows:

- The proposed 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 the 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, $P_{ii'k}$ represents the transition probability from state i to state i' given action k, $R_{ii'k}$ 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 $\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{i_t i_{t+1} k_t}\right]$.[2]

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 $P_{ii'k}$ and $R_{ii'k}$ 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:

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

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 $\varphi_{ik} \geq 0$ as the *discounted occupancy measure*, representing the expected discounted number of times the agent visits state i

[2]We 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.

and selects action k. The dual LP maximizes the total expected discounted reward subject to linear *flow-balance* constraints:

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

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

3 Scalable Policy Optimization with Trees

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

To address this issue, we draw inspiration from the classical value iteration algorithm [5, 13]. Rather than attempting to solve the entire coupled system simultaneously, we adopt an iterative strategy. We start with an initial estimate of the value function, denoted by V^{old}, and perform a single-step Bellman update for each state based on this fixed value. For each state i, the single-step Bellman update can be formulated as:

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

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

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

$$\begin{aligned} \max_{\mu} \quad & \sum_{k\in\mathcal{A}} \mu_{ik} \sum_{i'\in\mathcal{S}} P_{ii'k}(R_{ii'k} + \gamma V_{i'}^{\text{old}}) \\ \text{s.t.} \quad & \sum_{k\in\mathcal{A}} \mu_{ik} = 1, \\ & \mu_{ik} \geq 0, \quad \forall k \in \mathcal{A}, \end{aligned}$$

where μ_{ik} is the dual variable which can also considered as the *policy*. When extending this policy update formulation jointly across all states $\mathcal{S}$, we arrive at the following aggregated optimization problem

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

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

$$\mu_{ik} \geq 0, \quad \forall i \in \mathcal{S}, k \in \mathcal{A}. \tag{3c}$$

Here, the coefficient $\Phi_i^{\pi^{\text{old}}}$ serves as a weight, indicating the relative importance given to each state i 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 $t = 1, \ldots, T$, where $T = 2^{D+1} - 1$ denotes the total number of nodes for a tree of depth D. Following the notation from [15], 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 $\mathcal{T}_B = \{1, \ldots, \lfloor T/2 \rfloor\}$ and leaf nodes $\mathcal{T}_L = \{\lfloor T/2 \rfloor + 1, \ldots, T\}$. The resulting constraints for the decision-tree-structured policy optimization problem are as follows:

Tree Constraints

$$\sum_{k \in \mathcal{A}} c_{kt} = 1, \qquad \forall t \in \mathcal{T}_L \tag{3d}$$

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \qquad \forall i \in \mathcal{S} \tag{3e}$$

$$a_m^T(x_i + \epsilon - \epsilon_{\min}) + \epsilon_{\min} \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), \qquad \forall t \in \mathcal{T}_L, m \in A_L(t), i \in \mathcal{S} \tag{3f}$$

$$a_m^T x_i \geq b_m - (1 - z_{it}), \qquad \forall t \in \mathcal{T}_L, m \in A_R(t), i \in \mathcal{S} \tag{3g}$$

$$\sum_{j=1}^{F} a_{jt} = d_t, \qquad \forall t \in \mathcal{T}_B \tag{3h}$$

$$0 \leq b_t \leq d_t, \qquad \forall t \in \mathcal{T}_B \tag{3i}$$

$$d_t \leq d_p(t), \qquad \forall t \in \mathcal{T}_B \tag{3j}$$

Binary Constraints

$$a_{jt}, d_t \in \{0, 1\}, \qquad \forall t \in \mathcal{T}_B \tag{3k}$$

$$z_{it} \in \{0, 1\}, \qquad \forall t \in \mathcal{T}_L \tag{3l}$$

$$c_{kt} \in \{0, 1\}, \qquad \forall k \in \mathcal{A}, t \in \mathcal{T}_L \tag{3m}$$

Policy-Tree Coupling Constraint

$$z_{it} + c_{kt} - 1 \leq \mu_{ik}, \qquad \forall i \in \mathcal{S}, k \in \mathcal{A}, t \in \mathcal{T}_L \tag{3n}$$

In alignment with the constraint-design framework from [15], the tree structure is encoded through four sets of variables: a, b, c, d. At each decision node $t \in \mathcal{T}_B$, the binary variable d_t indicates whether a node splits. When a split occurs, it is characterized by a binary feature selection vector $a_t = [a_{1t}, \ldots, a_{Ft}]^\top \in \{0, 1\}^F$ 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] (See also Appendix A).

These constraints collectively preserve the logical and structural coherence of the decision tree. Specifically, Constraint (3d) ensures that each leaf node is associated with exactly one action label. Constraint (3e) guarantees that each state is assigned uniquely to one leaf node. Constraints (3f) and (3g) implement the appropriate branching behavior, directing states correctly based on their feature values. Constraints (3h) and (3i) 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 (3j) maintains hierarchical consistency, enforcing that splits at parent nodes precede splits at child nodes, thereby ensuring logical downward propagation of tree branches. Finally, Constraint (3n) 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 defined as:

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

Algorithm 1: Scalable Policy Optimization with Trees (SPOT)

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

Initialize random policy π_0 and compute its value function V^0;

Set $V^{\text{old}} \leftarrow V^0$;

for $l = 1$ *to* n **do**

  Fix: Select tree parameters as tunable with probability ϕ_l; fix others;

  Solve: With V^{old}, solve Problem (3) via Algorithm 2 to obtain policy π_l;

  Evaluate: Compute V_l and state distribution Φ^{π_l} for π_l;

  Update: $V^{\text{old}} \leftarrow V_l$, $\Phi^{\pi^{\text{old}}} \leftarrow \Phi^{\pi_l}$;

end

Output: The policy in $\{\pi_1, \ldots, \pi_n\}$ with highest expected return

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.

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

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

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

4 Reduced-Space Branch-and-Bound Framework

The optimization problem described by (3) can be viewed as a two-stage stochastic programming (TSSP) problem. Although traditional off-the-shelf solvers are generally effective, directly solving these large-scale problems is often computationally intensive or even intractable in practice.

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

A significant advantage of this RSBB approach is its proven convergence, which occurs despite branching exclusively on tree-structure descriptive variables, such as the split-indicator and threshold vectors (i.e., a, b, c, d in constraints of Problem (3)). This convergence property remains valid [15] even in the presence of constraints involving both continuous and binary second-stage variables.

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

$$\lim_{i \to \infty} \alpha_i = \lim_{i \to \infty} \beta_i = f. \tag{4}$$

Here, f indicates the complete MILP formula of Problem (3). The finite nature of the feasible region for optimal policies arises naturally from the finite number of distinct tree structures that can result from the discrete selection of thresholds b at each decision node.

4.1 Two Stage Stochastic Program

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

$$f(M_0) = \max_{m \in M_0} \sum_i Q_i(m), \tag{5}$$

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

$$Q_i(m) = \max_{z_i, \mu_i} \ \Phi_i^{\pi^{\text{old}}} \sum_k \mu_{ik} \sum_{i'} P_{ii'k} \left(R_{ii'k} + \gamma V_{i'}^{\text{old}} \right) \tag{6a}$$

$$\text{s.t. Constraints (3a), (3d) } \sim \text{ (3n).} \tag{6b}$$

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

Leveraging this problem reformulation and its decomposable structure, we can derive effective lower and upper bounding strategies within the RSBB framework to solve Problem (5). The lower bounds can be directly constructed using existing techniques from the optimal decision tree literature [15], including scenario grouping, relaxed mixed-integer linear programming (MILP) bounds, and simple primal bound searches. Unlike traditional formulations for optimal decision tree policy learning, the decomposable structure of our approach enables straightforward parallelization, which significantly accelerates convergence.

However, to fully leverage the unique structure and decomposability of Problem (5), we introduce a tailored closed-form decomposable upper bounding strategy specifically designed for this maximization problem. Additionally, this approach allows us to implement effective bound-tightening techniques that accelerate convergence within the RSBB framework by pre-determining the optimal actions for certain states during the branch-and-bound process. The overall structure and procedure of the RSBB algorithm are summarized in Algorithm 2 (see Supplementary Material).

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

Input: M_0, non-zero tolerance ϵ

Set iteration index $i = 0$, $\mathbb{M} \leftarrow \{M_0\}$;
Initial upper and lower bounds $\alpha_i = \alpha(M_0)$, $\beta_i = \beta(M_0)$;
repeat
  Node Selection
    Select a set $M \in \mathbb{M}$ satisfying $\beta(M) = \beta_i$;
    $\mathbb{M} \leftarrow \mathbb{M} \setminus \{M\}$;
    $i \leftarrow i + 1$;
  Branching
    Partition M into subsets M_1 and M_2 according to the branch strategy in Section E;
    Add each subset to $\mathbb{M}$ to create separated descendent nodes;
  Bounding
    Compute $\alpha(M_1)$, $\beta(M_1)$, $\alpha(M_2)$, $\beta(M_2)$;
    If $\beta_s(M_j), j \in \{1, 2\}$ is infeasible for some $s \in \mathcal{S}$, $\mathbb{M} \leftarrow \mathbb{M} \setminus \{M_j\}$;
    $\beta_i \leftarrow \max\{\beta(M') \mid M' \in \mathbb{M}\}$;
    $\alpha_i \leftarrow \max\{\alpha_{i-1}, \alpha(M_1), \alpha(M_2)\}$;
    Remove all M' from $\mathbb{M}$ if $\alpha(M') \leq \beta_i$;
    If $|\beta_i - \alpha_i| \leq \delta$, STOP;
until $\mathbb{M} = \emptyset$;

4.2 Closed-Form Upper Bounding Strategy

At each BB node, the optimization problem includes an implicit constraint known as the non-anticipativity constraint within the stochastic programming literature. This constraint requires that

all states share the same first-stage variables (i.e., the decision tree structure). By relaxing this non-anticipativity constraint, we derive an upper bounding problem defined as

$$\alpha(M) := \max_{m_i \in M} \sum_{i \in \mathcal{S}} Q_i(m_i). \tag{7}$$

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

State Action Pre-determination An important benefit of the enumeration process utilized in the closed-form upper bound calculation is the reduction of feasible leaf nodes for each state i from the 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 current subregion M. By comparing the action selection indicators c_{kt} with their associated range (i.e., c^l_{kt}, c^u_{kt}) across each leaf node $t \in \mathcal{T}_{z_i}$, it becomes possible to pre-determine the optimal action and reward for 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 $k \in \mathcal{A}, i \in \mathcal{S}$,

$$R_i = \begin{cases} Q_{ik} & \text{if } \bigwedge_{t \in T_{z_i}} c^l_{kt} = c^u_{kt} = 1 \\ \perp & \text{otherwise,} \end{cases} \tag{8}$$

where $\perp$ means the upper bound is not determined.

Once the optimal action and corresponding value for a specific state i have been determined at a node within the 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 [27] 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 SPOT (Algorithm 1), 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 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`). Furthermore, our analysis reveals that the warm start strategy is particularly effective for SPOT on larger MDPs, such as `firewire`, `csma_2_4`, and `wlan0`. Even when OMDT already yields strong results (e.g., `wlan0`), SPOT maintains comparable performance.

Table 1: Comparison of OMDT and SPOT on Large MDPs with Tree Depth $D = 3$. Values represent normalized returns. The gain is calculated as 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 (5 min)	OMDT (60 min)	SPOT (60 min)	SPOT+WS (60 min)	SPOT Gain (%)	SPOT+WS Gain (%)
sys_ad_1	256	9	8	0.90884	0.94064	0.94260	0.93861	**+0.2084**	-0.2158
sys_ad_2	256	9	8	0.60547	0.77446	0.80902	0.80902	**+4.4625**	**+4.4625**
tic_vs_ran	2424	9	27	-1.0170	-1.0143	-1.0081	-1.0158	**+0.6113**	-0.1479
tiger_vs_ant	625	5	4	0.47150	0.81333	0.84102	0.65309	**+3.4045**	-19.7016
csma_2_2	1038	8	11	1.2025e-4	4.7332e-4	2.3898e-4	2.8681e-3	-49.5165	**+505.95**
csma_2_4	7958	8	11	-1.1582e-2	-1.1582e-2	-1.1699e-2	1.1466e-4	-1.0103	**+100.99**
firewire	4093	13	10	-9.9350e-3	4.8978e-2	-1.0035e-2	0.63757	-120.4926	**+1201.7**
wlan0	2954	6	13	0.93399	1.00000	0.99558	1.00000	-0.4420	**0.0000**
wlan1	6825	6	13	-0.76336	0.98730	0.99694	0.99694	**+0.9764**	**+0.9764**

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. SPOT provides solid improvements on challenging MDPs while maintaining competitive performance on simpler instances

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 large-scale decision-making problems.

Acknowledgments and Disclosure of Funding

The authors are grateful to Wenhui Zhao for his valuable feedback. Cheng Hua is partly supported by the National Natural Science Foundation of China (72301172, 72394370:72394375, 72495130:72495132), Shanghai Education Commission Chenguang Program (22CGA12), and Shanghai Jiao Tong University Office of Liberal Arts (ZHWK2502).

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] Claire Glanois, Paul Weng, Matthieu Zimmer, Dong Li, Tianpei Yang, Jianye Hao, and Wulong Liu. A survey on interpretable reinforcement learning. *Machine Learning*, 113(8):5847–5890, 2024.

[12] 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.

[13] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[14] 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.

[15] 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.

[16] 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.

[17] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994.

[18] 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.

[19] 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.

[20] André Schidler and Stefan Szeider. Sat-based decision tree learning for large data sets. *Journal of Artificial Intelligence Research*, 80:875–918, 2024.

[21] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[22] 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.

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

[24] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. *ArXiv*, abs/1804.02477, 2018.

[25] Sicco Verwer and Yingqian Zhang. Learning decision trees with flexible constraints and objectives using integer optimization. In *Integration of AI and OR Techniques in Constraint Programming: 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings 14*, pages 94–103. Springer, 2017.

[26] Daniël Vos and Sicco Verwer. Efficient training of robust decision trees against adversarial examples. In *International Conference on Machine Learning*, pages 10586–10595. PMLR, 2021.

[27] Daniël Vos and Sicco Verwer. Optimal Decision Tree Policies for Markov Decision Processes. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 5457–5465, Macau, SAR China, August 2023. International Joint Conferences on Artificial Intelligence Organization.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes] .

   Justification: The claims made in the abstract and introduction accurately reflect the core contributions of the paper, including the proposed decomposition strategy, the accompanying theoretical analysis, and the well-designed experiments that support the main findings.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: We state our limitations in the conclusion.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All proof are put in appendix and all theory are clearly stated with proper assumptions.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The detailed experiment setting are explained in Section 5 and we also provide a complete code in the supplementary material for reproduction.

   Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide the whole code package including the testing data file in the supplementary material for reproduction.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.

- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We explained our experimental settings in Section 5.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We have additional experiment results with statistical analysis such as error bars in the appendix file of supplementary material package.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5 and Supplementary material provide the details.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This work conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: In the part of introduction, we state the importance of explainable RL model in many critical applications, which will help the development of our society.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

Justification: The paper poses no such risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The resource of datasets are introducted in Section 5 and part of them comes from the free release of the author of OMDT paper, while others are xxxxx. Their usage complies with their terms of access and non-commercial research intent.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: The paper does not release new assets.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Notation

We now clarify the symbols we used in writing:

Table 2: Summary of Notation

Category	Description	Example Variables
Decision Variables	Optimization variables to be solved	$a_{jt} \in \{0,1\}$, $b_t \in [0,1]$, $\mu_{ik} \in [0,1]$
MDP Parameters	Fixed transition probabilities and rewards	$P_{ii'k}$, $R_{ii'k}$, γ
Algorithm Constants[3]	Fixed hyperparameters and computed values	ϵ, ϵ_{max},ϵ_{min}, $\Phi_i^{\pi^{\text{old}}}$
State/Action Indices	Index sets for states and actions	$i, i' \in \mathcal{S}$, $k \in \mathcal{A}$
Time Indices	Temporal or iteration indices	t

B Case study: `tiger_vs_ant`

The `tiger_vs_ant` environment is a 5×5 grid world in which a tiger pursues an antelope. The state is the tuple antelope_x, antelope_y, tiger_x, tiger_y, with features normalized to [0,1]. The tiger can move up, right, down, left, or wait. The antelope moves randomly among valid cells, never leaving the grid or stepping onto the tiger's current cell. The agent receives a reward of 1 when the tiger catches the antelope (same cell) and 0 otherwise, and the episode ends upon capture, and rewards are discounted with a factor $\gamma \in (0,1)$. The task is challenging because the agent must anticipate stochastic antelope motion and use positioning to restrict escape routes

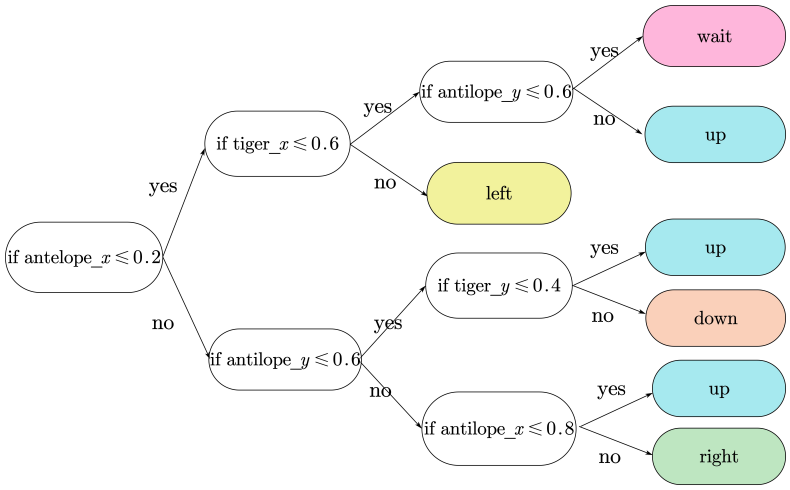


Figure 3: Learned interpretable decision tree policy with $D = 3$ in environment `tiger_vs_ant`

We will demonstrate the interpretability of the above learned policy through the following aspects:

1. Domain-Meaningful Behavior (which shows the strategic coherence)

From the learned decision tree policy, we summarize the following strategies:

Cornering Strategy: When antelope is trapped near the left wall ($x \leq 0.2$), the tiger adapts:

[3] ϵ, ϵ_{max},ϵ_{min} are introduced due to the Big-M reformulation on tree direction constraints. Here ϵ is a vector of length F (i.e., the number of state features), where for each state feature j, we define ϵ_j as follow [7]: $\epsilon_j = \min\{x_{(i+1),j} - x_{i,j} | x_{(i+1),j} \neq x_{i,j}, i \in [|\mathcal{S}| - 1]\}$, where $x_{i,j}$ denotes the ith largest value in the jth feature. $\epsilon_{min} = \min_j\{\epsilon_j\}$ and $\epsilon_{max} = \max_j\{\epsilon_j\}$.

- If tiger is well-positioned ($x \leq 0.6$), it waits patiently or chases vertically
- If tiger is far away ($x > 0.6$), it prioritizes closing the horizontal gap

Pursuit Strategy: In open areas, the tiger uses intelligent vertical positioning:

- Moves upward when positioned below the prey (tiger_y ≤ 0.4)
- Moves downward when positioned above the prey (tiger_y > 0.4)

Escape Prevention: Near top boundary (antelope_y > 0.8), the policy focuses on blocking escape routes rather than direct pursuit

2. Addresses Complexity Concern (This pattern correlates to the concept of Simulatability in [11])

- Only 8 decision rules cover the entire 625-state space (5^4 states)
- Each path through the tree corresponds to a clear strategic situation
- The shallow depth ($D = 3$) ensures human comprehension

3. Uses Interpretable Features (This pattern correlates to the concept of Decomposability in [11])

- Spatial boundaries: Recognizes walls ($x \leq 0.2$ for left edge, $y > 0.8$ for top)
- Relative positioning: Implicitly reasons about tiger-antelope spatial relationships
- No complex engineered features: Uses raw normalized coordinates that directly map to grid positions.

C Additional Experiment Details

C.1 Experiment Details

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

C.2 Experiments with Tree Depth $D = 4$

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

Results. Table 3 reports the normalized return across all benchmarks. We find that even the vanilla SPOT method surpasses OMDT in seven out of nine instances—for example, it increases the return on `sys_ad_2` from 0.75403 to 0.83337 (+10.76%) and reduces the loss on `tic_vs_ran` from –1.00610 to –1.00089. Incorporating warm-start initialization further amplifies these improvements: on `csma_2_2`, SPOT+WS boosts the return from 0.00024 to 0.01485 (+6101.14%), and on `firewire`, from –0.009935 to 0.63699 (+6514.90%). Even in the moderate case of `csma_2_4`, warm starts boost performance by +100.97% (–0.01158 $\rightarrow$ 0.01147). These results confirm that SPOT's depth tuning yields steady improvements and that warm-start strategies can unlock substantial additional performance gains in more challenging MDPs.

C.3 Experiments on Running Time

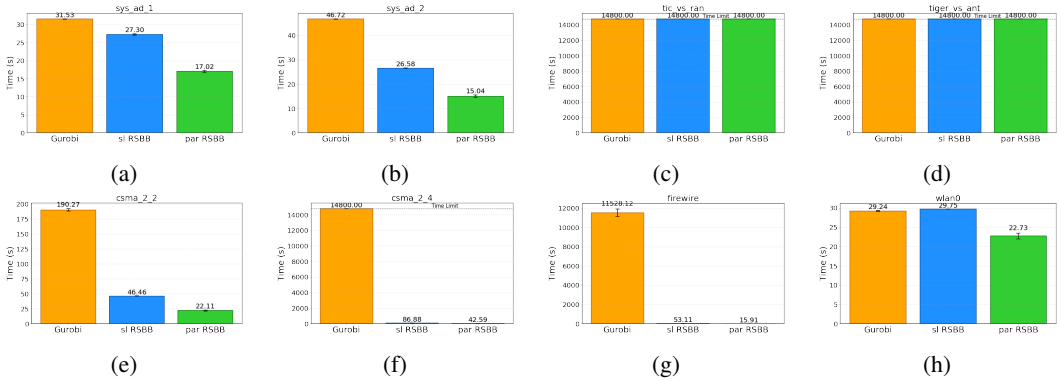
To demonstrate SPOT's efficiency in solving each iteration problem, we conduct experiments comparing the runtime of different solution methods on the same optimization task.

Table 3: Comparison of OMDT and Our Methods on Large MDPs with Tree Depth $D = 4$. Values are normalized returns.

MDP	$\|\mathcal{S}\|$	$\|\mathcal{A}\|$	F	OMDT (5 min)	OMDT (60 min)	SPOT (60 min)	SPOT+WS (60 min)	SPOT Gain (%)	SPOT+WS Gain (%)
`sys_ad_1`	256	9	8	0.90823	0.94403	0.95416	0.95027	**+1.073**	+0.6610
`sys_ad_2`	256	9	8	0.74003	0.75403	0.83337	0.83519	+10.52	**+10.76**
`tic_vs_ran`	2424	9	27	-1.01700	-1.00610	-1.00089	0.38958	+0.5178	**+138.7**
`tiger_vs_ant`	625	5	4	0.53266	0.80464	0.92347	0.81389	**+14.77**	+1.150
`csma_2_2`	1038	8	11	2.3902e-4	2.3944e-4	-1.2252e-2	1.4848e-2	-5216.9	**+6101.14**
`csma_2_4`	7958	8	11	-1.1582e-2	-1.1582e-2	1.1466e-4	-1.1582e-2	**+100.99**	0.0000
`firewire`	4093	13	10	-9.9350e-3	-9.9350e-3	0.63699	0.63757	+6511.6	**+6517.4**
`wlan0`	2954	6	13	0.42805	1.00000	0.97792	0.97792	-2.208	-2.208
`wlan1`	6825	6	13	-0.76973	0.37665	0.99694	0.99694	**+164.68**	**+164.68**

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 4 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.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 4: 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.

D Proofs

D.1 Proof of Proposition 1

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

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

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

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

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 [21]. For a differentiable policy π_θ, the gradient of the expected return $J(\pi_\theta)$ can be written as

$$\nabla_\theta J(\pi_\theta) = \sum_{i} \Phi_i^{\pi_\theta} \sum_{k} \nabla_\theta \pi_\theta(k \mid i) Q^{\pi_\theta}(i,k).$$

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.

□

D.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:

$$\max_{\mu} \sum_{i} \Phi_i^{\pi^{\text{old}}} \sum_{k} \mu_{ik} \sum_{i'} P_{ii'k}\left(R_{ii'k} + \gamma V_{i'}^{\text{old}}\right). \tag{9}$$

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

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

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:

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

The policy π^{new} that achieves this maximum is precisely the greedy policy improvement over π^{old}: 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 tree (deterministic) policy, assuming the tree function approximator is flexible enough to represent that state-to-action mapping.

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

The described greedy update is exactly the policy improvement step from dynamic programming. By the policy improvement theorem [17], the new policy π^{new} is guaranteed to be no worse than π^{old} in terms of its value for all states, and in fact strictly better for at least one state if π^{old} was not already optimal. In other words, $V^{\pi^{\text{new}}}(i) \geq V^{\pi^{\text{old}}}(i)$ for all states i, with a strict inequality for some state as long as π^{old} is not optimal. Because our decision-tree function class can represent the policy exactly and SPOT guarantees solving the optimal policy in each step, it finds the optimal policy after convergence.

□

D.3 Proof of Theorem 1

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

Additionally, Problem (5) can be viewed as a specialized instance of a two-stage stochastic programming problem. Consequently, the convergence analysis for our RSBB algorithm can leverage established methodologies presented in earlier literature, particularly the frameworks described by [9] and [15]. Although our problem features a distinct cost function formulation tailored for reinforcement learning policy optimization, the fundamental structure of the branching process in SPOT closely mirrors that of the classification tree optimization problems studied in [15]. Specifically, the choice of branching variables in each iteration, such as the split indicator variables (d), state feature selection indicator variables (a), and threshold variables (b), remain identical to those in previous optimal decision tree frameworks. Therefore, despite variable b being continuous rather than discrete, the convergence properties derived in earlier works can also be applied in the current problem. This continuity does not compromise convergence guarantees, as the underlying finite combinatorial structure induced by discrete structural variables ensures that the algorithm systematically explores a finite solution space and thus converges to the global optimum.

E Branching Strategy for Algorithm 2

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

For the remaining variables (a, b, c), we employ a heuristic procedure as follows: we first compute a branching threshold defined as $\tau = 1 - \frac{1}{2}\|b^u - b^l\|_\infty$. We then compare τ to a uniformly generated random number in the range $[0, 1]$. If this random number exceeds τ, branching occurs on the discrete variable a, which specifies the splitting feature at the decision node. Should all components of a already be fixed, we instead branch on the variable c, controlling the action assignment at leaf nodes. Alternatively, if the random number is less than or equal to τ, branching is conducted on the continuous variable b, selecting the midpoint between its current bounds as the branching point.

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