Limits of reinforcement learning for decision trees in Markov decision processes

Anonymous Authors[1]

Abstract

For applications like medicine, machine learning models ought to be interpretable. In that case, models like decision trees are preferred over neural networks because humans can read their predictions from the root to the leaves. Learning such decision trees for sequential decision making problems is a relatively new research direction and most of the existing literature focuses on imitating (or distilling) neural networks. In contrast, we study reinforcement learning (RL) algorithms that *directly* return decision trees optimizing some trade-off of cumulative rewards and interpretability in a Markov decision process (MDP). We show that such algorithms can be seen as learning policies for partially observable Markov decision processes (POMDPs). We use this parallel to understand why in practice it is often easier to use imitation learning than to learn the decision tree from scratch for MDPs.

1. Introduction

Interpretability in machine learning is commonly divided into local and global approaches (**?**). Local methods—also referred to as explainability or post-hoc methods (**?**)—provide explanations for individual predictions using tools such as local linear approximations (**?**), saliency maps (**?**), feature attributions (**?**), or attention mechanisms (**?**). Although widely used, these methods approximate the behavior of an underlying black-box model and may therefore be unfaithful to its true computations (**?**).

Global interpretability approaches instead restrict the model class so that the learned model is transparent by construction. Decision trees (**?**) are a canonical example, as their predictions can be inspected, reasoned about, and formally verified. This makes them particularly attractive for safety-critical applications and has motivated extensive research in supervised learning (**?????**).

Extending global interpretability to sequential decision making, however, remains challenging. Existing approaches largely rely on *indirect* methods (**?**): a high-performing but opaque policy (typically a neural network) is first learned using reinforcement learning, and an interpretable model is then trained to imitate its behavior. A prominent example is VIPER (**?**), which distills neural network policies into decision trees using imitation learning (**?**). Such methods have demonstrated strong empirical performance and enable formal verification (**?**), but they optimize a surrogate objective—policy imitation—rather than the original reinforcement learning objective. As a result, the best decision tree policy for the task may differ substantially from the tree that best approximates a neural expert. The curious reader will find an example of this phenomenon in the appendix B.

This limitation motivates the study of *direct* approaches that learn interpretable policies by optimizing the reinforcement learning objective itself. While direct decision tree learning is well understood in supervised settings, it is far less developed for sequential decision making. Understanding why direct optimization is difficult—and when it can succeed—is the central focus of this work.

In this article, we show that reinforcement learning of decision tree policies for MDPs, i.e. learning a decision tree that directly optimizes the cumulative reward of the process without relying on a black-box expert, is often very difficult. To do so, we construct very simple MDPs for which we know optimal decision tree policies and show that RL consistently fails to retrieve those policies. We identify partial observability as a key reason for those failures.

In section 2, we present the related work on reinforcement learning to train decision tree policies for MDPs. In section 3, we present key concepts for decsion trees, MDPs, and the formalism of **?** for reinforcement learning of decision tree policies. In section 4, we show that this direct approach is equivalent to learning a *deterministic memoryless* policy for partially observable MDP (POMDP) (**?**) which is a hard problem (**?**). In section 5, we present our methodology to

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

benchmark RL algorithms that train decision tree policies. In section 5.3, we show that when RL fails to retrieve optimal decision tree policies for MDPs it is most likely because partial observability is involved.

2. Related work

There exist reinforcement learning algorithms that directly train decision tree policies optimizing the cumulative rewards in a given MDP. These approaches can be divided into methods based on *parametric* and *non-parametric* trees.

Parametric decision trees fix the structure in advance and only learn decision thresholds, enabling differentiable optimization with policy gradients (**?**). Several works (**???**) train such trees with PPO, but the fixed structure limits adaptively balancing interpretability and performance, often requiring pruning or stabilization tricks (**?**).

Non-parametric trees instead grow structure during training, as in supervised learning (**??**), but extending this to optimize cumulative MDP reward remains largely unexplored (**?**). To our knowledge, only **?** study this setting via iterative bounding MDPs (IBMDPs), where certain policies correspond to downstream decision tree policies and can be learned with standard RL.

A few specialized approaches also exist, e.g., constructive methods for mazes (**?**) or planning-based shallow trees in known MDPs (**?**).

3. Technical preliminaries

3.1. Markov decision processes

Markov decision processes were first introduced in the 1950s by Richard Bellman (**?**). Informally, an MDP models how an agent acts over time to achieve a goal. At every time step, the agent observes its current state (e.g., patient weight and tumor size) and takes an action (e.g., administers a certain amount of chemotherapy). The agent receives a reward that helps evaluate the quality of the action with respect to the goal (e.g., tumor size decrease when the objective is to cure cancer). Finally, the agent transitions to a new state (e.g., the updated patient state) and repeats this process over time:

Definition 3.1 (Markov decision process)**.** An MDP is a tuple $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$. S is a finite set of states representing all possible configurations of the environment. A is a finite set of actions available to the agent. $R : S \times A \to \mathbb{R}$ is a deterministic reward function that assigns a real-valued reward to each state-action pair. While in general reward functions are often stochastic, in this manuscript we focus deterministic ones without loss of generality. $T : S \times A \to \Delta(S)$ is the transition function that maps state-action pairs to probability distributions over next states $\Delta(S)$. $T_0 \in \Delta(S)$ is the initial distribution over states.

Informally, we would like to act in an MDP so that we obtain as much reward as possible over time. We can formally define this objective, that we call the reinforcement learning objective, as follows:

Definition 3.2 (Reinforcement learning objective)**.** Given an MDP (definition 3.1) $\mathcal{M} \equiv \langle S, A, R, T, T_0 \rangle$, the goal of reinforcement learning for sequential decision making is to find a model, also known as a policy, $\pi : S \to A$ that maximizes the expected discounted sum of rewards:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \mid s_0 \sim T_0, s_{t+1} \sim T(s_t, \pi(s_t))\right]$$

where $0 < \gamma \leq 1$ is the discount factor that controls the trade-off between immediate and future rewards.

Algorithms presented in this article aim to find an optimal policy $\pi^\star \in \underset{\pi}{\operatorname{argmax}}\, J(\pi)$ that maximizes the above reinforcement learning objective. In particular, RL algorithms (**?????**) learn such optimal policies using data of MDP interactions without prior knowledge of the reward and transition models. Useful quantities for such algorithms include *value* of states and actions.

Definition 3.3 (Value of a state)**.** In an MDP $\mathcal{M}$ (definition 3.1), the value of a state $s \in S$ under policy π is the expected discounted sum of rewards starting from state s and following policy π:

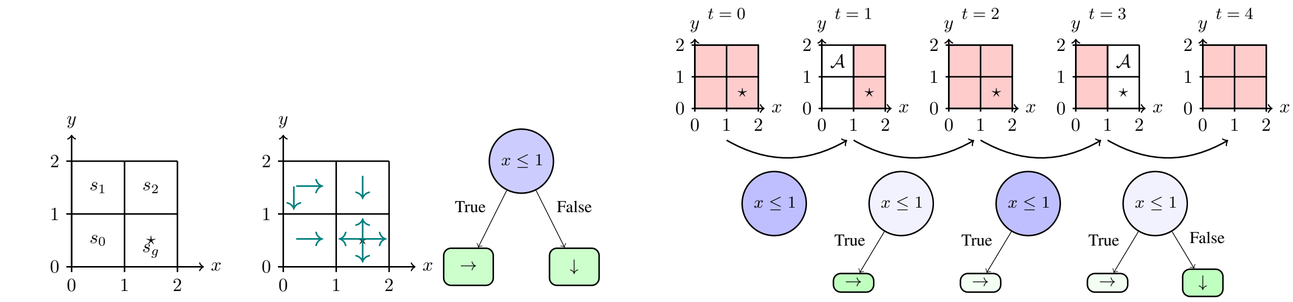
$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \mid s_0 = s, s_{t+1} \sim T(s_t, \pi(s_t))\right]$$

Applying the Markov property gives a recursive definition of the value of s under policy π: $V^\pi(s) = R(s, \pi(s)) + \gamma \mathbb{E}\left[V^\pi(s') | s' \sim T(s, \pi(s))\right]$. The optimal value of a state $s \in S$, $V^\star(s)$, is the value of state s when following the optimal policy $\pi^\star$ (the policy that maximizes the RL objective (definition 3.2)): $V^\star(s) = V^{\pi^\star}(s)$. Similarly, the optimal value of a state–action pair $(s, a) \in S \times A$, $Q^\star(s, a)$, is the value when taking action a in state s and then following the optimal policy: $Q^\star(s, a) = R(s, a) + \gamma \mathbb{E}\left[V^\star(s') | s' \sim T(s, a)\right]$.

3.2. Decision tree policies

While other interpretable policy classes exist (**?**), one conjecture from **?** is that interpretable models are all hard to optimize or learn because they are non-differentiable in nature. This is something that will be key in our study of decision tree policy that we introduce next.

Definition 3.4 (Decision tree policy)**.** A decision tree policy is a rooted tree $\pi_{\mathcal{T}} = (\mathcal{N}, E)$. Each internal node $\nu \in \mathcal{N}$ is associated with a test that maps an MDP state attribute to a Boolean, e.g. $s_i \leq v$ for $v \in \mathbb{R}$. Each edge $e \in E$ from an internal node corresponds to an outcome of the associated



(a) MDP, optimal actions, and an optimal decision tree policy

(b) An associated IBMDP trajectory.

Figure 1. On the left of figure 1a, a grid world MDP with, four states, four directional actions that moves an agent, and rewards of 0 for every transitions except when taking an action in the bottom right goal state ($\star$). The states have discrete labels that represent their coordinates in the $[0, 2] \times [0, 2]$ square, e.g. for s_0, $x = 0.5,\ y = 0.5$. In the centre, an optimal actions w.r.t the RL objective (definition 3.2). On the right, an optimal depth-1 decision tree policy w.r.t the RL objective (definition 3.2) that takes a state label as input, performs some tests on the state coordinate, and returns a directional action. On figure 1b, an IBMDP trajectory when the downstream MDP is the grid world from figure 1a. In the top row, we graphically represent what the IBMDP state at a given time t, and in the bottom row, we present the corresponding decision tree policy traversal. When the pink covers the whole grid, the information contained in the observation $\boldsymbol{o}_t$ could be interpreted as 'the current state features could be anywhere in the grid". The more information-gathering actions are taken, the more refined the bounds on the current downstream state features get. At $t = 0$, the downstream state features are $\boldsymbol{s}_0 = (0.5, 1.5)$ and the initial observation is always the downstream MDP default state feature bounds (definition 3.5): $\boldsymbol{o}_0 = (0, 2, 0, 2)$ because the downstream state features are in $[0, 2] \times [0, 2]$. This means that the overall IBMDP state is $\boldsymbol{s}_{IB} = (0.5, 1.5, 0, 2, 0, 2)$. The first action is an IGA $\langle x, 1\rangle$ that tests the feature x of the downstream state against the value 1 and the reward is ζ (definition 3.5). This transition corresponds to going through an internal node $x \leq 1$ in a decision tree policy as illustrated in the figure. At $t = 1$, after gathering the information that the x-value of the current downstream state is below 1, the observation is updated with the refined bounds $\boldsymbol{o}_1 = (0, 1, 0, 2)$, i.e. more information has been gathered and the obstructed pink area shrinks. The downstream state features remain unchanged. The agent then takes a downstream action that is to move right. This gives a reward 0, resets the observation to the original downstream state feature bounds, and changes the features to $\boldsymbol{s}_2 = (1.5, 1.5)$. And the trajectory continues like this until the absorbing downstream state $\boldsymbol{s}_4 = (1.5, 0.5)$ is reached. By masking the current downstream state features to an agent, we would force it to take information-gathering actions otherwise it would not know how to act optimally (figure 1a).

test function. Each leaf node $l \in \mathcal{N}$ is associated with an MDP action $a_l \in A$. For any input $s \in S$, the tree defines a unique path from root to leaf, determining the prediction $\pi_{\mathcal{T}}(s) = a_l$ where l is the reached leaf. The depth of a tree is the maximum path length from root to any leaf.

In figure 1a, we present an example Markov decision process (definition 3.1), the optimal actions that maximize the RL objective (definition 3.2), and a decision tree policy (definition 3.4) that also maximizes the RL objective. Next, we present the class of MDPs introduced in **?** useful for our to understand reinforcement learning of decision tree policies that directly optimize the RL objective in an MDP.

3.3. Iterative bounding Markov decision processes

The key thing to know about iterative bounding Markov decision processes (IBMDPs) is that they are, as their name suggests, MDPs (definition 3.1). Hence, IBMDPs admit an optimal deterministic Markovian policy that maximizes the RL objective (**?**). From now on, we will assume that all the MDPs we consider are MDPs with continuous state spaces and a finite set of actions, and we use bold fonts for states and observations as that are vector-valued. However all our results generalize to discrete states (in $\mathbb{Z}^m$) MDPs that we can factor using one-hot encodings. Given an MDP for which we want to learn a decision tree policy–the downstream MDP–IBMDP states are concatenations of the downstream MDP state features and some observations. Those observations are information about the downstream state features that are refined–'iteratively bounded"– at each step. Those observations essentially represent some knowledge about where some downstream state features lie in the state space. Actions available in an IBMDP are: (i) the actions of the downstream MDP, that change downstream state features, and (ii) *information-gathering* actions (IGAs) that change the aforementioned observations. Now, downstream actions in an IBMDP are rewarded like in the downstream MDP, this ensures that the RL objective w.r.t. the downstream MDP is encoded in the IBMDP reward. When taking an information-gathering action, the reward is an arbitrary value such that optimizing the RL objective in the IBMDP is equivalent to optimizing some trade-off between interpretability and the RL objective in the downstream MDP. Before showing how to get decision tree policies from IBMDP policies, we give a formal definition of IBMDPs following **?**:

Definition 3.5 (Iterative bounding Markov decision process (**?**, section 4.1))**.** Given a downstream MDP $\mathcal{M} \equiv \langle S, A, R, T, T_0\rangle$ (definition 3.1), an associated iterative bounding Markov decision process $\mathcal{M}_{IB}$ is a tuple:

$$\langle \overbrace{S \times O}^{\text{State space}}, \overbrace{A \cup A_{info}}^{\text{Action space}}, \overbrace{(R, \zeta)}^{\text{Reward}}, \overbrace{(T_{info}, T, T_0)}^{\text{Transitions}} \rangle$$

S are the downstream MDP state features. Downstream state features $\boldsymbol{s} = (s_1, \dots, s_p) \in S$ are bounded: $s_j \in [L_j, U_j]$ where $\infty < L_j \leq U_j < \infty \; \forall 1 \leq j \leq p$. O are observations. They represent bounds on the downstream state features: $O \subsetneq S^2 = [L_1, U_1] \times \cdots \times [L_p, U_p] \times [L_1, U_1] \times \cdots \times [L_p, U_p]$. So the complete IBMDP state space is $S \times O$: the concatenations of downstream state features and observations. Given some downstream state features $\boldsymbol{s} = (s_1, \dots, s_p) \in S$ and some observation $\boldsymbol{o} = (L_1, U_1, \dots, L_p, U_p)$, an IBMDP state is $\boldsymbol{s}_{IB} = (\overbrace{s_1, \dots, s_p}^{\text{downstream state features}}, \overbrace{L_1, U_1, \dots, L_p, U_p}^{\text{observation}})$. A are the downstream MDP actions. A_{info} are *information-gathering* actions of the form $\langle j, v \rangle$ where j is a state feature index $1 \leq j \leq p$ and v is a real number between L_j and U_j. So the complete action space of an IBMDP is the set of downstream MDP actions and information-gathering actions $A \cup A_{info}$. $R : S \times A \to \mathbb{R}$ is the downstream MDP reward function. ζ is a reward signal for taking an information-gathering action. So the IBMDP reward function is to get a reward from the downstream MDP if the action is a downstream MDP action or to get ζ if the action is an IGA action. $T_{info} : S \times O \times (A_{info} \cup A) \to \Delta(S \times O)$ is the transition function of IBMDPs: given some observation $\boldsymbol{o}_t = (L'_1, U'_1, \dots, L'_p, U'_p) \in O$ and downstream state features $\boldsymbol{s}_t = (s'_1, s'_2, \dots, s'_p)$ if an IGA $\langle j, v \rangle$ is taken, the new observation $\boldsymbol{o}_{t+1}$ is $(L'_1, U'_1, \dots, L'_j, \min\{v, U'_j\}, \dots, L'_p, U'_p)$ if $s_j \leq v$ or $(L'_1, U'_1, \dots, \max\{v, L'_j\}, U'_j, \dots, L'_p, U'_p)$ if $s_j > v$. If a downstream action is taken, the observation is reset to the default downstream state feature bounds $(L_1, U_1, \dots, L_p, U_p)$ and the downstream state features change according to the downstream MDP transition function: $\boldsymbol{s}_{t+1} \sim T(\boldsymbol{s}_t, a_t)$. At initialization, the downstream state features are drawn from the downstream MDP initial distribution T_0 and the observation is always set to the default downstream state features bounds $\boldsymbol{o}_0 = (L_1, U_1, \dots, L_p, U_p)$.

We present an IBMDP for a the grid-world MDP (figure 1a) in figure 1b. Now remains to extract a decision tree policy (definition 3.4) for a downstream MDP $\mathcal{M}$ from a policy for an associated IBMDP $\mathcal{M}_{IB}$.

3.4. From policies to trees

information-gathering actions (definition 3.5) resemble the tests $1_{\{x_{\cdot,j} \leq v\}}$ that make up internal decision tree nodes (figure 1b). Indeed, a policy taking actions in an IBMDP essentially builds a tree by taking sequences of IGAs (internal nodes) and then a downstream action (leaf node) and repeats this process over time. In particular, the IGA rewards ζ can be seen as a regularization or a penalty for interpretability: if ζ is very small compared to downstream rewards, a policy will try to take downstream actions as often as possible, i.e. build shallow trees with short paths between root and leaves.

? show that not all IBMDP policies are decision tree policies for the downstream MDP. In particular, their algorithm that converts IBMDP policies into decision trees (algorithm 1) takes as input deterministic policies depending solely on the observations of the IBMDP. If the policies were not deterministic, different subtrees could stem from similar IBMDP observations: this means that the corresponding policy would be a stochastic decision tree (**?**). While there is nothing wrong in learning stochastic decision tree policies from a performance standpoint, interpreting a stochastic policy is an open problem that is not our focus. If the policies were depending on the current full IBMDP state rather than solely on the current IBMDP observation, then a learning agent has 0 incentive to take information-gathering actions to build a decision tree policy as all the state information required to optimally control the downstream MDP as well as downstream actions are available to the agent (definition 3.5 and figure 1b). The connections between partially observable MDPs (POMDPs (**??**)) and extracting decision tree policies from IBMDPs might seem obvious but they are absent from the original IBMDP paper (**?**) and from subsequent work. In the next section we bridge this gap.

4. Bridging the gap with the partially observable MDPs literature

4.1. An adequate formalism

To better understand reinforcement learning of decision tree policies for MDPs, we explictly re-write the problem of optimizing a deterministic policy depending on current observations in an IBMDP as the problem of optimizing a deterministic policy depending only on current observations–also known as a deterministic *memoryless* policy–in a partially observable Markov decision process (POMDP (**?**)). By doing so, we can leverage results from the POMDP literature that is richer than interpretable reinforcement learning literature.

Definition 4.1 (Partially observable Markov decision process)**.** A partially observable Markov decision process is a tuple $\langle X, A, O, R, T, T_0, \Omega \rangle$. X is the hidden state space. A is a finite set of actions. O is a set of observations. $T : X \times A \to \Delta(X)$ is the transition function, where $T(\boldsymbol{x}_t, a, \boldsymbol{x}_{t+1}) = P(\boldsymbol{x}_t | \boldsymbol{x}_{t+1}, a)$ is the probability of transitioning to state $\boldsymbol{x}_t$ when taking action a in state $\boldsymbol{x}$. T_0: is the initial distribution over states. $\Omega : X \to \Delta(O)$ is the observation function, where $\Omega(\boldsymbol{o}, a, \boldsymbol{x}) = P(\boldsymbol{o} | \boldsymbol{x}, a)$ is the probability of observing $\boldsymbol{o}$ in state $\boldsymbol{x}$. $R : X \times A \to \mathbb{R}$ is the reward function, where $R(\boldsymbol{x}, a)$ is the immediate reward for taking action a in state $\boldsymbol{x}$. Note that $\langle X, A, R, T, T_0\rangle$ defines an MDP (definition 3.1).

Next, we can define partially observable iterative bounding Markov decision processes (POIBMDPs). They are IBMDPs (definition 3.5) for which we explicitly define an observation space and an observation function.

Definition 4.2 (Partially observable iterative bounding Markov decision process)**.** a partially observable iterative bounding Markov decision process $\mathcal{M}_{POIB}$ is a tuple:

$$\langle \overbrace{S \times O}^{\text{States}}, \overbrace{A \cup A_{info}}^{\text{Action space}}, \overbrace{O}^{\text{Observations}}, \overbrace{(R, \zeta)}^{\text{Rewards}}, \overbrace{(T_{info}, T, T_0)}^{\text{Transitions}}, \Omega\rangle$$

, where $\langle S \times O, A \cup A_{info}, (R, \zeta), (T, T_0, T_{info})\rangle$ is an IBMDP (definition 3.5). The transition function Ω maps concatenation of state features and observations–IBMDP states–to observations, $\Omega : S \times O \to O$, with $P(\boldsymbol{o}|(\boldsymbol{s}, \boldsymbol{o})) = 1$

POIBMDPs are particular instances of POMDPs where the observation function simply applies a mask over some features of the hidden state. This setting has other names in the literature. For example, POIBMDPs are mixed observability MDPs (**?**) with downstream MDP state features as the *hidden variables* and feature bounds as *visible* variables. POIBMDPs can also be seen as non-stationary MDPs (N-MDPs) (**?**) in which there is one different transition function per downstream MDP state: these are called hidden-mode MDPs (**?**). Following (**?**) we can write the value of a deterministic memoryless policy $\pi : O \to A \cup A_{info}$ in observation $\boldsymbol{o}$.

Definition 4.3 (Value of an observation)**.** In a POIBMDP (definition 4.2), the expected cumulative discounted reward of a deterministic memoryless policy $\pi : O \to A \cup A_{info}$ starting from observation o is $V^\pi(\boldsymbol{o})$: $V^\pi(\boldsymbol{o}) = \sum_{(s,\boldsymbol{o}') \in S \times O} P^\pi((\boldsymbol{s}, \boldsymbol{o}')|\boldsymbol{o}) V^\pi((\boldsymbol{s}, \boldsymbol{o}'))$. With $P^\pi((\boldsymbol{s}, \boldsymbol{o}')|\boldsymbol{o})$ the asymptotic occupancy distribution (see section 4 from **?** for the full definition) of the hidden POIBMDP state $(\boldsymbol{s}, \boldsymbol{o}')$ given the partial observation o and $V^\pi((s, o'))$ the classical state-value function (definition 3.3). We abuse notation and denote both values of observations and values of states by V since the function input is not ambiguous.

4.2. Reinforcement learning in POMDP

In general, the policy that maximizes the RL objective (definition 3.2) in a POMDP (definition 4.1) maps "belief states" or observation histories to actions (**?**). Hence, those policies do not correspond to decicion trees since we require that policies depend only on the current observation (algorithm 1). If we did not have this constraint, we could apply any standard RL algorithm to solve POIBMDPs by seeking policies depending on belief states or observations histories because those are sufficient to optimally control any POMDP (**??**).

In particular, the problem of finding the optimal deterministic memoryless policies for POMDPs is NP-HARD, even with full knowledge of transitions and rewards(section3.2 from **?**). It means that it is impractical to enumerate all possible policies and take the best one. For even moderate-sized POMDPs, a brute-force approach would take a very long time since there are $|A|^{|O|}$ deterministic memoryless policies. Hence it is interesting to study reinforcement learning for finding the best deterministic memoryless policy since it would not enumerate the whole solution space.

In **?**, the authors show that the optimal memoryless policy can be stochastic. Hence, policy gradient algorithms (**?**)–that return stochastic policies–are to avoid since we seek the best *deterministic* policy. Furthermore, the optimal deterministic memoryless policy might not maximize all the values of all observations simultaneously (**?**) which makes it difficult to use TD-learning algorithms like Q-learning (**?**). Indeed, doing a TD-learning update of one observation's value (definition 4.3) can change the value of *all* other observations in an uncontrollable manner because of the dependence in $P^\pi((s, \boldsymbol{o}')|\boldsymbol{o})$ (definition 4.3).

Despite those hardness results, applying RL to POMDPs, by naively replacing the processes (hidden) states $\boldsymbol{x}$ by the observation $\boldsymbol{o}$ in Q-learning or Sarsa (**?**), has already demonstrated successful in practice (**?**). More recently, the framework **??** called asymmetric RL, has also shown promising results to learn policies for POMDPs. Asymmetric RL algorithms train a model–a policy or a value function–depending on hidden state (only available at train time) and a history dependent (or observation dependent) model. The history or observation dependent model serves as target or critic to train the hidden state dependent model. The history dependent (or observation dependent) model can thus be deployed in the POMDP after training since it does not require access to the hidden state to output actions. In appendix F we present asymmetric variants of standard tabular RL algorithm. For example, asymmetric Q-learning is a variant of Q-learning that returns a deterministic memoryless policy. Given a POMDP, asymmetric Q-learning trains a partially observable Q-function $Q : O \times A \to \mathbb{R}$ and a Q-function $U : X \times A \to \mathbb{R}$. The hidden state dependent Q-function U serves as a target in the temporal difference learning update. It is the tabular version of the modified DQN algoritm used in **?**. Indeed, when learning deterministic memoryless policies for IBMDPs, **?** were using RL algorithms corresponding to asymmetric DQN or asymmetric PPO from **??** before those were published. We provide a reproducibility study in appendix C of **?**. In appendix F, we describe tabular asymmetric variants such as asymmetric Q-learning, which learns an observable Q-function $Q : O \times A \to \mathbb{R}$ using a hidden-state target $U : X \times A \to \mathbb{R}$. This mirrors the modified DQN approach used in **?**, which effectively applied asymmetric DQN/PPO before these frameworks were formalized. We also provide a reproducibility study of **?** in appendix C. Until recently, the benefits of asymmetric RL over standard RL was only shown empirically and only for history-dependent models. The work of **?** proves that some asymmetric RL algorithms should in theory learn better history-dependent **or** memoryless policies for POMDPs. This is exactly what we wish for. However, those algorithms are not practical because they require estimations of the asymptotic occupancy distribution $P^{\pi}((s, \boldsymbol{o}')|\boldsymbol{o})$ (definition 4.3) for candidate policies which in turn would require to perform costly Monte-Carlo estimations. We leave it to future work to use those algorithms that combine asymmetric RL and estimation of future visitation frequencies since those results are contemporary to the writing of this manuscript.

5. Methodology

The goal of this section is to check if the aforementioned approach can consistently retrieve optimal decision tree policies for a simple grid world MDP (figure 1a). In particular, we use reinforcement learning to train decision tree policies for MDPs by seeking deterministic memoryless policies that optimize the RL objective in POIBMDPs (figure 1b and section 4).

5.1. Computing some decision tree policies

To assess the performance of reinforcement learning, for different trade-off reward ζ (definition 3.5), we identify the deterministic memoryless policies that maximize the RL objective (definition 3.2). Each of those policies correspond to one of the decision tree policies for the grid world downstream MDP illustrated in figure 2: (i) a depth-0 tree equivalent to always taking the same downstream action ($\pi_{\mathcal{T}_0}$), (ii) a depth-1 tree equivalent alternating between an IGA and a downstream action ($\pi_{\mathcal{T}_1}$), (iii) an unbalanced depth-2 tree that sometimes takes two IGAs then a downstream action and sometimes a an IGA then a downstream action ($\pi_{\mathcal{T}_u}$), (iv) a depth-2 tree that alternates between taking two IGAs and a downstream action ($\pi_{\mathcal{T}_2}$), or (v) an infinite 'tree" that only takes IGAs. We will particularly focus on trying to retrieve the depth-1 decision tree that is the most interpretable–smallest number of nodes and shallowest–tree taking optimal actions. Because from **?** we know that for POMDPs, stochastic memoryless policies can sometimes get better expected discounted rewards than deterministic memoryless policies, we also compute the value of the stochastic policy that randomly alternates between two downstream actions: $\rightarrow$ and $\downarrow$. Taking those two downstream actions always lead to the goal state in expectation (figure 1a). Because we know all the downstream states, all the observations, all the actions, all the rewards and all the transitions of our POIBMDPs, we can compute the RL objective values of those different deterministic memoryless policies exactly given ζ and γ a discount factor. We plot, in figure 2, the RL objective values of the decision tree policies as functions of ζ when we fix $\gamma = 0.99$ (standard choice of discount in practice (**?**)). Despite objective values being very similar for the depth-1 and unbalanced depth-2 tree, we now know from the green shaded area that a depth-1 tree is optimal when $0 < \zeta < 1$ in the POIBMDP. Interestingly, two challenges of learning in POMDPs described in (**?**) are visible in figure 2. First, there is a whole range of ζ values for which the optimal memoryless policy is stochastic. Second, for e.g. $\zeta = 0.5$, while the optimal deterministic memoryless policy corresponds to a depth-1 tree, the value of the (PO)IBMDP state $(\boldsymbol{s}_2, \boldsymbol{o}_0) = (1.5, 1.5, 0, 2, 0, 2)$, i.e. the agent current position is upper right cell in the downstream MDP but it has not information about it, is not maximized by the optimal memoryless policy that will take 1 information-gathering action in between each downstream action (figure 2), but by the sub-optimal policy that always goes down. Next, we present the specific experimental setup that we use in the remaining of the paper.

5.2. Experimental setup

All our code is open source[1]. The downstream MDP for which we want to learn decision tree policies with reinforcement learning is the grid world MDP (figure 1a). We then get 100 associated POIBMDPs following figure 1b with ζ chosen uniformly among 100 different in $]0, 1[$.

We will evaluate two types of reinforcement learning algorithm. Frist we use standard tabular RL algorithms, namely Q-learning, Sarsa, and vanilla policy gradient on a softmax policy (**???**), to learn deterministic memoryless policies in POIBMDPs by simply replacing the current state in the algorithm descriptions by the current observation. In theory the policy gradient algorithm should not be a good candidate for our problem since it searches for stochastic policies that we showed can be better than our sought depth-1 decision tree policy (figure 2), but for completeness we will see what trees are obtained after greddification of the stochastic policies.

Second, we also use the more specialised asymmetric Q-learning, asymmetric Sarsa, and asymmetric policy gradient (algorithm 4, section 4.2). Each algorithm is trained until convergence on each POIBMDP, and each one of those runs is repeated 100 times. For all baselines we use, when applicable, exploration rates $\epsilon = 0.3$ and learning rates $\alpha = 0.1$. All the training curves are presented in appendix E.

[1] https://anonymous.4open.science/r/poibmdps-5BFE/

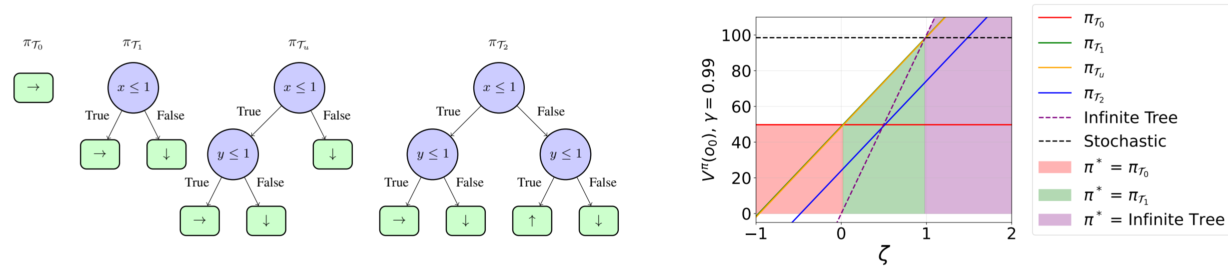


Figure 2. Decision tree policies and their RL objective values (definition 3.2) as functions of the rewards ζ in POIBMDPs associated to the grid world MDP (figure 1b). Shaded areas on the right plot indicate which deterministic memoryless policy is optimal depending on ζ. Recall that ζ can be seen as a reward that encourages the deterministic memoryless policy to take information-gathering actions, i.e. the reward that trades off interpretability of the corresponding decision tree and its cumulative reward.

We will consider two metrics. We consider the distribution of the learned trees over the 100 training seeds. Indeed, since for every POIBMDP we run each algorithm 100 times, at the end of training we get 100 deterministic memoryless policies, from which we can extract the equivalent 100 decision tree policies using algorithm 1 and we can count which one have e.g. a depth of 1. This helps understand which trees RL algorithms tend to learn as a function of the trade-off reward ζ.

5.3. Can reinforcement learning find the optimal deterministic memoryless POIBMDP policies?

In figure 13, we plot the distributions of the final learned trees over the 100 random seeds in function of ζ from the above runs. For example, in figure 13, in the top left plot, when learning 100 times in a POIBMDP with $\zeta = 0.5$, Q-learning returned almost 100 times a depth-0 tree. Again, on none of those subplots do we see a high rate of learned depth-1 trees for $\zeta \in]0, 1[$. It is alerting that the most frequent learned trees are the depth-0 trees for $\zeta \in]0, 1[$ because such trees are way more sub-optimal than e.g. the depth-2 unbalanced trees (figure 2). One interpretation of this phenomenon is that the learning in POIBMDPs is very difficult and so agents tend to converge to trivial policies, e.g., repeating the same downstream action. Furhtermore, in appendix E we show that for POIBMDPs with $\zeta \in [-1, 0]$ and $\zeta \in [1, 2]$, baselines consistently learn the optimal policies–a depth-0 tree and an infinite tree respectively–which is concerning as it means that RL seem to find only trivial policies. On the positive side, we observe that asymmetric versions of Q-learning and Sarsa have found the optimal deterministic memoryless policy–the depth-1 decision tree–more frequently throughout the optimality range $]0, 1[$, than their symmetric counter-parts for $\zeta \in]0, 1[$. Next, we quantify how difficult it is to do RL to learn memoryless policies in POIBMDPs as opposed to standard Markovian polciies.

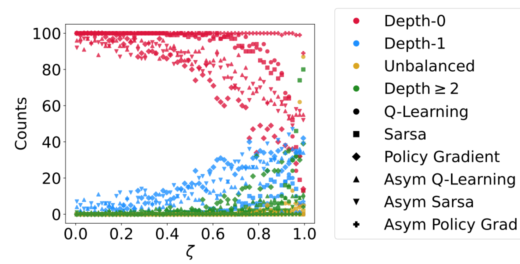


Figure 3. Distributions of final tree policies learned across the 100 seeds. For each ζ value, there are four colored points. Each point represent the share of depth-0 trees (red), depth-1 trees (green), unbalanced depth-2 trees (orange) and depth-2 trees (blue).

5.4. How difficult is it to learn in POIBMDPs?

In this section we run the same (asymmetric) reinforcement learning algorithms to learn standard Markovian policies in MDPs (definition 3.1) or IBMDPs (definition 3.5), or deterministic memoryless policies in POIBMDPs (definition 4.2).

In order to see how difficult each of these three problems is, we can run a *great* number of experiments for each problem and compare solving rates. To make solving rates comparable we consider a unique instance for each of those problems. Problem 1 is learning one of the optimal standard Markovian deterministic policy ($\pi : S \rightarrow A$)from figure 1a for the grid world MDP with $\gamma = 0.99$. Problem 2 is learning one of the optimal standard Markovian deterministic policy ($\pi : S \times O \rightarrow A \cup A_{info}$) for the IBMDP from figure 1b with $\gamma = 0.99$ and $\zeta = 0.5$. Problem 3 is what has been done in the previous section to learn deterministic memoryless policies ($\pi : O \rightarrow A \cup A_{info}$) where in addition of fixing $\gamma = 0.99$ we also fix $\zeta = 0.5$.

We use the six (asymmetric) RL algorithms from the previous section and try a wide set of hyperparameters and additional learning tricks (optimistic Q-function, eligibility traces, entropy regularization and ϵ-decay, all are described in (**?**)). The complete detailed lists of hyperparameters are

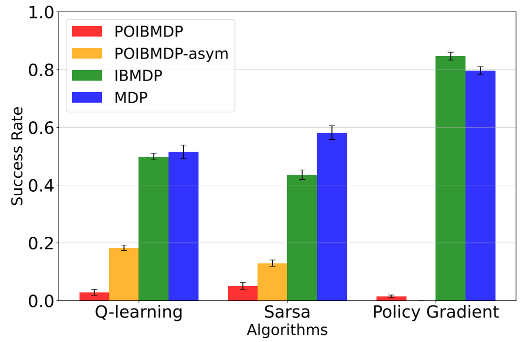


Figure 4. Success rates of different (asymmetric) RL algorithms over thousands of runs when applied to learning deterministic memoryless policies in a POIBMDP or learning deterministic policies in associated MDP and IBMDP.

given in the appendix F and a summary is given in table 6. Furthermore, the careful reader might notice that there is no point running asymmetric RL on MDPs or IBMDPs when the problem does not require partial observability. Hence, we only run asymmetric RL for POIBMDPs and otherwise run all other RL algorithms and all problems.

Each unique hyperparameter combination for a given algorithm on a given problem is run 10 times on 1 million learning steps to get standard errors. For example, for asymmetric Sarsa, we run a total of $10 \times 768 = 7680$ experiments for learning deterministic memoryless policies for a POIBMDP. To get a success rate, we can simply divide the number of learned optimal depth-1 tree by 768 (recall that for $\gamma = 0.99$ and $\zeta = 0.5$, the optimal policy is a depth-1 tree (e.g. figure 2).

The key observations from figure 4 is that reinforcement learning a deterministic memoryless policy in a POIBMDP, is way harder than learning a standard Markovian policy. For example, Q-learning finds the optimal solution in only 3% of the experiments while the same algorithms to optimize the standard RL objective (definition 3.2) in an MDP or IBMDP found the optimal solutions 50% of the time. Even though asymmetry seems to increase performances; learning a decision tree policy for a simple grid world directly with RL using the framework of POIBMDP originally developed in **?** seems way too difficult and costly as successes might require a million steps for such a seemingly simple problem. An other difficulty in practice that we did not cover here, is the choice of information gathering actions. For the grid world MDP, choosing good IGAs ($x \leq 1$ and $y \leq 1$) is simple but what about more complicated MDPs: how to instantiate the (PO)IBMDP action space such that internal nodes in resulting trees are useful for predictions? Next, we further support that partial observability is the main limitation for reinforcement learning to train decision tree policies for MDPs.

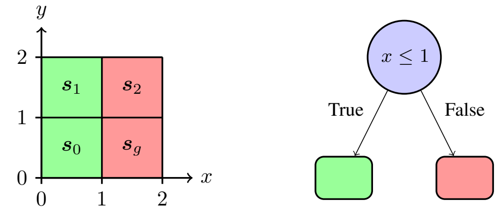


Figure 5. In this MDP, there are four data to which to assign either a green or red label. On the right, there is the unique optimal depth-1 tree for this particular MDP. This depth-1 tree also maximizes the accuracy on the corresponding classification task.

5.5. Are they downstream MDPs for which partial observability is not a limitation?

In this section, we show that for a special class of POIBMDPs, reinforcement learning can learn optimal deterministic memoryless policies w.r.t the RL objective, i.e. we can do direct decision tree policy learning for MDPs. This class of POIBMDPs are those for which downstream MDPs have uniform transitions, i.e. $T(\boldsymbol{s}, a, \boldsymbol{s}') = \frac{1}{|S|}$ (definitions 3.1 and 3.5). Such downstream MDPs include classification tasks formulated as MDPs. This implies that learning deterministic memoryless policies in POIBMDPs where the downstream MDP encodes a classification task is equivalent to doing supervised learning of a decision tree (figure 5). This is exactly what is done in e.g. **?**. In figure 5 we give an example of such downstream MDPs for a classification task with 4 data in the training set and 2 classes: $\mathcal{X} = \{(0.5, 0.5), (0.5, 1.5), (1.5, 1.5), (1.5, 0.5)\}$ and $y = \{0, 0, 1, 1\}$.

In appendix G, we show that POIBMDPs associated to downstream MDPs with uniform transitions are themselves standard MDP (definition 3.1). This means that in principle, standard RL algorithms like Q-learning, should work as well as for any MDP. If RL does work for such fully observable POIBMDPs, this would mean that the difficulty of direct learning of decision tree policies for *any* MDP using POIBMDPs, exhibited in sections, is most likely due to the partial observability. This is exactly what we check next. We use the same direct approach to learn decision tree policies as in previous sections, except that now the downstream MDP is a classification task and not a sequential decision making task like reaching a goal in a grid world.

We construct POIBMDPs for the classification task from figure 5, with $\gamma = 0.99$, 200 values of $\zeta \in [0, 1]$ and IGAs $x \leq 1$ and $y \leq 1$. Since those POIBMDPs are MDPs, we do not need to analyze asymmetric RL baselines. We see on figure 6 that compared to general POIBMDPs from previous sections, RL can be used to consistently learn optimal deterministic memoryless policies $O :\rightarrow A \cup A_{info}$. Such policies are equivalent to decision tree classifiers.

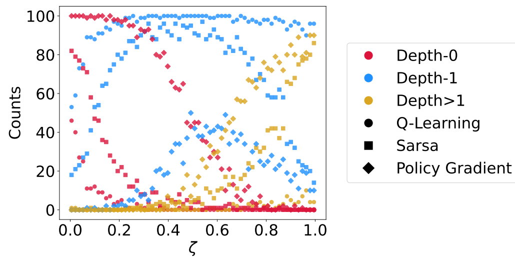


Figure 6. We reproduce the same plot as in figure 13 for POIB-MDPs associated to a downstream MDP encoding a classification task. Each colored dot is the number of final learned trees with a specific structure for a given ζ.

6. Discussion

In this paper, we were interested in algorithms that can learn decision tree policies that directly optimize some trade-off of interpretability and performance in MDPs without relying on an oracle or expert policy. Starting from the IBMDP formulation from **?**, we have shown that direct learning of decision tree policies for MDPs can be framed as learning deterministic memoryless policies in POMDPs that we called POIBMDPs. By bridging the gap with the POMDP literature, we conjectured that partial observability is the main limitation for reinforcement learning of decision tree policies for MDPs. We then supported this conjecture by benchmarking different reinforcement learning algorithms on carefully crafted problems for which we knew the exact optimal decision tree policies. Across our experiments, we found that only when partial observablity is absent from the learning task can good decision tree policies be trained. Attempting to overcome the partial observability challenges highlighted so far seems like a bad research direction. Indeed, while algorithms tailored specifically for the problem of learning deterministic memoryless policies for POIB-MDPs might exist, imitation learning works well in practice and has been the state-of-the-art for interpretable sequential decision making for a while. Furthermore there are other limitations that we did not cover in the framework of **?** such as how to choose good candidates information-gathering actions or simply how to choose ζ for a target interpretability-performance trade-off. Finally, while we focused on non-parametric tree learning assuming RL algorithms should naturally trade off interpretability and performance through the reward signal ζ for adding nodes to the decision tree policy, another future research avenue could be to develop better algortihms training parametric trees. Indeed parametric tree policies, on the other hand, can be computed with reinforcement learning directly in the downstream MDP. However such RL algorithms for parametric decision tree policies (**???**) require to re-train a policy entirely for each desired level of interpretability, i.e. each unique tree structure. Future research in this direction should focus on algorithms for parametric tree policies that can re-use samples from one tree learning to train a different tree structure more efficiently. This would reduce the required quantity of a priori knowledge on the decision tree policy structure.

Impact statement

Our work put great emphasis on covering existing work, open sourcing code, and being transparent about limitations. We hope that the impact of our work is going to be positive for society through advancing reseach in interpretable machine learning.

Algorithm 1 Extract a decision tree policy (algorithm 1 from **?**)

Data: Deterministic partially observable policy π_{po} for IBMDP $\mathcal{M}_{IB} \equiv \langle S \times O, A \cup A_{info}, (R, \zeta), (T_{info}, T, T_0) \rangle$ and IBMDP observation $\boldsymbol{o} = (L'_1, U'_1, \dots, L'_p, U'_p)$

Result: Decision tree policy $\pi_{\mathcal{T}}$ for MDP $\mathcal{M} \equiv \langle S, A, R, T, T_0 \rangle$

Function `Subtree_From_Policy(`$\boldsymbol{o}, \pi_{po}$`)`**:**

- $a \leftarrow \pi_{po}(\boldsymbol{o})$
- **if** *a is a downstream action* **then**
  - **return** Leaf_Node(action: a)
- **end**
- **else**
  - $\langle i, v \rangle \leftarrow a$
  - $\boldsymbol{o}_L \leftarrow \boldsymbol{o}$; $\boldsymbol{o}_R \leftarrow \boldsymbol{o}$
  - $\boldsymbol{o}_L \leftarrow (L'_1, U'_1, \dots, L'_j, v, \dots, L'_p, U'_p)$; $\boldsymbol{o}_R \leftarrow (L'_1, U'_1, \dots, v, U'_j, \dots, L'_p, U'_p)$
  - $child_L \leftarrow$ Subtree_From_Policy($\boldsymbol{o}_L, \pi_{po}$)
  - $child_R \leftarrow$ Subtree_From_Policy($\boldsymbol{o}_R, \pi_{po}$)
  - **return** Internal_Node(feature: i, value: v, children: $(child_L, child_R)$)
- **end**

A. From a policy to a tree

B. Imitation learning: a baseline for indirect decision tree policy learning

In this section we present decision tree policies of this manuscript obtained using Dagger or VIPER (**??**) after learning an expert Q-function for the grid world MDP. Recall the optimal policies for the grid world, taking the green actions in each state in figure 1a. Among the optimal policies, the ones that go left or up in the goal state can be problematic for imitation learning algorithms. Indeed, we know that for this grid world MDP there exists decision tree policies with a very good interpretability-performance trade-off: depth-1 decision trees that are optimal w.r.t. the RL objective. One could even say that those trees have the *optimal* interpretability-performance trade-off because they are the shortest trees that are optimal w.r.t. the RL objective.

In figure 7, we present a depth-1 decision tree policy that is optimal w.r.t. the RL objective and a depth-1 tree that is sub-optimal. The other optimal depth-1 tree is to go right when $y \leq 1$ and down otherwise.

Now a fair question is: can Dagger or VIPER learn such an optimal depth-1 tree given access to an expert optimal policy from figure 1a?

We start by running the standard Q-learning algorithm with $\epsilon = 0.3$, $\alpha = 0.1$ over 10,000 time steps. While for Q-learning, Sutton and Barto break ties by index value in their book (**?**) (the greedy action is the argmax action with smallest index), we show that the choice of tie-breaking greatly influences the performance of subsequent imitation learning algorithms. Indeed, depending on how actions are ordered in practice, Q-learning may be biased toward some optimal policies rather than others. While this does not matter for one who just wants to find an optimal policy, in our example of finding the optimal depth-1 decision tree policy, it matters *a lot*.

In the left plot of figure 8, we see that Q-learning, independently of how ties are broken, consistently converges to an optimal policy over 100 runs (random seeds). However, in the right plot of figure 8, where we plot the proportion over 100 runs of optimal decision trees returned by Dagger or VIPER at different stages of Q-learning, we observe that imitating the optimal policy obtained by breaking ties at random consistently yields more optimal trees than breaking ties by indices. What actually happens is that the most likely output of Q-learning when ties are broken by indices is the optimal policy that goes left in the goal state, which cannot be perfectly represented by a depth-1 decision tree, because there are three different actions taken and a binary tree of depth $D = 1$ can only map to $2^D = 2$ labels.

This short experiment shows that imitation learning approaches can sometimes be very bad at learning decision tree policies with good interpretability-performance trade-offs for very simple MDPs. Despite VIPER almost always finding the optimal depth-1 decision tree policy in terms of the RL objective when ties are broken at random, we have shed light on the sub-optimality of indirect approaches such as imitation learning. This motivates the study of direct approaches to directly search for policies with good interpretability-performance trade-offs with respect to the original RL objective.

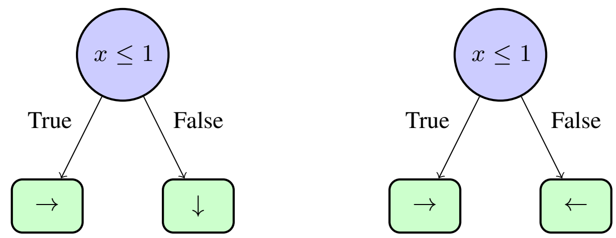


Figure 7. Left, an optimal depth-1 decision tree policy for the grid world MDP from figure 1a. On the right, a sub-optimal depth-1 decision tree policy.

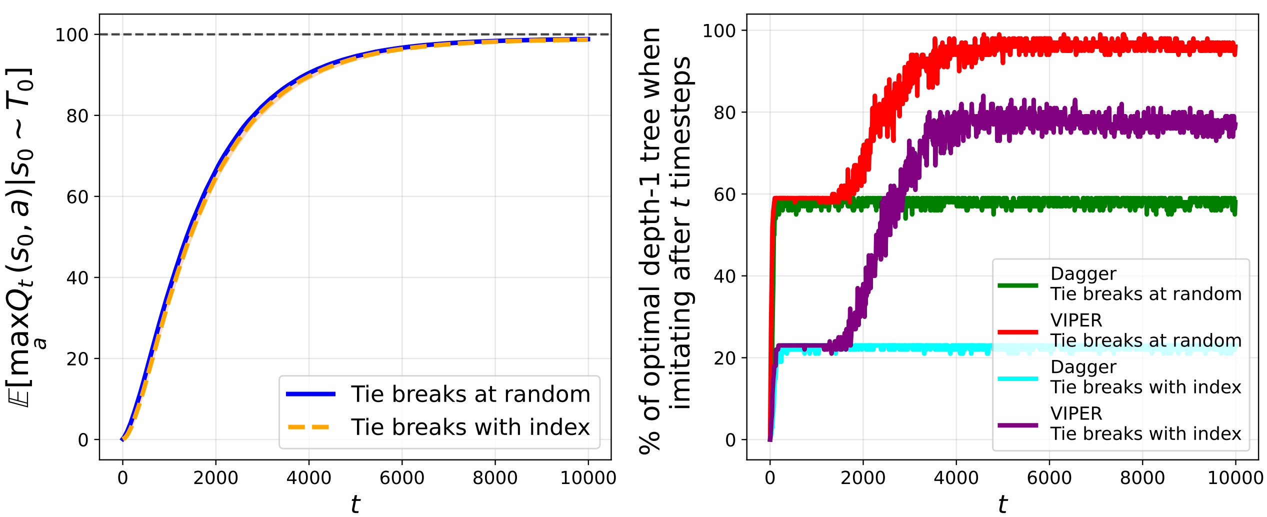


Figure 8. Left, sample complexity curve of Q-learning with default hyperparameters on the 2×2 grid world MDP over 100 random seeds. Right, performance of indirect interpretable methods when imitating the greedy policy with a tree at different Q-learning stages.

C. Reproducing "Iterative Bounding MDPs: Learning Interpretable Policies via Non-Interpretable Methods"

We attempt to reproduce the results from (?) in which authors compare direct and indirect learning of decision tree policies of depth at most 2 for the CartPole MDP (?). In the original paper, the authors find that both direct and indirect learning yields decision tree policies with similar RL objective values (definition 3.2) for the CartPole. On the other hand, we find that, imitation learning, despite not directly optimizing the RL objective for CartPole, outperforms deep RL which directly optimizes a trade-off of the standard RL objective and interpretability.

Authors of ? use two deep reinforcement learning baselines to which they apply some modifications in order to learn memoryless policies. Authors modify the standard DQN (?) to return a memoryless policy. The trained Q-function is approximated with a neural network $O \to \mathbb{R}^{|A \cup A_{info}|}$ rather than $S \times O \to \mathbb{R}^{|A \cup A_{info}|}$. In this modified DQN, the temporal difference error target for the Q-function $O \to A \cup A_{info}$ is approximated by a neural network $S \times O \to A \cup A_{info}$ that is in turn trained by bootstrapping the temporal difference error with itself. We present the modifications in algorithm 2. Similar modifications are applied to the standard PPO (?) that we present in the appendix (algorithm 3). In the modified PPO, neural network policy $O \to A \cup A_{info}$ is trained using a neural network value function $S \times O \to A \cup A_{info}$ as a critic.

Those two variants of DQN and PPO have first been introduced in (?) for robotic tasks with partially observable components, under the name "asymmetric" actor-critic. Asymmetric RL algorithms that have policy and value estimates using different information from a POMDP (??) were later studied theoretically to solve POMDPs in Baisero's work (??). The connections from Deep RL in IBMDPs for objective is absent from (?) and we defer their connections to direct interpretable reinforcement learning to the next chapter as our primary goal is to reproduce (?) *as is*. Next, we present the precise experimental setup we use to reproduce (?) in order to study direct deep reinforcement learning of decision tree policies for the CartPole MDP.

C.1. IBMDP formulation

Given a base MDP $\mathcal{M} \equiv \langle S, A, R, T, T_0 \rangle$ (cf. definition 3.1), in order to define an IBMDP $\mathcal{M}_{IB} \langle S \times O, A \cup A_{info}, (R, \zeta), (T, T_0, T_{info}) \rangle$ (cf. definition 3.5), the user needs to provide the set of information gathering actions A_{info} and the reward ζ for taking those. Authors of topin2021iterative propose to parametrize the set of IGAs with $j \times q$ actions $\langle j, v_k \rangle$ with v_k depending on the current observation $\boldsymbol{o}_t = (L'_1, U'_1, \ldots, L'_j, U'_j, \ldots, L'_p, U'_p)$: $v_k = \frac{k(U'_j - L'_j)}{q+1}$. This parametric IGAs space keeps the discrete IBMDP action space at a reasonable size while providing a learning algorithm with varied IGAs to try.

For example, if we define an IBMDP with $q = 3$ for the grid world from figure 1a, the grid world action space is augmented with six IGAs. At $t = 0$, recall that $\boldsymbol{o}_0 = (0, 2, 0, 2)$, so if an IGA is taken, e.g. $\langle 2, v_2 \rangle$, the effective IGA is $\langle j, v_2 = \frac{k(2-0)}{3+1} \rangle = \langle 1, 2 \rangle$ which in turn effectively corresponds to an internal decision tree node $y \leq 1$. If the current state y-feature value is 0.5, then the next observation at $t = 1$ is $\boldsymbol{o}_1 = (0, 2, 0, 1)$. At $t = 2$ if $a_t = \langle 2, v_2 \rangle$ again, it would be effectively $\langle j, v_2 = \frac{k(1-0)}{3+1} \rangle = \langle 2, 0.5 \rangle$. This would give the next observation at $t = 2$ $\boldsymbol{o}_2 = (0, 2, 0, 0.5)$ and so on.

Furthermore, author propose to regularize the learned decision tree policy with a maximum depth parameter D. Unfortunately, the authors did not describe how they implemented the depth control in their work, hence we have to try different approaches to reproduce their results.

To control the tree depth during learning in the IBMDP, we can either give negative reward for taking D IGAs in a row, or terminate the trajectory. In practice, we could also have a state-dependent action space such that taking an IGA is not allowed after taking D IGAs in a row. The latter approach–sometimes called action masking–is not compatible with the definition of an MDP (cf. definition 3.1) in which all actions are available in all states. To apply the penalization approaches, one can extend the MDP states to keep track of the current tree depth. Similarly, the termination approach requires a transition function that depends on the current tree depth.

We actually find that when $q+1$, the parameter that defines threshold values in decision tree policy nodes (cf. definition 3.5), is a prime number, then as a direct consequence of the *Chinese Remainder Theorem*[2], the current tree depth is directly encoded in the current observation $\boldsymbol{o}_t$. Hence, when $q+1$ is prime, we can control the depth through either transitions or rewards without tracking the tree depth. We use the exact same downstream MDP and associated IBMDPs for our

[2] https://en.wikipedia.org/wiki/Chinese_remainder_theorem

experiments as (?) except when mentioned otherwise.

downstream MDP The task at hand is to optimize the RL objective (definition 3.2) with a decision tree policy for the CartPole MDP (?). At each time step a learning algorithm observes the cart's position and velocity and the pole's angle and angular velocity, and can take action to push the CartPole left or right. While the CartPole is roughly balanced, i.e., while the cart's angle remains in some fixed range, the agent gets a positive reward. If the CartPole is out of balance, the MDP transitions to an absorbing terminal state and gets 0 reward forever. Like in (?), we use the gymnasium `CartPole-v0` implementation (?) of the CartPole MDP in which trajectories are truncated after 200 timesteps making the maximum cumulative reward, i.e. the optimal value of the RL objective when $\gamma = 1$, to be 200. The state features of the CartPole MDP are in $[-2, 2] \times [-2, 2] \times [-0.14, 0.14] \times [-1.4, 1.4]$.

IBMDP Authors define the associated IBMDP (definition 3.5) with $\zeta = -0.01$ and 4 information gathering actions. In addition to the original IBMDP paper, we also try $\zeta = 0.01$ and 3 information gathering actions. We use the same discount factor as the authors: $\gamma = 1$. We try two different approaches to limit the depth of decision tree policies to be at most 2: terminating trajectories if the agent takes too many information gathering actions in a row or simply giving a reward of -1 to the agent every time it takes an information gathering action past the depth limit. In practice, we could have tried an action masking approach, i.e. having a state dependent-action set, but we want to abide to the MDP formalism in order to properly understand direct interpretable approaches. We will also try IBMDPs where we do not limit the maximum depth for completeness.

Table 1. IBMDP hyperparameters. We try 12 different IBMDPs. In green we highlight the hyperparameters from the original paper and in red we highlight the hyperparameter names for which author do not give information.

Hyperparameter	**Values**
Discount factor γ	1
Information gathering actions parameter q	2, 3
Information gathering actions rewards ζ	-0.01, 0.01
Depth control	Done signal, negative reward, none

Table 2. (Modified) DQN trained on 10^6 timesteps. This gives four different instantiation of (modified) DQN. Hyperparameters not mentioned are stable-baselines3 default. In green we highlight the hyperparameters from the original paper and in red we highlight the hyperparameter names for which author do not give information.

Hyperparameter	**Values**
Buffer size	10^6
Random transitions before learning	10^5
Epsilon start	0.9, 0.5
Epsilon end	0.05
Exploration fraction	0.1
Optimizer	RMSprop ($\alpha = 0.95$)
Learning rate	2.5×10^{-4}
Networks architectures	[128, 128]
Networks activation	tanh(), relu()

Table 4. Top 5 hyperparameter configurations for modified DQN + IBMDP, bold font represent the original paper hyperparameters.

Rank	q	Depth control	Activation	Exploration	ζ	Mean Final Performance
1	3	termination	tanh	0.9	0.01	53
2	2	termination	tanh	0.5	-0.01	24
3	**3**	**termination**	tanh	**0.5**	**-0.01**	**24**
4	2	termination	tanh	0.5	0.01	23
5	2	termination	tanh	0.9	-0.01	22

Table 5. Top 5 hyperparameter configurations for modified PPO + IBMDP, bold font represent the original paper hyperparameters.

Rank	q	Depth Control	Activation	ζ	Mean Final Performance
1	3	reward	relu	0.01	139
2	3	termination	relu	0.01	132
3	**3**	**reward**	tanh	**-0.01**	**119**
4	3	reward	relu	-0.01	117
5	3	reward	tanh	0.01	116

Table 3. (Modified) PPO trained on 4×10^6 timesteps. This gives two different instantiation of (modified) PPO. Hyperparameters not mentioned are stable-baselines3 default. In green we highlight the hyperparameters from the original paper and in red we highlight the hyperparameter names for which author do not give information.

Hyperparameter	**Values**
Steps between each policy gradient steps	512
Number of minibatch for policy gradient updates	4
Networks architectures	[64, 64]
Networks activations	$\tanh()$, $\mathrm{relu}()$

In tables 4 and 5 we report the top-5 hyperparameters for Modified RL baselines when learning partially observable IBMDP policies in terms of extracted decision tree policies performances in the CartPole MDP.

C.2. Experimental setup

Modified DQN and Modified PPO as mentioned above, the authors use the modified version of DQN from algorithm 2. We use the exact same hyperparameters for modified DQN as the authors when possible. We use the same layers width (128) and number of hidden layers (2), the same exploration strategy (ϵ-greedy with linearly decreasing value ϵ between 0.5 and 0.05 during the first 10% of the training), the same replay buffer size (10^6) and the same number of transitions to be collected randomly before doing value updates (10^5). We also try to use more exploration during training (change the initial ϵ value to 0.9). We use the same optimizer (RMSprop with hyperparameter 0.95 and learning rate 2.5×10^{-4}) to update the Q-networks. Authors did not share which DQN implementation they used so we use the stable-baselines3 one (**?**)[3]. Authors did not share which activation functions they used so we try both $\tanh$ and relu. For the modified PPO algorithm (algorithm 3), we can exactly match the authors hyperparameters since they use the open source stable-baselines3 implementation of PPO. We match training budgets: we train modified DQN on 1 million timesteps and modified PPO on 4 million timesteps.

DQN and PPO We also benchmark the standard DQN and PPO when learning standard Markovian IBMDP policies $\pi : S \times O \to A \cup A_{info}$ and when learning standard $\pi : S \to A$ policies directly in the CartPole MDP. We summarize hyperparameters for the IBMDP and for the learning algorithms in appendices 1, 2 and 3.

Indirect methods We also compare modified RL algorithm to imitation learning. To do so, we use VIPER or Dagger to imitate greedy neural network policies obtained with standard DQN learning directly on CartPole. We use Dagger to imitate

[3]We are cleaning our source code and will open source it as soon as possible

Algorithm 2 Modified Deep Q-Network. We highlight in green the changes to the standard DQN (**?**).

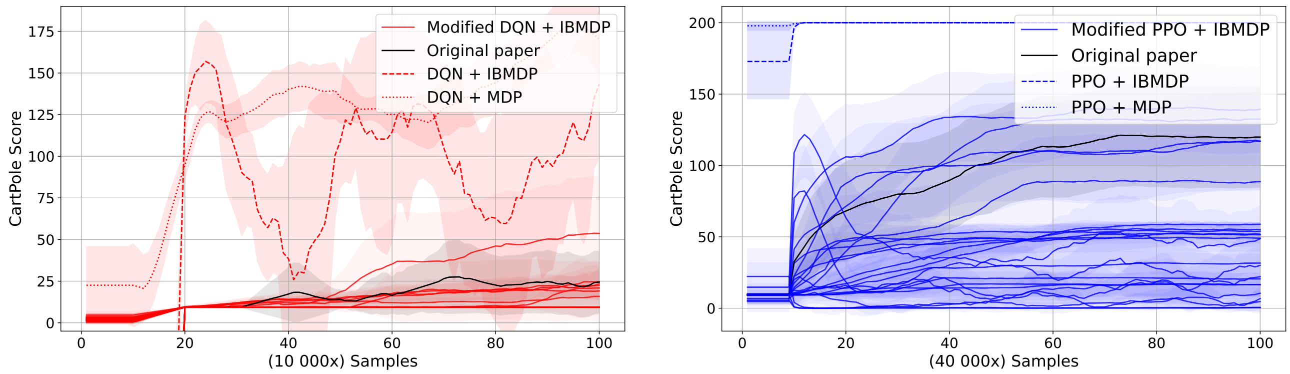
Data: IBMDP $\mathcal{M}_{IB}\langle S \times O, A \cup A_{info}, (R, \zeta), (T, T_0, T_{info})\rangle$, learning rate α, exploration rate ϵ, partially observable Q-network parameters θ, Q-network parameters ϕ, replay buffer $\mathcal{B}$, update frequency C
Result: deterministic memoryless policy π_{po}
Initialize partially observable Q-network parameters θ
Initialize Q-network parameters ϕ and target network parameters $\phi^- = \phi$

Initialize replay buffer $\mathcal{B} = \emptyset$
for *each episode* **do**
  Initialize downstream state features $\boldsymbol{s}_0 \sim T_0$
  Initialize observation $\boldsymbol{o}_0 = (L_1, U_1, \dots, L_p, U_p)$

  for *each step* t **do**
    Choose action a_t using ϵ-greedy: $a_t = \operatorname{argmax}_a Q_\theta(\boldsymbol{o}_t, a)$ with prob. $1 - \epsilon$
    Take action a_t, observe r_t
    Store transition $(\boldsymbol{s}_t, \boldsymbol{o}_t, a_t, r_t, \boldsymbol{s}_{t+1})$ in $\mathcal{B}$
    Sample random batch $(\boldsymbol{s}_i, \boldsymbol{o}_i, a_i, r_i, \boldsymbol{s}_{i+1}) \sim \mathcal{B}$
    $a' = \underset{a}{\operatorname{argmax}} Q_\theta(\boldsymbol{o}_i, a)$
    $y_i = r_i + \gamma Q_{\phi^-}(\boldsymbol{s}_{i+1}, a')$ // Compute target $\phi \leftarrow \phi - \alpha \nabla_\phi (Q_\phi(\boldsymbol{s}_i, a_i) - y_i)^2$ // Update Q-network $\theta \leftarrow \theta - \alpha \nabla_\theta (Q_\theta(\boldsymbol{o}_i, a_i) - y_i)^2$ // Update partially observable Q-network
    if $t \bmod C = 0$ **then**
      $\theta^- \leftarrow \theta$ // Update target network
    end
    $\boldsymbol{s}_t \leftarrow \boldsymbol{s}_{t+1}$
    $\boldsymbol{o}_t \leftarrow \boldsymbol{o}_{t+1}$
  end
end
$\pi_{po}(\boldsymbol{o}) = \operatorname{argmax}_a Q_\theta(\boldsymbol{o}, a)$ // Extract greedy policy

Algorithm 3 Modified Proximal Policy Optimization

Data: IBMDP $\mathcal{M}_{IB}\langle S \times O, A \cup A_{info}, (R, \zeta), (T, T_0, T_{info})\rangle$, learning rate α, policy parameters θ, clipping parameter ϵ, value function parameters ϕ
Result: Memoryless stochastic policy π_{po_θ}
Initialize policy parameters θ and value function parameters ϕ
for *each episode* **do**
  Generate trajectory $\tau = (\boldsymbol{s}_0, \boldsymbol{o}_0, a_0, r_0, \boldsymbol{s}_1, \boldsymbol{o}_1, a_1, r_1, \dots)$ following π_θ
  for *each timestep* t *in trajectory* **do**
    $G_t \leftarrow \sum_{k=t}^{T} \gamma^{k-t} r_k$ // Compute return $A_t \leftarrow G_t - V_\phi(\boldsymbol{s}_t)$ // Compute advantage $r_t(\theta) \leftarrow \frac{\pi_{po_\theta}(a_t|\boldsymbol{o}_t)}{\pi_{po_\theta\, old}(a_t|\boldsymbol{o}_t)}$ // Compute probability ratio $L_t^{CLIP} \leftarrow \min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)$ // Clipped objective $\theta \leftarrow \theta + \alpha \nabla_\theta L_t^{CLIP}$ // Policy update $\phi \leftarrow \phi + \alpha \nabla_\phi (G_t - V_\phi(\boldsymbol{s}_t))^2$ // Value function update
  end
  $\theta_{old} \leftarrow \theta$ // Update old policy
end



(a) DQN variants *(b)* PPO variants

Figure 9. Comparison of modified reinforcement learning algorithms on different CartPole IBMDPs. (a) Shows variations of modified DQN and DQN (table 2), while (b) shows variations of modified PPO and PPO (table 3). For both algorithms, we give different line-styles for the learning curves when applied directly on the CartPole MDP versus when applied on the IBMDP to learn standard Markovian policies. We color the modified RL algorithm variant from the original paper in black. Shaded areas represent the confidence interval at 95% at each measure on the y-axis.

neural network policies obtained with the standard PPO learning directly on CartPole. For each indirect method, we imitate the neural network experts by fitting decision trees on 10000 expert transitions using the CART (**?**) implementation from scikit-learn (**?**) with default hyperparameters and maximum depth of 2 like in (**?**).

C.2.1. Metrics

The key metric of this section is performance when controlling the CartPole, i.e, the average *undiscounted* cumulative reward of a policy on 100 trajectories (RL objective with $\gamma = 1$). For modified RL algorithms that learn a memoryless policy (or Q-function) in an IBMDP, we periodically extract the policy (or Q-function) and use algorithm 1 to extract a decision tree for the CartPole MDP. We then evaluate the tree on 100 independent trajectories in the MDP and report the mean undiscounted cumulative reward. For RL applied to IBMDPs, since we can't deploy learned policies directly to the downstream MDP as the state dimensions mismatch–such policies are $S \times O \rightarrow A \cup A_{info}$ but the MDP states are in S–we periodically evaluate those IBMDP policies in a copy of the IBMDP in which we fix $\zeta = 0$ ensuring that the copied IBMDP undiscounted cumulative rewards only account rewards from the CartPole MDP (non-zero rewards in the IBMDP only occur when a reward from the downstream MDP is given, i.e. when $a_t \in A$ in the IBMDP (definition 3.5)). Similarly, we do 100 trajectories of the extracted policies in the copied IBMDP and report the average undiscounted cumulative reward. For RL applied directly to the downstream MDP we can just periodically extract the learned policies and evaluate them on 100 CartPole trajectories.

Since imitation learning baselines train offline, i.e, on a fixed dataset, their performances cannot directly be reported on the same axis as RL baselines. For that reason, during the training of a standard RL baseline, we periodically extract the trained neural policy/Q-function that we consider as the expert to imitate. Those experts are then imitated with VIPER or Dagger using 10 000 newly generated transitions and then fitsted decision tree policies are then evaluated on 100 CartPole trajectories. We do not report the imitation learning objective values during VIPER or Dagger training. Every single combination of IBMDP and Modified RL hyperparameters is run 20 times. For standard RL on either an IBMDP or an MDP, we use the paper original hyperparameters when they were specified, with depth control using negative rewards, $\tanh()$ activations. We use 20 individual random seeds for every experiment in this chapter. Next, we present our results when reproducing (**?**).

C.3. Results

C.3.1. How well do modified deep RL baselines learn in IBMDPs?

On figure 9a, we observe that modified DQN can learn in IBMDPs–the curves have an increasing trend–but we also observe that modified DQN finds poor decision tree policies for the CartPole MDP in average–the curves flatten at the end of the x-axis and have low y-values. In particular, the highest final y-value, among all the learning curves that could possibly

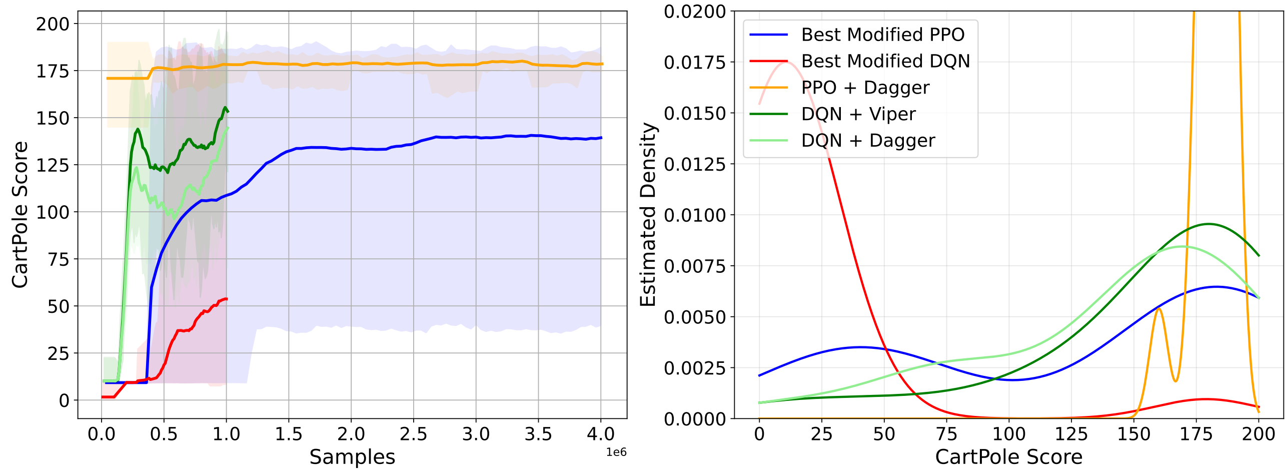


Figure 10. (left) Mean performance of the best–w.r.t. the RL objective for CartPole–modified RL + IBMDP combination. Shaded areas represent the min and max performance over the 20 seeds during training. (right) Corresponding score distribution of the final decision tree policies w.r.t. the RL objective for CartPole.

correspond to the original paper modified DQN, correspond to poor performances on the CartPole MDP. On figure 9b, we observe that modified PPO finds decision tree policies with almost 150 cumulative rewards towards the end of training. The performance difference with modified DQN could be because we trained modified PPO longer, like in the original paper. However it could also be because DQN-like algorithms with those hyperparameters struggle to learn in CartPole (IB)MDPs. Indeed, we notice that for DQN-like baselines, learning seems difficult in general independently of the setting. On figures 9a and 9b, we observe that standard RL baselines (RL + IBMDP and RL + MDP), learn better CartPole policies in average than their modified counterparts that learn memoryless policies. On figure 9b, it is clear that for the standard PPO baselines, learning is super efficient and algorithms learn optimal policies with reward 200 in few thousands steps.

C.3.2. Which decision tree policies does direct reinforcement learning return for the CartPole MDP?

On figure 10, we isolate the best performing algorithms instantiations that learn decision tree policies for the CartPole MDP. We compare the best modified DQN and modified PPO to imitation learning baselines that use the surrogate imitation objective to find CartPole decision tree policies. We find that despite having poor performances in *average*, the modified deep reinforcement learning baselines can find very good decision tree policies as shown by the min-max shaded areas on the left of figure 10 and the corresponding estimated density of learned trees performances. However this is not desirable, a user typically wants an algorithm that can consistently find good decision tree policies. As shown by the estimated densities, indirect methods consistently find good decision tree policies (the higher modes of distributions are on the right of the plot). On the other hand, the decision tree policies returned by direct RL methods seem equally distributed on both extremes of the scores.

On figure 11, we present the best decision tree policies for CartPole returned by modified DQN and modified PPO. We used algorithm 1 to extract 20 trees from the 20 memoryless policies returned by the modified deep reinforcement learning algorithms over the 20 training seeds. We then plot the best tree for each baseline. Those trees get an average RL objective of roughly 175. Similarly, we plot a representative tree for imitation learning baseline as well as a tree that is optimal for CartPole w.r.t. the RL objective obtained with VIPER. Unlike for direct methods, the trees returned by imitation learning are extremely similar across seeds. In particular they often only vary in the scalar value used in the root node but in general have the same structure and test the angular velocity. On the other hand the most frequent trees across seeds returned by modified RL baselines are "trivial" decision tree policies that either repeat the same downstream action forever or repeat the same IGA (definition 3.5) forever.

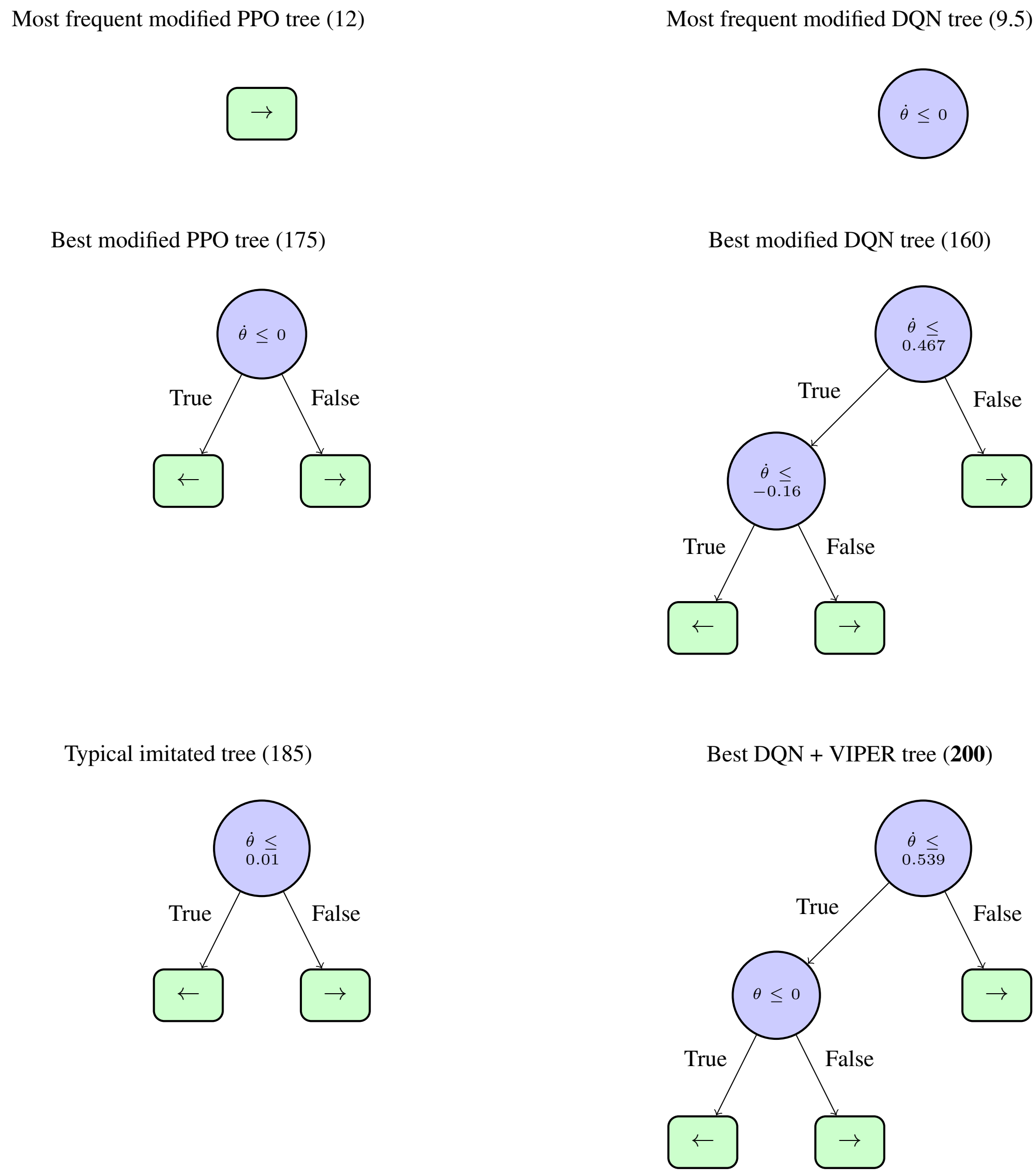


Figure 11. Trees obtained by modified deep RL in IBMDPs against trees obtained with imitation (RL objective value). θ and $\dot{\theta}$ are respectively the angle and the angular velocity of the pole

D. RL objective values calculations

Optimal depth-1 decision tree policy $\pi_{\mathcal{T}_1}$ has one root node that tests $x \leq 1$ (respectively $y \leq 1$) and two leaf nodes $\rightarrow$ and $\downarrow$. To compute $V^{\pi}_{\mathcal{T}_1}(\boldsymbol{o}_0)$, we compute the values of $\pi_{\mathcal{T}_1}$ in each of the possible starting states $(\boldsymbol{s}_0, \boldsymbol{o}_0), (\boldsymbol{s}_1, \boldsymbol{o}_0), (\boldsymbol{s}_2, \boldsymbol{o}_0), (\boldsymbol{s}_g, \boldsymbol{o}_0)$ and compute the expectation over those. At initialization, when the downstream state is $\boldsymbol{s}_g = (1.5, 0.5)$, the depth-1 decision tree policy cycles between taking an information gathering action $x \leq 1$ and moving down to get a positive reward for which it gets the returns:

$$\begin{aligned}
V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_g, \boldsymbol{o}_0) &= \zeta + \gamma + \gamma^2\zeta + \gamma^3 \dots \\
&= \sum_{t=0}^{\infty}\gamma^{2t}\zeta + \sum_{t=0}^{\infty}\gamma^{2t+1} \\
&= \frac{\zeta + \gamma}{1 - \gamma^2}
\end{aligned}$$

At initialization, in either of the downstream states $\boldsymbol{s}_0 = (0.5, 0.5)$ and $\boldsymbol{s}_2 = (1.5, 1.5)$, the value of the depth-1 decision tree policy is the return when taking one information gathering action $x \leq 1$, then moving right or down, then following the policy from the goal state $\boldsymbol{s}_g$:

$$\begin{aligned}
V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_0, o_0) &= \zeta + \gamma 0 + \gamma^2 V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_g, o_0) \\
&= \zeta + \gamma^2 V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_g, o_0) \\
&= V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_2, o_0)
\end{aligned}$$

Similarly, the value of the best depth-1 decision tree policy in state $\boldsymbol{s}_1 = (0.5, 1.5)$ is the value of taking one information gathering action then moving right to $\boldsymbol{s}_2$ then following the policy in $\boldsymbol{s}_2$:

$$\begin{aligned}
V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_1, \boldsymbol{o}_0) &= \zeta + \gamma 0 + \gamma^2 V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_2, \boldsymbol{o}_0) \\
&= \zeta + \gamma^2 V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_2, o_0) \\
&= \zeta + \gamma^2(\zeta + \gamma^2 V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_g, o_0)) \\
&= \zeta + \gamma^2\zeta + \gamma^4 V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_g, o_0)
\end{aligned}$$

Since the probability of being in any downstream states at initialization given that the observation is $\boldsymbol{o}_0$ is simply the probability of being in any downstream states at initialization, we can write:

$$\begin{aligned}
V^{\pi_{\mathcal{T}_1}}(\boldsymbol{o}_0) &= \frac{1}{4}V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_g, \boldsymbol{o}_0) + \frac{2}{4}V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_2, \boldsymbol{o}_0) + \frac{1}{4}V^{\pi_{\mathcal{T}_1}}(\boldsymbol{s}_1, \boldsymbol{o}_0) \\
&= \frac{1}{4}\frac{\zeta + \gamma}{1 - \gamma^2} + \frac{2}{4}(\zeta + \gamma^2\frac{\zeta + \gamma}{1 - \gamma^2}) + \frac{1}{4}(\zeta + \gamma^2\zeta + \gamma^4\frac{\zeta + \gamma}{1 - \gamma^2}) \\
&= \frac{1}{4}\frac{\zeta + \gamma}{1 - \gamma^2} + \frac{2}{4}(\frac{\zeta + \gamma^3}{1 - \gamma^2}) + \frac{1}{4}(\frac{\zeta + \gamma^5}{1 - \gamma^2}) \\
&= \frac{4\zeta + \gamma + 2\gamma^3 + \gamma^5}{4(1 - \gamma^2)}
\end{aligned}$$

Depth-0 decision tree: has only one leaf node that takes a single downstream action indefinitely. For this type of tree the best reward achievable is to take actions that maximize the probability of reaching the objective $\rightarrow$ or $\downarrow$. In that case the objective value of such tree is: In the goal state $G = (1, 0)$, the value of the depth-0 tree $\mathcal{T}_0$ is:

$$\begin{aligned}
V^{\mathcal{T}_0}_G &= 1 + \gamma + \gamma^2 + \dots \\
&= \sum_{t=0}^{\infty}\gamma^t \\
&= \frac{1}{1 - \gamma}
\end{aligned}$$

In the state $(0,0)$ when the policy repeats going right respectively in the state $(0,1)$ when the policy repeats going down, the value is:

$$\begin{aligned} V_{S_0}^{\mathcal{T}_0} &= 0 + \gamma V_g^{\mathcal{T}_0} \\ &= \gamma V_G^{\mathcal{T}_0} \end{aligned}$$

In the other states the policy never gets positive rewards; $V_{S_1}^{\mathcal{T}_0} = V_{S_2}^{\mathcal{T}_0} = 0$. Hence:

$$\begin{aligned} J(\mathcal{T}_0) &= \frac{1}{4}V_G^{\mathcal{T}_0} + \frac{1}{4}V_{S_0}^{\mathcal{T}_0} + \frac{1}{4}V_{S_1}^{\mathcal{T}_0} + \frac{1}{4}V_{S_2}^{\mathcal{T}_0} \\ &= \frac{1}{4}V_G^{\mathcal{T}_0} + \frac{1}{4}\gamma V_G^{\mathcal{T}_0} + 0 + 0 \\ &= \frac{1}{4}\frac{1}{1-\gamma} + \frac{1}{4}\gamma\frac{1}{1-\gamma} \\ &= \frac{1+\gamma}{4(1-\gamma)} \end{aligned}$$

Unbalanced depth-2 decision tree: the unbalanced depth-2 decision tree takes an information gathering action $x \leq 0.5$ then either takes the $\downarrow$ action or takes a second information $y \leq 0.5$ followed by $\rightarrow$ or $\downarrow$. In states G and S_2, the value of the unbalanced tree is the same as for the depth-1 tree. In states S_0 and S_1, the policy takes two information gathering actions before taking a downstream action and so on:

$$V_{S_0}^{\mathcal{T}_u} = \zeta + \gamma\zeta + \gamma^2 0 + \gamma^3 V_G^{\mathcal{T}_1}$$

$$\begin{aligned} V_{S_1}^{\mathcal{T}_u} &= \zeta + \gamma\zeta + \gamma^2 0 + \gamma^3 V_{S_0}^{\mathcal{T}_u} \\ &= \zeta + \gamma\zeta + \gamma^2 0 + \gamma^3(\zeta + \gamma\zeta + \gamma^2 0 + \gamma^3 V_G^{\mathcal{T}_1}) \\ &= \zeta + \gamma\zeta + \gamma^3\zeta + \gamma^4\zeta + \gamma^6 V_G^{\mathcal{T}_1} \end{aligned}$$

We get:

$$\begin{aligned} J(\mathcal{T}_u) &= \frac{1}{4}V_G^{\mathcal{T}_u} + \frac{1}{4}V_{S_0}^{\mathcal{T}_u} + \frac{1}{4}V_{S_1}^{\mathcal{T}_u} + \frac{1}{4}V_{S_2}^{\mathcal{T}_u} \\ &= \frac{1}{4}V_G^{\mathcal{T}_1} + \frac{1}{4}(\zeta + \gamma\zeta + \gamma^3 V_G^{\mathcal{T}_1}) + \frac{1}{4}(\zeta + \gamma\zeta + \gamma^3\zeta + \gamma^4\zeta + \gamma^6 V_G^{\mathcal{T}_1}) + \frac{1}{4}V_{S_2}^{\mathcal{T}_1} \\ &= \frac{1}{4}(\frac{\zeta+\gamma}{1-\gamma^2}) + \frac{1}{4}(\frac{\gamma\zeta + \gamma^4 + \zeta - \gamma^2\zeta}{1-\gamma^2}) + \frac{1}{4}(\zeta + \gamma\zeta + \gamma^3\zeta + \gamma^4\zeta + \gamma^6 V_G^{\mathcal{T}_1}) + \frac{1}{4}V_{S_2}^{\mathcal{T}_1} \\ &= \frac{1}{4}(\frac{\zeta+\gamma}{1-\gamma^2}) + \frac{1}{4}(\frac{\gamma\zeta + \gamma^4 + \zeta - \gamma^2\zeta}{1-\gamma^2}) + \frac{1}{4}(\frac{\zeta + \gamma\zeta - \gamma^2\zeta - \gamma^5\zeta + \gamma^6\zeta + \gamma^7}{1-\gamma^2}) + \frac{1}{4}V_{S_2}^{\mathcal{T}_1} \\ &= \frac{1}{4}(\frac{\zeta+\gamma}{1-\gamma^2}) + \frac{1}{4}(\frac{\gamma\zeta + \gamma^4 + \zeta - \gamma^2\zeta}{1-\gamma^2}) + \frac{1}{4}(\frac{\zeta + \gamma\zeta - \gamma^2\zeta - \gamma^5\zeta + \gamma^6\zeta + \gamma^7}{1-\gamma^2}) + \frac{1}{4}(\frac{\zeta+\gamma^3}{1-\gamma^2}) \\ &= \frac{\zeta(4 + 2\gamma - 2\gamma^2 - \gamma^5 + \gamma^6) + \gamma + \gamma^3 + \gamma^4 + \gamma^7}{4(1-\gamma^2)} \end{aligned}$$

The balanced depth-2 decision tree: alternates in every state between taking the two available information gathering actions and then a downstream action. The value of the policy in the goal state is:

$$\begin{aligned} V_G^{\mathcal{T}_2} &= \zeta + \gamma\zeta + \gamma^2 + \gamma^3\zeta + \gamma^4\zeta + \dots \\ &= \sum_{t=0}^{\infty}\gamma^{3t}\zeta + \sum_{t=0}^{\infty}\gamma^{3t+1}\zeta + \sum_{t=0}^{\infty}\gamma^{3t+2} \\ &= \frac{\zeta}{1-\gamma^3} + \frac{\gamma\zeta}{1-\gamma^3} + \frac{\gamma^2}{1-\gamma^3} \end{aligned}$$

Following the same reasoning for other states we find the objective value for the depth-2 decision tree policy to be:

$$\begin{aligned} J(\mathcal{T}_2) &= \frac{1}{4}V_G^{\mathcal{T}_2} + \frac{2}{4}V_{S_2}^{\mathcal{T}_2} + \frac{1}{4}V_{S_1}^{\mathcal{T}_2} \\ &= \frac{1}{4}V_G^{\mathcal{T}_2} + \frac{2}{4}(\zeta + \gamma\zeta + \gamma^2 0 + \gamma^3 V_G^{\mathcal{T}_2}) + \frac{1}{4}(\zeta + \gamma\zeta + \gamma^2 0 + \gamma^3\zeta + \gamma^4\zeta + \gamma^5 0 + \gamma^6 V_G^{\mathcal{T}_2}) \\ &= \frac{\zeta(3+3\gamma) + \gamma^2 + \gamma^5 + \gamma^8}{4(1-\gamma^3)} \end{aligned}$$

Infinite tree: we also consider the infinite tree policy that repeats an information gathering action forever and has objective: $J(\mathcal{T}_{\text{inf}}) = \frac{\zeta}{1-\gamma}$

Stochastic policy: the other non-trivial policy that can be learned by solving a partially observable IBMDP is the stochastic policy that guarantees to reach G after some time: fifty percent chance to do $\rightarrow$ and fifty percent chance to do $\downarrow$. This stochastic policy has objective value:

$$\begin{aligned} V_G^{\text{stoch}} &= \frac{1}{1-\gamma} \\ V_{S_0}^{\text{stoch}} &= 0 + \frac{1}{2}\gamma V_G^{\text{stoch}} + \frac{1}{2}\gamma V_{S_1}^{\text{stoch}} \\ V_{S_2}^{\text{stoch}} &= 0 + \frac{1}{2}\gamma V_G^{\text{stoch}} + \frac{1}{2}\gamma V_{S_1}^{\text{stoch}} = V_{S_0}^{\text{stoch}} \\ V_{S_1}^{\text{stoch}} &= 0 + \frac{1}{2}\gamma V_{S_2}^{\text{stoch}} + \frac{1}{2}\gamma V_G^{\text{stoch}} = \frac{1}{2}\gamma V_{S_0}^{\text{stoch}} + \frac{1}{2}\gamma V_G^{\text{stoch}} \end{aligned}$$

Solving these equations:

$$\begin{aligned} V_{S_1}^{\text{stoch}} &= \frac{1}{2}\gamma V_{S_0}^{\text{stoch}} + \frac{1}{2}\gamma V_G^{\text{stoch}} \\ &= \frac{1}{2}\gamma(\frac{1}{2}\gamma V_G^{\text{stoch}} + \frac{1}{2}\gamma V_{S_1}^{\text{stoch}}) + \frac{1}{2}\gamma V_G^{\text{stoch}} \\ &= \frac{1}{4}\gamma^2 V_G^{\text{stoch}} + \frac{1}{4}\gamma^2 V_{S_1}^{\text{stoch}} + \frac{1}{2}\gamma V_G^{\text{stoch}} \\ V_{S_1}^{\text{stoch}} - \frac{1}{4}\gamma^2 V_{S_1}^{\text{stoch}} &= \frac{1}{4}\gamma^2 V_G^{\text{stoch}} + \frac{1}{2}\gamma V_G^{\text{stoch}} \\ V_{S_1}^{\text{stoch}}(1 - \frac{1}{4}\gamma^2) &= (\frac{1}{4}\gamma^2 + \frac{1}{2}\gamma)V_G^{\text{stoch}} \\ V_{S_1}^{\text{stoch}} &= \frac{\frac{1}{4}\gamma^2 + \frac{1}{2}\gamma}{1 - \frac{1}{4}\gamma^2}V_G^{\text{stoch}} \\ &= \frac{\gamma(\frac{1}{4}\gamma + \frac{1}{2})}{1 - \frac{1}{4}\gamma^2} \cdot \frac{1}{1-\gamma} \\ &= \frac{\gamma(\frac{1}{4}\gamma + \frac{1}{2})}{(1 - \frac{1}{4}\gamma^2)(1-\gamma)} \end{aligned}$$

$$
\begin{aligned}
V_{S_0}^{\text{stoch}} &= \frac{1}{2}\gamma V_G^{\text{stoch}} + \frac{1}{2}\gamma V_{S_1}^{\text{stoch}} \\
&= \frac{1}{2}\gamma \cdot \frac{1}{1-\gamma} + \frac{1}{2}\gamma \cdot \frac{\gamma(\frac{1}{4}\gamma + \frac{1}{2})}{(1-\frac{1}{4}\gamma^2)(1-\gamma)} \\
&= \frac{\frac{1}{2}\gamma}{1-\gamma} + \frac{\frac{1}{2}\gamma^2(\frac{1}{4}\gamma + \frac{1}{2})}{(1-\frac{1}{4}\gamma^2)(1-\gamma)} \\
&= \frac{\frac{1}{2}\gamma(1-\frac{1}{4}\gamma^2) + \frac{1}{2}\gamma^2(\frac{1}{4}\gamma + \frac{1}{2})}{(1-\frac{1}{4}\gamma^2)(1-\gamma)} \\
&= \frac{\frac{1}{2}\gamma - \frac{1}{8}\gamma^3 + \frac{1}{8}\gamma^3 + \frac{1}{4}\gamma^2}{(1-\frac{1}{4}\gamma^2)(1-\gamma)} \\
&= \frac{\frac{1}{2}\gamma + \frac{1}{4}\gamma^2}{(1-\frac{1}{4}\gamma^2)(1-\gamma)} \\
&= \frac{\gamma(\frac{1}{2} + \frac{1}{4}\gamma)}{(1-\frac{1}{4}\gamma^2)(1-\gamma)}
\end{aligned}
$$

$$
\begin{aligned}
J(\mathcal{T}_{\text{stoch}}) &= \frac{1}{4}(V_G^{\text{stoch}} + V_{S_0}^{\text{stoch}} + V_{S_1}^{\text{stoch}} + V_{S_2}^{\text{stoch}}) \\
&= \frac{1}{4}\left(\frac{1}{1-\gamma} + 2\cdot\frac{\gamma(\frac{1}{2}+\frac{1}{4}\gamma)}{(1-\frac{1}{4}\gamma^2)(1-\gamma)} + \frac{\gamma(\frac{1}{4}\gamma+\frac{1}{2})}{(1-\frac{1}{4}\gamma^2)(1-\gamma)}\right) \\
&= \frac{1}{4}\left(\frac{1}{1-\gamma} + \frac{2\gamma(\frac{1}{2}+\frac{1}{4}\gamma) + \gamma(\frac{1}{4}\gamma+\frac{1}{2})}{(1-\frac{1}{4}\gamma^2)(1-\gamma)}\right) \\
&= \frac{1}{4}\left(\frac{1}{1-\gamma} + \frac{\gamma + \frac{1}{2}\gamma^2 + \frac{1}{4}\gamma^2 + \frac{1}{2}\gamma}{(1-\frac{1}{4}\gamma^2)(1-\gamma)}\right) \\
&= \frac{1}{4}\left(\frac{1}{1-\gamma} + \frac{\frac{3}{2}\gamma + \frac{3}{4}\gamma^2}{(1-\frac{1}{4}\gamma^2)(1-\gamma)}\right) \\
&= \frac{1}{4}\left(\frac{1 - \frac{1}{4}\gamma^2 + \frac{3}{2}\gamma + \frac{3}{4}\gamma^2}{(1-\frac{1}{4}\gamma^2)(1-\gamma)}\right) \\
&= \frac{1}{4}\left(\frac{1 + \frac{3}{2}\gamma + \frac{1}{2}\gamma^2}{(1-\frac{1}{4}\gamma^2)(1-\gamma)}\right) \\
&= \frac{1 + \frac{3}{2}\gamma + \frac{1}{2}\gamma^2}{4(1-\frac{1}{4}\gamma^2)(1-\gamma)}
\end{aligned}
$$

E. Training curves

We plot the sub-optimality gaps during training, w.r.t. the RL objective (definition 3.2), between the learned memoryless policy and the optimal deterministic memoryless policy: $|\mathbb{E}\left[V^{\pi^\star}(\boldsymbol{s}_0, \boldsymbol{o}_0)|\boldsymbol{s}_0 \sim T_0\right] - \mathbb{E}\left[V^{\pi}(\boldsymbol{s}_0, \boldsymbol{o}_0)|\boldsymbol{s}_0 \sim T_0\right]|$. Because we know the whole POIBMDP model that we can represent exactly as tables, and because we know for each ζ the RL objective value of the optimal deterministic memoryless policy (figure 2), we can report the *exact* sub-optimality gaps. In figure 12, we plot the sub-optimality gaps–averaged over 100 seeds–of learned policies during training. We do so for 200 different POIBMDPs where we change the reward for information gathering actions: we sample 200 ζ values uniformly in $[-1, 2]$. In figure 12, a different color represents a different POIBMDP.

Recall from figure 2 that for: (i) $\zeta \in [-1, 0]$, the optimal deterministic memoryless policy is a depth-0 tree, (ii) $\zeta \in]0, 1[$, the optimal deterministic memoryless policy is a depth-1 tree, and (iii) $\zeta \in [1, 2]$, the optimal deterministic memoryless policy is a "infinite" tree that contains infinite number of internal nodes. We observe that, despite all sub-optimality gaps converging independently of the ζ values, not all algorithms in all POIBMDPs fully minimize the sub-optimality gap. In particular, all algorithms seem to consistently minimize the gap, i.e. learn the optimal policy or Q-function, only for $\zeta \in [1, 2]$ (all the yellow lines go to 0). However, we are interested in the range $\zeta \in]0, 1[$ where the optimal decision tree policy is non-trivial,

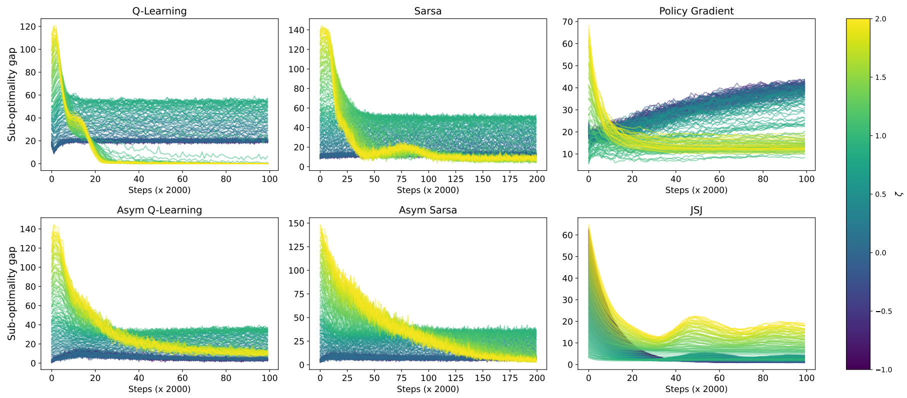


Figure 12. (Asymmetric) reinforcement learning in POIBMDPs. In each subplot, each single line is colored by the value of ζ in the corresponding POIBMDP in which learning occurs. Each single learning curve represent the sub-optimality gap averaged over 100 seeds.

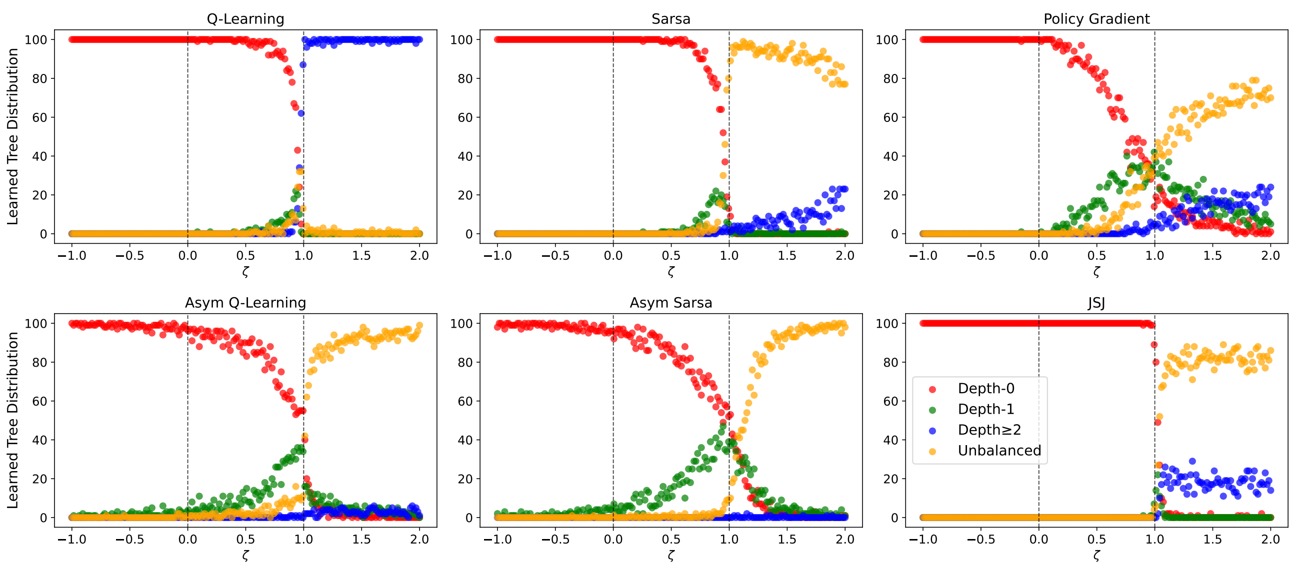


Figure 13. Distributions of final tree policies learned across the 100 seeds. For each ζ value, there are four colored points. Each point represent the share of depth-0 trees (red), depth-1 trees (green), unbalanced depth-2 trees (orange) and depth-2 trees (blue).

i.e. not taking the same action forever. In that range, no baseline consistently minimizes the sub-optimality gap.

F. Tabular RL algorithmic details for POIBMDPs

F.1. Training with the best hyperparameters

G. POIBMDPs for classification tasks

Let us show that, POIBMDPs associated with MDPs encoding supervised learning tasks, are in fact MDPs themselves. Let us define such supervised learning MDPs in the context of a classification task (this definition extends trivially to regression tasks).

Definition G.1 (Classification Markov decision process)**.** Given a set of N examples denoted $\mathcal{E} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ where each datum $\boldsymbol{x}_i \in \mathcal{X}$ is described by a set of p features x_{ij} with $1 \leq j \leq p$, and $y_i \in \mathbb{Z}^m$ is the label associated with $\boldsymbol{x}_i$, a classification Markov decision Process is an MDP $\langle S, A, R, T, T_0 \rangle$ (definition 3.1). The state space is $S = \{\boldsymbol{x}_i\}_{i=1}^N$, the set of training data features. The action space is $A = \mathbb{Z}^m$, the set of unique labels. The reward function is $R : S \times A \to \{0, 1\}$ with $R(\boldsymbol{s} = \boldsymbol{x}_i, a) = 1_{\{a=y_i\}}$. The transition function is $T : S \times A \to \Delta(S)$ with $T(\boldsymbol{s}, a, \boldsymbol{s}') = \frac{1}{N} \quad \forall \boldsymbol{s}, a, \boldsymbol{s}'$. The initial distribution is $T_0(\boldsymbol{s}_0 = \boldsymbol{s}) = \frac{1}{N}$.

Algorithm 4 Asymmetric Q-Learning. We highlight in green the differences with the standard Q-learning (**?**)

Data: A POMDP, learning rates $\alpha_u,\ \alpha_q$, exploration prob. ϵ
Result: $\pi : O \to A$
Initialize $U(\boldsymbol{x}, a) = 0$ for all $x \in X, a \in A$
Initialize $Q(\boldsymbol{o}, a) = 0$ for all $o \in O, a \in A$
for *each episode* **do**
  Initialize state $x_0 \sim T_0$
  Initialize observation $\boldsymbol{o}_0 \sim \Omega(\boldsymbol{x}_0)$

  for *each step* t **do**
    Choose action a_t using ϵ-greedy: $a_t = \operatorname{argmax}_a Q(\boldsymbol{o}_t, a)$ with prob. $1 - \epsilon$
    Take action a_t, observe $r_t = R(\boldsymbol{x}_t, a_t)$, $x_{t+1} \sim T(x_t, a_t)$, and $\boldsymbol{o}_{t+1} \sim \Omega(\boldsymbol{x}_{t+1})$
    $y \leftarrow r + \gamma U(\boldsymbol{x}_{t+1}, \operatorname{argmax}_{a'} Q(\boldsymbol{o}_{t+1}, a'))$
    $U(\boldsymbol{x}_t, a_t) \leftarrow (1 - \alpha_u)U(\boldsymbol{x}_t, a_t) + \alpha_u y$
    $Q(\boldsymbol{o}_t, a_t) \leftarrow (1 - \alpha_q)Q(\boldsymbol{o}_t, a_t) + \alpha_q y$
    $x_t \leftarrow \boldsymbol{x}_{t+1}$
    $\boldsymbol{o}_t \leftarrow \boldsymbol{o}_{t+1}$

  end
end
$\pi(o) = \operatorname{argmax}_a Q(\boldsymbol{o}, a)$

Algorithm 5 Asymmetric Sarsa

Data: POMDP $\mathcal{M}_{po} = \langle X, O, A, R, T, T_0, \Omega \rangle$, learning rates α_u, α_q, exploration rate ϵ
Result: $\pi : O \to A$
Initialize $U(x, a) = 0$ for all $x \in X, a \in A$
Initialize $Q(o, a) = 0$ for all $o \in O, a \in A$

for *each episode* **do**
  Initialize state $x_0 \sim T_0$
  Initialize observation $\boldsymbol{o}_0 \sim \Omega(x_0)$
  Choose action a_0 using ϵ-greedy: $a_0 = \operatorname{argmax}_a Q(\boldsymbol{o}_0, a)$ with prob. $1 - \epsilon$

  for *each step* t **do**
    Take action a_t, observe $r_t = R(x_t, a_t)$, $x_{t+1} \sim T(x_t, a_t)$, and $\boldsymbol{o}_{t+1} \sim \Omega(x_{t+1})$
    Choose action a_{t+1} using ϵ-greedy: $a_{t+1} = \operatorname{argmax}_a Q(\boldsymbol{o}_{t+1}, a)$ with prob. $1 - \epsilon$
    $y \leftarrow r + \gamma U(x_{t+1}, a_{t+1})$ `// TD target using actual next action`
    $U(x_t, a_t) \leftarrow (1 - \alpha_u)U(x_t, a_t) + \alpha_u y$
    $Q(\boldsymbol{o}_t, a_t) \leftarrow (1 - \alpha_q)Q(\boldsymbol{o}_t, a_t) + \alpha_q y$
    $x_t \leftarrow x_{t+1}$
    $\boldsymbol{o}_t \leftarrow \boldsymbol{o}_{t+1}$
    $a_t \leftarrow a_{t+1}$

  end
end
$\pi(bo) = \operatorname{argmax}_a Q(\boldsymbol{o}, a)$ `// Extract greedy policy`

Algorithm 6 Asymmetric policy gradient algorithm. Uses Monte Carlo estimates of the average reward value functions to perform policy improvements.

Data: POMDP $\mathcal{M}_{po} = \langle X, O, A, R, T, T_0, \Omega \rangle$, learning rate α, policy parameters θ, number of trajectories N
Result: Stochastic partially observable policy $\pi_\theta : O \rightarrow \Delta(A)$
Initialize policy parameters θ
Initialize $Q(\boldsymbol{o}, a) = 0$ for all observations o and actions a
for *each episode* **do**
  for $i = 1$ *to* N **do**
    Generate trajectory $\tau_i = (\boldsymbol{s}_0, a_0, r_0, \boldsymbol{s}_1, a_1, r_1, \ldots, \boldsymbol{s}_T)$ following π_θ
    for *each timestep* t *in trajectory* τ_i **do**
      $G_t \leftarrow \sum_{k=t}^{T} \gamma^{k-t} r_k$ `// Compute return`
      Store $(\boldsymbol{o}_t, a_t, G_t)$ for later averaging
    end
  end
  for *each unique observation-action pair* (o, a) **do**
    $Q(o, a) \leftarrow \frac{1}{|\{(\boldsymbol{o},a)\}|} \sum_{(\boldsymbol{o},a,G)} G$ `// Monte Carlo estimate`
  end
  for *each observation* o **do**
    for *each action* a **do**
      $\pi_1(a|\boldsymbol{o}) \leftarrow 1.0$ if $a = \text{argmax}_{a'} Q(\boldsymbol{o}, a')$, 0.0 otherwise `// Deterministic policy from Q-values`
      $\pi(a|\boldsymbol{o}) \leftarrow (1-\alpha)\pi(a|\boldsymbol{o}) + \alpha\pi_1(a|\boldsymbol{o}o)$ `// Policy improvement step`
    end
  end
  Reset $Q(\boldsymbol{o}, a) = 0$ for all observations $\boldsymbol{o}$ and actions a `// Reset for next episode`
end

Table 6. Summary of RL baselines Hyperparameters

algorithm	Problem	Hyperparameters comb.
Policy Gradient	PO/IB/MDP	420
JSJ	POIBMDP	15
Q-learning	PO/IB/MDP	192
Asym Q-learning	POIBMDP	768
Sarsa	PO/IB/MDP	192
Asym Sarsa	POIBMDP	768

Table 7. PG Hyperparameter Space (140 combinations)

Hyperparameter	Values	Description
Learning Rate (lr)	0.001, 0.005, 0.01, 0.05, 0.1	Policy gradient step size
Entropy Regularization (tau)	-1.0, -0.1, -0.01, 0.0, 0.01, 0.1, 1.0	Entropy regularization coefficient
Temperature (eps)	0.01, 0.1, 1.0, 10	Softmax temperature
Episodes per Update (n_steps)	20, 200, 2000	Number of episodes per policy update

Table 8. PG-IBMDP Hyperparameter Space (140 combinations)

Hyperparameter	Values	Description
Learning Rate (lr)	0.001, 0.005, 0.01, 0.05, 0.1	Policy gradient step size
Entropy Regularization (tau)	-1.0, -0.1, -0.01, 0.0, 0.01, 0.1, 1.0	Entropy regularization coefficient
Temperature (eps)	0.01, 0.1, 1.0, 10	Softmax temperature
Episodes per Update (n_steps)	10, 100, 1000	Number of episodes per policy update

Table 9. QL Hyperparameter Space (192 combinations)

Hyperparameter	**Values**	**Description**
Epsilon Schedules	(0.3, 1), (0.3, 0.99), (1, 1)	Initial exploration and decrease rate
Epsilon Schedules	(0.1, 1), (0.1, 0.99), (0.3, 0.99)	Initial exploration and decrease rate
Lambda	0.0, 0.3, 0.6, 0.9	Eligibility trace decay
Learning Rate (lr_o)	0.001, 0.005, 0.01, 0.1	Observation Q-learning rate
Optimistic	True, False	Optimistic initialization

Table 10. QL-Asym Hyperparameter Space (768 combinations)

Hyperparameter	**Values**	**Description**
Epsilon Schedules	(0.3, 1), (0.3, 0.99), (1, 1)	Initial exploration and decrease rate
Epsilon Schedules	(0.1, 1), (0.1, 0.99), (0.3, 0.99)	Initial exploration and decrease rate
Lambda	0.0, 0.3, 0.6, 0.9	Eligibility trace decay
Learning Rate (lr_o)	0.001, 0.005, 0.01, 0.1	Observation Q-learning rate
Learning Rate (lr_v)	0.001, 0.005, 0.01, 0.1	State-action Q-learning rate
Optimistic	True, False	Optimistic initialization

Table 11. QL-IBMDP Hyperparameter Space (192 combinations)

Hyperparameter	**Values**	**Description**
Epsilon Schedules	(0.3, 1), (0.3, 0.99), (1, 1)	Initial exploration and decrease rate
Epsilon Schedules	(0.1, 1), (0.1, 0.99), (0.3, 0.99)	Initial exploration and decrease rate
Lambda	0.0, 0.3, 0.6, 0.9	Eligibility trace decay
Learning Rate (lr_v)	0.001, 0.005, 0.01, 00.1	State-action Q-learning rate
Optimistic	True, False	Optimistic initialization

Table 12. SARSA Hyperparameter Space (192 combinations)

Hyperparameter	**Values**	**Description**
Epsilon Schedules	(0.3, 1), (0.3, 0.99), (1, 1)	Initial exploration and decrease rate
Epsilon Schedules	(0.1, 1), (0.1, 0.99), (0.3, 0.99)	Initial exploration and decrease rate
Lambda	0.0, 0.3, 0.6, 0.9	Eligibility trace decay
Learning Rate (lr_o)	0.001, 0.005, 0.01, 0.1	Observation SARSA learning rate
Optimistic	True, False	Optimistic initialization

Table 13. SARSA-Asym Hyperparameter Space (768 combinations)

Hyperparameter	**Values**	**Description**
Epsilon Schedules	(0.3, 1), (0.3, 0.99), (1, 1)	Initial exploration and decrease rate
Epsilon Schedules	(0.1, 1), (0.1, 0.99), (0.3, 0.99)	Initial exploration and decrease rate
Lambda	0.0, 0.3, 0.6, 0.9	Eligibility trace decay
Learning Rate (lr_o)	0.001, 0.005, 0.01, 0.1	Observation SARSA learning rate
Learning Rate (lr_v)	0.001, 0.005, 0.01, 0.1	State-action SARSA learning rate
Optimistic	True, False	Optimistic initialization

Table 14. SARSA-IBMDP Hyperparameter Space (192 combinations)

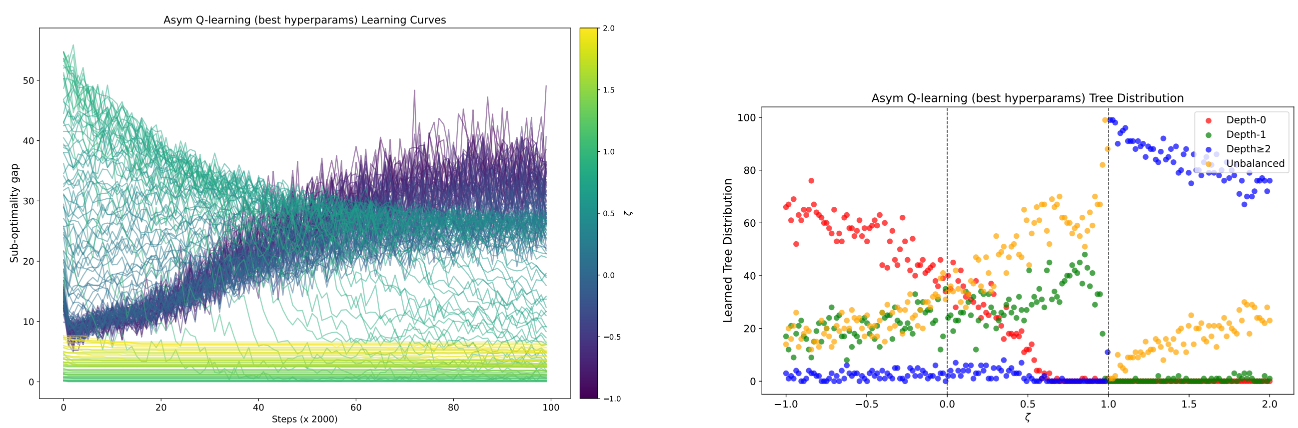
Hyperparameter	**Values**	**Description**
Epsilon Schedules	(0.3, 1), (0.3, 0.99), (1, 1)	Initial exploration and decrease rate
Epsilon Schedules	(0.1, 1), (0.1, 0.99), (0.3, 0.99)	Initial exploration and decrease rate
Lambda	0.0, 0.3, 0.6, 0.9	Eligibility trace decay
Learning Rate (lr_v)	0.001, 0.005, 0.01, 0.1	State-action SARSA learning rate
Optimistic	True, False	Optimistic initialization

Table 15. Asymmetric sarsa hyperparameters (768 combinations each run 10 times)

Hyperparameter	Values	Description
Epsilon Schedules	(0.3, 1), (0.3, 0.99), (1, 1)	Initial exploration and decrease rate
Epsilon Schedules	(0.1, 1), (0.1, 0.99), (0.3, 0.99)	Initial exploration and decrease rate
Lambda	0.0, 0.3, 0.6, 0.9	Eligibility trace decay
Learning Rate U	0.001, 0.005, 0.01, 0.1	learning rate for the Q-function
Learning Rate Q	0.001, 0.005, 0.01, 0.1	learning rate for the partial observation dependent Q-function
Optimistic	True, False	Optimistic initialization

Hyperparameter	**Asym Q-learning (10/10)**	**Asym Sarsa (10/10)**	**PG (4/10)**
epsilon_start	1.0	1.0	-
epsilon_decay	0.99	0.99	-
batch_size	1	1	-
lambda_	0.0	0.0	-
lr_o	0.01	0.1	-
lr_v	0.1	0.005	-
optimistic	False	False	-
lr	-	-	0.05
tau	-	-	0.1
eps	-	-	0.1
n_steps	-	-	2000

Table 16. Best hyperparameters for each algorithm on the POIBMDP problem



(a) Learning curves for asymmetric Q-learning with good hyperparameters.

(b) Trees distributions for asymmetric Q-learning with good hyperparameters

Figure 14. Analysis of the top-performing asymmetric Q-learning instantiation. (left) Learning curves, and (right) tree distributions across different POIBMDP configurations.

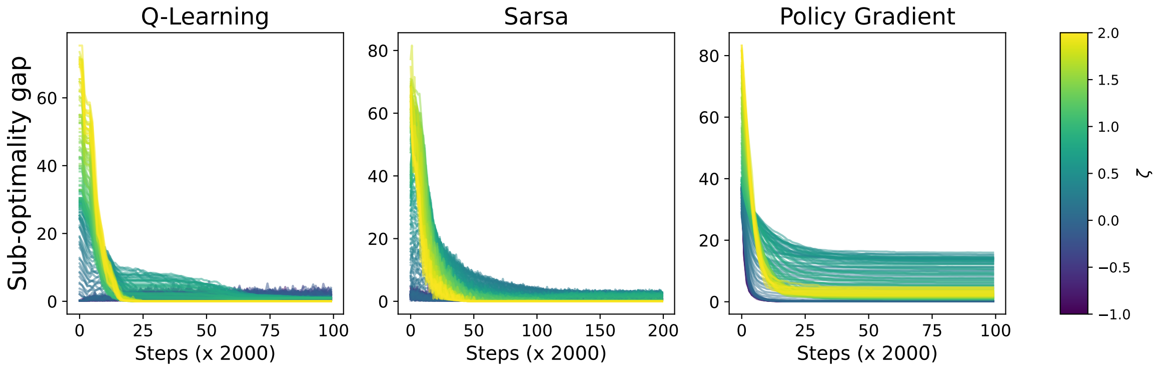


Figure 15. We reproduce the same plot as in figure 12 for classification POIBMDPs. Each individual curve is the sub-optimality gap of the learned policy during training averaged over 100 runs for a single ζ value.

One can be convinced that policies that maximize the RL objective (definition 3.2) in classification MDPs are classifiers that maximize the prediction accuracy because $\sum_{i=1}^{N} 1_{\pi(\boldsymbol{x}_i)=y_i} = \sum_{i=1}^{N} R(\boldsymbol{x}_i, \pi(\boldsymbol{x}_i))$. We defer the formal proof in the next part of the manuscript in which we extensively study supervised learning problems.

Now let us show that associated POIBMDPs are in fact MDPs. We show this by construction.

Definition G.2 (Classification POIBMDP)**.** Given a classification MDP $\langle \{\boldsymbol{x}_i\}_{i=1}^{N}, \mathbb{Z}^m, R, T, T_0 \rangle$ (definition G.1), and an associated POIBMDP $\langle S, O, A, A_{info}, R, \zeta, T_{info}, T, T_0 \rangle$ (definition 4.2), a classification POIBMDP is an MDP (definition 3.1):

$$\langle \overbrace{O}^{\text{State space}}, \overbrace{\mathbb{Z}^m, A_{info},}^{\text{Action space}} \overbrace{R, \zeta}^{\text{Reward function}}, \overbrace{\mathcal{P}, \mathcal{P}_0}^{\text{Transition functions}} \rangle$$

O is the set of possible observations in $[L_1, U_1] \times \cdots \times [L_p, U_p] \times [L_1, U_1] \times \cdots \times [L_p, U_p]$ where L_j is the minimum value of feature j over all data $\boldsymbol{x}_i$ and U_j the maximum. $\mathbb{Z}^m \cup A_{info}$ is action space: actions can be label assignments in $\mathbb{Z}^m$ or bounds refinements in A_{info}. The reward for assigning label $a \in \mathbb{Z}^m$ when observing some observation $\boldsymbol{o} = (L'_1, U'_1, \ldots, L'_p, U'_p)$ is the proportion of training data satisfying the bounds and having label a: $R(o, a) = \frac{|\{\boldsymbol{x}_i : L'_j \leq x_{ij} \leq U'_j \forall i,j\} \cap \{\boldsymbol{x}_i : y_i = a \forall i\}|}{|\{\boldsymbol{x}_i : L'_j \leq x_{ij} \leq U'_j \forall i,j\}|}$. The reward for taking an information gathering action that refines bounds is ζ. The transition function is $\mathcal{P} : O \times (\mathbb{Z}^m \cup A_{info}) \to \Delta(O)$. When $a \in \mathbb{Z}^m$, $\mathcal{P}(\boldsymbol{o}, a, (L_1, U_1, \ldots, L_p, U_p)) = 1$ (reset to full bounds). When $a = (k, v) \in A_{info}$, from $\boldsymbol{o} = (L'_1, U'_1, \ldots, L'_p, U'_p)$, the MDP will transit to $\boldsymbol{o}_{left} = (L'_1, U'_1, \ldots, L_k, v, \ldots, L'_p, U'_p)$ (resp. $\boldsymbol{o}_{right} = (L'_1, U'_1, \ldots, U'_k, v, \ldots, L'_p, U'_p)$) with probability $\frac{|\{\boldsymbol{x}_i : L'_j \leq x_{ij} \leq U'_j \forall j \wedge x_{ik} \leq v\}|}{|\{\boldsymbol{x}_i : L'_j \leq x_{ij} \leq U'_j \forall j\}|}$ (resp. $\frac{|\{\boldsymbol{x}_i : L'_j \leq x_{ij} \leq U'_j \forall j \wedge x_{ik} > v\}|}{|\{\boldsymbol{x}_i : L'_j \leq x_{ij} \leq U'_j \forall j\}|}$).

Those classification POIBMDPs are essentially MDPs with stochastic transitions. It means that deterministic memoryless policies $O :\to A \cup A_{info}$ are in fact Markovian policy for those classification POIBMDPs. More importantly, it means that, for a given γ and ζ, if we were to know the whole POIBMDP model, we could use planning, to compute *optimal* decision tree policies.