Limits of reinforcement learning for decision trees in Markov decision processes

Anonymous Authors[1]

Abstract

For applications like medicine where model interpretability is critical, decision trees might be preferred over neural networks because humans can read their predictions from the root to the leaves. Most of the litterature on interpretable reinforcement learning that focuses on imitating (or distilling) neural networks. In contrast, we study reinforcement learning algorithms returning decision trees that optimize 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 sequential decision making problems.

1. Introduction

Interpretable machine learning provides either local or global interpretations (**?**). Global methods, like decision tree induction (**?**), return a model whose outputs can be interpreted without having to run an additional algorithm. By contrast, local methods require to run an additional algorithm, e.g. linear regression (**?**), but are agnostic to the model class.

Local interpretable model-agnostic explanations (LIME) (**?**) is a popular local interpretability method. Given a model, LIME works by perturbing the input and learning a simple interpretable model locally to explain that particular prediction (cf. figure **??**). For each individual prediction, LIME provides interpretations by identifying which features were most important for that specific decision.

Local interpretability can also be called explainability. Global interpretability methods constrain the model class so that the computations are transparent or verifiable by construction. On the other hand, explainability methods–or local interpretability methods–keep black-box models and generates post hoc explanations of their decisions. In additions to linear models, explanations can take various forms: visual explanations with saliency maps (**?**), attribution such as SHAP(**?**), attention-based highlighting (**?**).

While useful for insight, these explanations are often subjective and might not be faithful to the underlying computations (**?**). For safety-critical settings, this motivates our focus on models that are interpretable by design. Those interpretable by design models can be obtained by global interpretability methods that we present next.

Global approaches are either direct or indirect (**?**). Direct algorithms, such as decision tree induction (**?**), *directly* learn an interpretable model optimizing some objective (cf. figure **??**). One key challenge motivating this work is that decision tree induction is well-developed for supervised learning but not for reinforcement learning. To directly learn interpretable models for sequential decision making, one must design new algorithms which will be the core of this paper.

Most existing research has focused on developing indirect methods. Indirect methods for interpretable sequential decision making—sometimes called *post hoc* methods—begin by learning a non-interpretable model (e.g., reinforcement learning of a neural network model), and then use supervised learning to fit an interpretable model that emulates the black-box. Indirect methods rely on behavior cloning or imitation learning (**??**) to emulate the black-box models. Most work on interpretable sequential decision focuses on the indirect approach(**??**).

Verifiable Reinforcement learning via Policy Extraction, or VIPER (**?**), is a strong indirect method to learn decision tree models for sequential decision making. VIPER first trains a neural network model with reinforcement learning and then fit a decision tree to minimize the disagreement between the neural network and the tree outputs given the same inputs. They show that decision tree models, in addition to being transparent, are also fast to verify in the formal sense of the

[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.

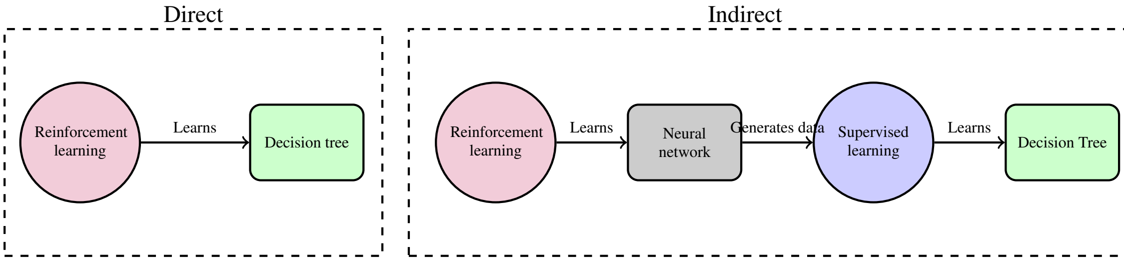


Figure 1. Comparison of direct and indirect approaches for learning interpretable models in sequential decision making.

term (**?**). Programmatic models are an interpretable class that contains decision trees. Programmatically Interpretable Reinforcement Learning (PIRL) (**?**) synthesizes programs in a domain-specific language, also by imitating a neural network model.

However, unlike direct methods that return interpretable models optimizing the desired objective, indirect methods learn an interpretable model to match the behavior of a black-box that itself optimizes the objective of interest. Due to the restricted policy class, the best decision tree model might employ a completely different strategy than the best black-box neural network model to solve the task. Furthermore, the decision tree model that best fits the best neural network model might be sub-optimal compared to the decision tree model that best solves the task of interest. Hence, there is no guarantee that optimizing this surrogate objective of best emulating a black-box yields the best interpretability–performance trade-offs. Figure 1 illustrates the key difference between these two approaches.

Beyond direct and indirect learning, a complementary strategy is to train experts that are inherently easier to imitate and understand. This is achieved by adding interpretability-oriented regularization during training. In the context of supervised learning tasks, the authors of (**?**) regularize the neural network model during training such that indirect approaches will be biased towards more interpretable trees.

In addition to finding models which computations can be read by humans or that can be formally verified, interpretable machine learning has also been used to detect reward misalignment in sequential decision making: by exposing the decision process of a model, one can identify goal mis-specification or unintended shortcuts. Such shortcuts can be, for example, following the shadow of someone instead of actually following someone because for the model they lead to the same reactions. The learning of interpretable models for misalignment detection has been heavily studied by Quentin Delfosse contemporarily to this manuscript (**????**).

1.1. Contributions

We show that direct reinforcement learning of decision tree policies for MDPs (cf. definition 3.2), i.e. learning a decision tree that optimizes the RL objective (cf. definition 3.3) is often very difficult. In particular, we provide some insights as to why it is so difficult and show that indirect imitation of a neural network policy (cf. section 4.3), despite optimizing the imitation learning objective (cf. definition 4.4) rather than the RL one, often yields very good tree policies in practice.

This first part of the manuscript is organized as follows. In this chapter, we present Nicholay Topin and colleagues' framework for direct reinforcement learning of decision tree policies (**?**). In chapter **??**, we reproduce experiments from (**?**) where we compare direct deep reinforcement learning (cf. section **??**) of decision tree policies to indirect imitation of neural network policies with decision trees for the simple CartPole MDP (**?**). In chapter **??**, we show that this direct approach is equivalent to learning a deterministic memoryless policy for partially observable MDP (POMDP)(**??**)–which is a hard problem (**?**)–and show that this might be the main reason for failures. In chapter **??**, we further support this claim by constructing special instances of such POMDPs where the observations contain all the information about hidden states, and show that in those cases, direct reinforcement learning of decision trees works well.

2. Learning decision tree policies for MDPs

In the introductory example (cf. section 4.4), we have shown that imitation learning algorithms that optimize the imitation learning objective (cf. definition 4.4) rather than the RL objective, are, unsurprisingly, prone to sub-optimality w.r.t. the latter objective. This motivates the study and development of *direct* decision tree policy learning algorithms. There already exists such algorithms that return decision tree policies optimizing the RL objective for a given MDP. Those algorithms either learn parametric trees or non-parametric trees.

Parametric trees are not "grown" from the root by iteratively adding internal or leaf nodes (cf. figure 2), but are rather "optimized": the depth, internal nodes arrangement, and state features to consider in each node are fixed *a priori* and only the thresholds of each node are learned. This is similar to doing gradient descent on neural network weights. As the reader might have guessed, those parametric trees are advantageous in that they can be learned with the policy gradient (**?**). In (**?**), (**?**) and (**?**), the authors use PPO (cf. algorithm **??**) to learn such differentiable decision trees that optimize directly the RL objective. In particular, (**?**) explicitly studies the gap in RL objective values between their direct optimization and the imitation learning algorithm VIPER (cf. algorithm **??**). While those methods return decision tree policies that optimize the RL objective well, in general a user cannot know *a priori* what a "good" tree policy structure should be for a particular MDP. It could be that the specified structure is too deep and pruning will be required after training or it could be that the tree structure is not expressive enough to encode a good policy, i.e. parametric trees cannot trade off interpretability and performances during training. Furthermore, the authors from (**?**) show that extra stabilizing tricks, such as adaptive batch sizes, are required during training to outperform indirect imitation in terms of RL objective.

Non-parametric trees are the standard model in supervised learning. Greedy algorithms (**???**) are fast and return decision tree classifiers (or regressors) that offer good trade-offs between interpretability (depth, or number of nodes) and the supervised learning objective (cf. definition **??**). On the other hand, to the best of our knowledge, there exists only one work studying non-parametric trees to optimize a trade-off between interpretability and the RL objective: Topin et. al. (**?**).

Given an MDP for which one wants to learn a decision tree policy, Topin et. al. introduced iterative bounding Markov decision processes (IBMDPs). From now on we refer to the MDP for which we want a decision tree policy as the "base" MDP. IBMDPs are an augmented version of this base MDP with more state features, more actions, additional reward signals, and additional transitions. Authors showed that certain policies in IBMDPs are equivalent to non-parametric decision tree policies that trade off between interpretability and the RL objective in the base MDP. Hence, the great promise of Topin et. al.'s work is that doing e.g. RL to learn such IBMDP policies is a way to directly optimize a trade-off between interpretability and the RL objective.

There also exists more specialized approaches that can return decision tree policies only for very specific problem classes. In (**?**), the authors prove that for maze-like MDPs, there always exists an optimal decision tree policy w.r.t. the RL objective and provide an algorithm to find it. Finally, in (**?**), the authors study decision tree policies for planning in MDPs (cf. algorithm **??**), i.e. when the transitions and rewards are known.

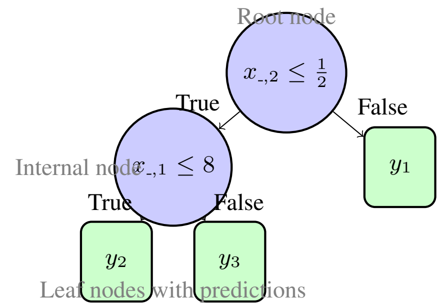


Figure 2. A generic depth 2 decision tree with 2 nodes and 3 leaves. The root node applies the test $1_{\{x_{\cdot,1} \leq \frac{1}{2}\}}$ to check if the first features of data is below $\frac{1}{2}$. Edges represent the outcomes of the tests in each internal nodes(True/False), and leaf nodes contain predictions $y_l \in \mathcal{Y}$. For any input $\boldsymbol{x}_i$, the tree defines a unique path from root to leaf.

3. Technical preliminaries

3.1. What are decision trees?

As the reader might have already guessed, we will put great emphasis on decision tree models as a means to study interpretability. While other interpretable models might have other properties than the ones we will highlight through this thesis, 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 models that we introduce next and that we illustrate in figure 2.

Definition 3.1 (Decision tree)**.** A decision tree is a rooted tree $T = (\mathcal{N}, E)$. Each internal node $\nu \in \mathcal{N}$ is associated with a test that maps input features $x_{ij} \in \mathcal{X}$ to a Boolean. Each edge $e \in E$ from an internal node corresponds to an outcome of the associated test function. Each leaf node $l \in \mathcal{N}$ is associated with a prediction $y_l \in \mathcal{Y}$, where $\mathcal{Y}$ is the output space. For any input $x \in \mathcal{X}$, the tree defines a unique path from root to leaf, determining the prediction $T(x) = y_l$ where l is the reached leaf. The depth of a tree is the maximum path length from root to any leaf.

3.2. Markov decision processes and problems

Markov decision processes (MDPs) 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 decreases 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. Following Martin L. Puterman's book on MDPs(**?**), we formally define:

Definition 3.2 (Markov decision process)**.** An MDP is a tuple $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$ where:

- 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. For example, in cancer treatment, the best outcome is to eliminate the patient's tumor as quickly as possible. We can formally define this objective, that we call the reinforcement learning objective, as follows:

Definition 3.3 (Reinforcement learning objective)**.** Given an MDP $\mathcal{M} \equiv \langle S, A, R, T, T_0 \rangle$ (cf. definition 3.2), 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:

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

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

Algorithms presented in this manuscript aim to find an optimal policy $\pi^\star \in \underset{\pi}{\operatorname{argmax}} J(\pi)$ that maximizes the above reinforcement learning (RL) objective.

3.3. Example: a grid world MDP

In figure 3, we present a very simple MDP (cf. definition 3.2). This MDP is essentially a grid where the starting state is chosen at random and the goal is to reach the bottom-right cell as fast as possible in order to maximize the RL objective (cf. definition 3.3). The state space is discrete with state labels representing 2D-coordinates. The actions are to move up, left, right, or down. The bottom-right cell gives reward 1 and is an absorbing state, i.e., once in the state, the MDP stays in this state forever. Other states give reward 0 and are not absorbing. The optimal actions that get to the goal as fast as possible in every state (cell) are presented in green in figure 3. Next we present the tools to find solutions to MDPs and compute such optimal policies.

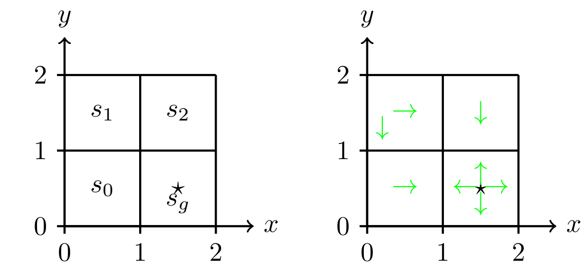


Figure 3. A grid world MDP (left) and optimal actions w.r.t. the objective (cf. definition 3.3) (right).

4. Exact solutions for Markov decision problems

We begin with the planning setting in which the MDP transitions and rewards (cf. definition 3.2) are known. Leveraging the Markov property, one can use dynamic programming (**?**) to compute optimal policies with respect to the RL objective (cf. definition 3.3). Algorithms based on dynamic programming use the notion of *value* of states and actions:

Definition 4.1 (Value of a state)**.** In an MDP $\mathcal{M}$ (cf. definition 3.2), 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:

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

Applying the Markov property gives a recursive definition of the value of s under policy π:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \mathbb{E}\left[V^\pi(s') | s' \sim T(s, \pi(s))\right]$$

Definition 4.2 (Optimal value of a state)**.** 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 (cf. definition 3.3)).

$$V^\star(s) = V^{\pi^\star}(s)$$

Definition 4.3 (Optimal value of a state–action pair)**.** 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.

$$Q^\star(s, a) = R(s, a) + \gamma \mathbb{E}\left[V^\star(s') | s' \sim T(s, a)\right]$$

More realistically, neither the transition function T nor the reward function R of the MDP are known, e.g. the doctor cannot **know** how the tumor and the patient's health will change after a dose of chemotherapy, but can only **observe** the change. This distinction in available information parallels the distinction between dynamic programming and reinforcement learning, described next.

4.1. Reinforcement learning of approximate solutions to MDPs

When the MDP transition function and reward function are unknown, one can use reinforcement learning algorithms–also known as agents–to learn values or policies maximizing the RL objective. Reinforcement learning algorithms popularized by Richard Sutton (**?**) don't **compute** an optimal policy but rather **learn** an approximate one based on sequences of transitions $(s_t, a_t, r_t, s_{t+1})_t$. RL algorithms usually fall into two categories: value-based (**?**) and policy search (**?**). Examples of these approaches are shown in algorithms 1, **??** and **??**. Q-learning and Sarsa compute an approximation of $Q^\star$ (cf. definition 4.3) using temporal difference learning (**?**). Q-learning is *off-policy*: it collects new transitions with a random policy, e.g. epsilon-greedy. Sarsa is *on-policy*: it collects new transitions greedily w.r.t. the current Q-values estimates. Policy gradient algorithms (**?**) leverage the policy gradient theorem to approximate $\pi^\star$.

Q-learning, Sarsa, and policy gradients algorithms are known to converge to the optimal value or (locally) optimal policy under some conditions. There are many other ways to learn policies such as simple random search (**?**) or model-based reinforcement learning that estimates MDP transitions and rewards before applying e.g. value iteration (**?**). Those RL algorithms–also known as tabular RL because they represent policies as tables with $|S| \times |A|$ entries–are limited to small state spaces. To scale to large state spaces, it is common to use a neural network to represent policies or values (**?**). In the next section, we present deep reinforcement learning algorithms designed specifically for neural networks.

4.2. Deep reinforcement learning

Reinforcement learning has also been successfully combined with function approximations to solve MDPs with large discrete state spaces or continuous state spaces ($S \subset \mathbb{R}^p$ in definition 3.2). In the rest of this manuscript, unless stated otherwise, we write $\boldsymbol{s}$ a state vector in a continuous state space[1].

Deep Q-Networks (DQN) (**?**), described in algorithm **??** achieved super-human performance on a set of Atari games. Authors successfully extended the Q-learning (cf. algorithm 1) to the function approximation setting by introducing target networks to mitigate distributional shift in the temporal difference error and replay buffer to increase sample efficiency.

Algorithm 1 Q-Learning (**?**)

Data: MDP $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$, learning rate α, exploration rate ϵ
Result: Policy π
Initialize $Q(s, a) = 0$ for all $s \in S, a \in A$
Initialize state $s_0 \sim T_0$
for *each step* t **do**
  Choose action a_t using e.g. ϵ-greedy policy: $a_t = \text{argmax}_a Q(s_t, a)$ with prob. $1 - \epsilon$
  Take action a_t, observe $r_t = R(s_t, a_t)$ and $s_{t+1} \sim T(s_t, a_t)$
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$
  $s_t \leftarrow s_{t+1}$
end
$\pi(s) = \text{argmax}_a Q(s, a)$ `// Extract greedy policy`

Proximal Policy Optimization (PPO) (**?**), described in algorithm **??**, is an actor-critic algorithm (**?**) optimizing a neural network policy. In actor-critic algorithms, cumulative discounted rewards starting from a particular state, also known as *the returns*, are also estimated with a neural network. PPO is known to work well in a variety of domains including robot control in simulation among others.

4.3. Imitation learning: a baseline (indirect) interpretable reinforcement learning method

Unlike PPO or DQN for neural networks, there exists no algorithm that trains decision tree policies to optimize the RL objective (cf. definition 3.3). In fact, we will show in the first part of the manuscript that training decision trees that optimize the RL objective is very difficult.

Hence, many interpretable reinforcement learning approaches first train a neural network policy–also called an expert policy–to optimize the RL objective (cf. definition 3.3) using e.g. PPO, and then fit a student policy such as a decision tree using CART (cf. algorithm **??**) to optimize the supervised learning objective (cf. definition **??**) with the neural policy actions as targets. This approach is known as imitation learning and is essentially training a student policy to optimize the objective:

Definition 4.4 (Imitation learning objective)**.** Given an MDP $\mathcal{M}$ (cf. definition 3.2), an expert policy π^* and a policy class Π, e.g. decision trees of depth at most 3, the imitation learning objective is to find a student policy $\hat{\pi} \in \Pi$ that minimizes the expected action disagreement with the

[1] Note that discrete states can be one-hot encoded as state vectors in $\{0, 1\}^{|S|}$.

expert:

$$IL(\pi) = \mathbb{E}_{\boldsymbol{s}\sim\rho(\boldsymbol{s})}\left[\mathcal{L}(\pi(\boldsymbol{s}), \pi^*(\boldsymbol{s}))\right] \tag{1}$$

where $\rho(\boldsymbol{s})$ is the state distribution in $\mathcal{M}$ induced by the student policy π and $\mathcal{L}$ is a loss function measuring the disagreement between the student policy's action $\pi(s)$ and the expert's action $\pi^*(s)$.

There are two main imitation learning methods used for interpretable reinforcement learning. Dagger (cf. algorithm **??**) is a straightforward way to fit a decision tree policy to optimize the imitation learning objective (cf. definition 4.4). VIPER (cf. algorithm **??**) was designed specifically for interpretable reinforcement learning. VIPER reweights the transitions collected by the neural network expert by a function of the state-action value (cf. definition 4). The authors of VIPER showed that decision tree policies fitted with VIPER tend to have the same RL objective value as Dagger trees while being more interpretable (shallower or with fewer nodes) and sometimes outperform Dagger trees. Dagger and VIPER are two strong baselines for decision tree learning in MDPs, but they optimize a surrogate objective only, even though in practice the resulting decision tree policies often achieve high RL objective value. We use these two algorithms extensively throughout the manuscript. Next we show how to learn a decision tree policy for the example MDP (cf. figure 3).

4.4. Your first decision tree policy

Now the reader should know how to train decision tree classifiers or regressors for supervised learning using CART (cf. section **??**). The reader should also know what an MDP is and how to compute or learn policies that optimize the RL objective (cf. definition 3.3) with (deep) reinforcement learning (cf. section **??**). Finally, the reader should now know how to obtain a decision tree policy for an MDP through imitation learning (cf. definition 4.4) by first using RL to get an expert policy and then fitting a decision tree to optimize the supervised learning objective, using the expert actions as labels.

In this section we present the first decision tree policies of this manuscript obtained using Dagger or VIPER after learning an expert Q-function for the grid world MDP from figure 3 using Q-learning (cf. algorithm 1). Recall the optimal policies for the grid world, taking the green actions in each state in figure 3. 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 4, 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. Indeed, figure **??** shows that the optimal depth-1 tree achieves exactly the same RL objective value as the optimal policies from figure 3, independently of the discount factor γ.

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 3?

We start by running the standard Q-learning algorithm as presented in algorithm 1 with $\epsilon = 0.3$, $\alpha = 0.1$ over 10,000 time steps. The careful reader might wonder how ties are broken in the $\arg\max$ operation from algorithm 1. While Sutton and Barto break ties by index value in their book (**?**) (the greedy action is the $\arg\max$ 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 5, 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 5, 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 (cf. figure 1) to directly search for policies with good interpretability-performance trade-offs with respect to the original RL objective.

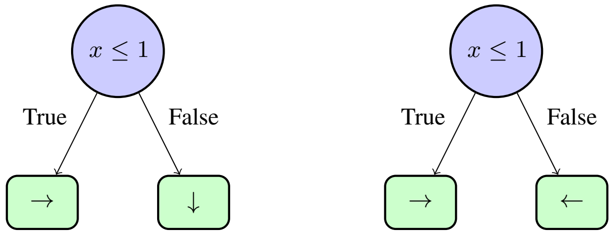


Figure 4. Left, an optimal depth-1 decision tree policy. On the right, a sub-optimal depth-1 decision tree policy.

5. Iterative bounding Markov decision processes

The key thing to know about IBMDPs is that they are, as their name suggests, MDPs (cf definition 3.2). Hence, IBMDPs admit an optimal deterministic Markovian policy that maximizes the RL objective. In this part we will assume that all the MDPs we consider are MDPs with continuous state spaces (cf. section **??**) with a finite set of actions, so we use bold fonts for states and observations as they 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 base MDP–IBMDP states are concatenations of the base MDP state features and some observations. Those observations are information about the base state features that are refined–"iteratively bounded"– at each step. Those observations essentially represent some knowledge about where some base state features lie in the state space. Actions available in an IBMDP are: 1) the actions of the base MDP, that change base state features, and 2) *information gathering* actions that change the aforementioned observations. Now, base actions in an IBMDP are rewarded like in the base MDP, this ensures that the RL objective w.r.t. the base 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 base MDP.

Before showing how to get decision tree policies from IBMDP policies, we give a formal definition of IBMDPs following Topin et. al. (**?**).

Definition 5.1 (Iterative bounding Markov decision process)**.** Given an MDP $\mathcal{M} \equiv \langle S, A, R, T, T_0 \rangle$ (cf. definition 3.2), an associated iterative bounding Markov decision process $\mathcal{M}_{IB}$ is a tuple:

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

, where:

- S are the base MDP state features. Base 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 base 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 base state features and observations Given some base 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 $\boldsymbol{s}_{IB} = (\overbrace{s_1, \dots, s_p}^{\text{base state features}}, \underbrace{L_1, U_1, \dots, L_p, U_p}_{\text{observation}})$.
- A are the base MDP actions.
- A_{info} are *information gathering* actions (IGAs) 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 base MDP actions and information gathering actions $A \cup A_{info}$.
- $R : S \times A \to \mathbb{R}$ is the base 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 base MDP if the action is a base 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 base state features $\boldsymbol{s}_t = (s'_1, s'_2, \dots, s'_p)$ if an IGA $\langle j, v \rangle$ is taken, the new observation is:

$$\boldsymbol{o}_{t+1} = \begin{cases} (L'_1, U'_1, \dots, L'_j, \min\{v, U'_j\}, \dots, L''_p, U'_p) \text{ if } s_j \leq v \\ (L'_1, U'_1, \dots, \max\{v, L'_j\}, U'_j, \dots, L''_p, U'_p) \text{ if } s_j > v \end{cases}$$

  If a base action is taken, the observation is reset to the default base state feature bounds $(L_1, U_1, \dots, L_p, U_p)$ and the base state features change according to the base MDP transition function: $\boldsymbol{s}_{t+1} \sim T(\boldsymbol{s}_t, a_t)$. At initialization, the base state features are drawn from the base MDP initial distribution T_0 and the observation is always set to the default base state features bounds $\boldsymbol{o}_0 = (L_1, U_1, \dots, L_p, U_p)$.

Now remains to extract a decision tree policy for MDP $\mathcal{M}$ from a policy for an associated IBMDP $\mathcal{M}_{IB}$.

5.1. From policies to trees

One can notice that information gathering actions (cf. definition 5.1) resemble the Boolean functions $1_{\{x_{\cdot,j} \leq v\}}$ that make up internal decision tree nodes (cf. figure 2). Indeed,

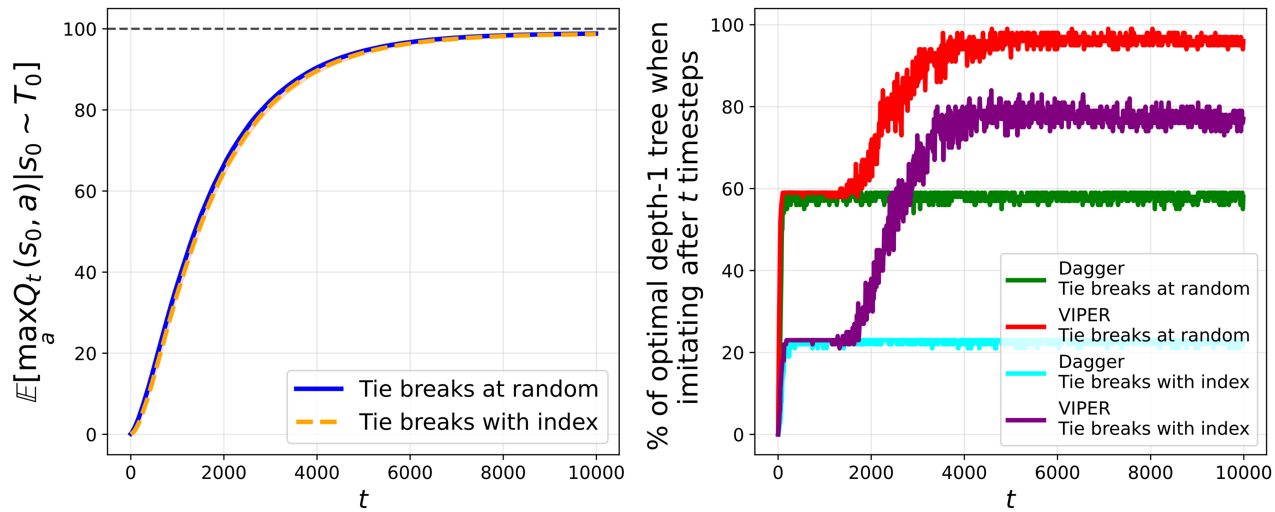


Figure 5. Left, sample complexity curve of Q-learning with default hyperparameters on the 2×2 grid world MDP over 100 random seeds. Right, performance of indirect interpretable methods when imitating the greedy policy with a tree at different Q-learning stages.

a policy taking actions in an IBMDP essentially builds a tree by taking sequences of IGAs (internal nodes) and then a base 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 base rewards, a policy will try to take base actions as often as possible, i.e. build shallow trees with short paths between root and leaves.

Authors from (**?**) show that not all IBMDP policies are decision tree policies for the base MDP. In particular, they show that only deterministic policies depending solely on the observations of the IBMDP are guaranteed to correspond to decision tree policies for the base MDP.

Proposition 5.2 (Deterministic partially observable IBMDP policies are decision tree policies)**.** *Given a base MDP $\mathcal{M}$ $\langle S, A, R, T, T_0\rangle$ and an associated IBMDP $\mathcal{M}_{IB}$ $\langle S \times O, A \cup A_{info}, (R, \zeta), (T_{info}, T, T_0)\rangle$ (cf. definition 5.1), a deterministic partially observable policy $\pi_{po} : O \to A \cup A_{info}$ is a decision tree policy $\pi_{\mathcal{T}} : S \to A$ for the base MDP $\mathcal{M}$.*

Proof. (Sketch) algorithm 2 that takes as input a deterministic partially observable policy (cf. definition 5.2) for an IBMDP $\mathcal{M}_{IB}$ $\langle S \times O, A \cup A_{info}, (R, \zeta), (T_{info}, T, T_0)\rangle$ (cf. definition 5.1), returns a decision tree policy $\pi_{\mathcal{T}}$ (cf. definition **??**) for the base MDP $\mathcal{M}$ $\langle S, A, R, T, T_0\rangle$ and always terminates unless the deterministic partially observable policy takes IGAs in every state. $\square$

While the connections with partially observable MDPs (**??**) is obvious, we defer the implications to chapter **??** as this connection was absent from the original IBMDP paper (**?**).

Algorithm 2 Extract a Decision Tree Policy

Data: Deterministic partially observable policy π_{po} for IBMDP $\langle S \times O, A \cup A_{info}, (R, \zeta), (T_{info}, T, T_0)\rangle$ and IBMDP observation $\boldsymbol{o} = (L'_1, U'_1, \ldots, L'_p, U'_p)$
Result: Decision tree policy $\pi_{\mathcal{T}}$ for MDP $\langle S, A, R, T, T_0\rangle$
Function `Subtree_From_Policy(`$\boldsymbol{o}, \pi_{po}$`)`**:**
  $a \leftarrow \pi_{po}(\boldsymbol{o})$
  if *a is a base action* **then**
    return Leaf_Node(action: a) `// Leaf if base action`
  end
  else
    $\langle i, v\rangle \leftarrow a$ `// Splitting action is feature and value`
    $\boldsymbol{o}_L \leftarrow \boldsymbol{o}$; $\boldsymbol{o}_R \leftarrow \boldsymbol{o}$
    $\boldsymbol{o}_L \leftarrow (L'_1, U'_1, \ldots, L'_j, v, \ldots, L'_p, U'_p)$; $\boldsymbol{o}_R \leftarrow (L'_1, U'_1, \ldots, v, U'_j, \ldots, L'_p, U'_p)$
    $child_L \leftarrow$ 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

Next we present an IBMDP for the MDP of example 3.

5.2. Example: an IBMDP for a grid world

We re-formulate the example MDP (example 3) as an MDP with a finite number of vector valued states (x, y-coordinates). The states are $S = \{(0.5, 0.5), (0.5, 1.5), (1.5, 1.5), (1.5, 0.5)\} \subsetneq [0, 2] \times [0, 2]$. The actions are the cardinal directions $A = \{\to$

$, \leftarrow, \downarrow, \uparrow\}$ that shift the states by one as long as the coordinates remain in the grid. The reward for taking any action is 0 except when in the bottom right state $(1.5, 0.5)$ which is an absorbing state: once in this state, you stay there forever. Standard optimal deterministic Markovian policies were presented for this MDP in example 3.

Suppose an associated IBMDP (definition 5.1) with two IGAs:

- $\langle x, 1\rangle$ that tests if $x \leq 1$
- $\langle y, 1\rangle$ that tests if $y \leq 1$

The initial observation is always the grid bounds $\boldsymbol{o}_0 = (0, 2, 0, 2)$ because the base state features in the grid world are always in $[0, 2] \times [0, 2]$. There are only finitely many observations since with those two IGAs there are only nine possible observations that can be attained from $\boldsymbol{o}_0$ following the IBMDP transitions (cf. definition 5.1). For example when the IBMDP initial base state features are $\boldsymbol{s}_0 = (0.5, 1.5)$, and taking first $\langle x, 1\rangle$ then $\langle y, 1\rangle$ the corresponding observations are first $\boldsymbol{o}_{t+1} = (0, 1, 0, 2)$ and then $\boldsymbol{o}_{t+2} = (0, 1, 1, 2)$. The full observation set is $O = \{(0, 2, 0, 2), (0, 1, 0, 2), (0, 2, 0, 1), (0, 1, 0, 1), (1, 2, 0, 2), (1, 2, 0, 1), (1, 2, 1, 2), (0, 1, 1, 2), (0, 2, 1, 2)\}$. The transitions and rewards are given in definition (cf. definition 5.1).

In figure 6 we illustrate a trajectory in this IBMDP.

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

We attempt to reproduce the results from (?)table 1]topin2021iterative 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 (cf. definition 3.3) for the CartPole. On the other hand, we find that, imitation learning, despite not directly optimizing the RL objective for CartPole, outperforms deep RL that optimizes the interpretable RL objective (cf. definition **??** in which the objective trades off the standard RL objective and interpretability).

Authors of (?) use two deep reinforcement learning baselines (cf. section **??**) to which they apply some modifications in order to learn partially observable policies as required by proposition 5.2 and by the interpretable RL objective (cf. definition **??**). Authors modify the standard DQN (cf. algorithm **??**) to return a partially observable policy. The trained Q-function is approximated with a neural network $O \rightarrow \mathbb{R}^{|A \cup A_{info}|}$ rather than $S \times O \rightarrow \mathbb{R}^{|A \cup A_{info}|}$. In this modified DQN, the temporal difference error target for the Q-function $O \rightarrow A \cup A_{info}$ is approximated by a neural network $S \times O \rightarrow A \cup A_{info}$ that is in turn trained by bootstrapping the temporal difference error with itself. We present the modifications in algorithm 3. Similar modifications are applied to the standard PPO (cf. algorithm **??**) that we present in the appendix (cf. algorithm **??**). In the modified PPO, neural network policy $O \rightarrow A \cup A_{info}$ is trained using a neural network value function $S \times O \rightarrow A \cup A_{info}$ 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 (?)table 1]topin2021iterative in order to study direct deep reinforcement learning of decision tree policies for the CartPole MDP.

7. Experimental setup

7.1. (IB)MDP

We use the exact same base MDP and associated IBMDPs for our experiments as (?) except when mentioned otherwise.

Base MDP The task at hand is to optimize the RL objective (cf. definition 3.3) 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 (cf. definition 5.1) with $\zeta = -0.01$ and 4 information gathering actions. In appendix **??**, we give more details about how the authors of the original IBMDP paper chose the information

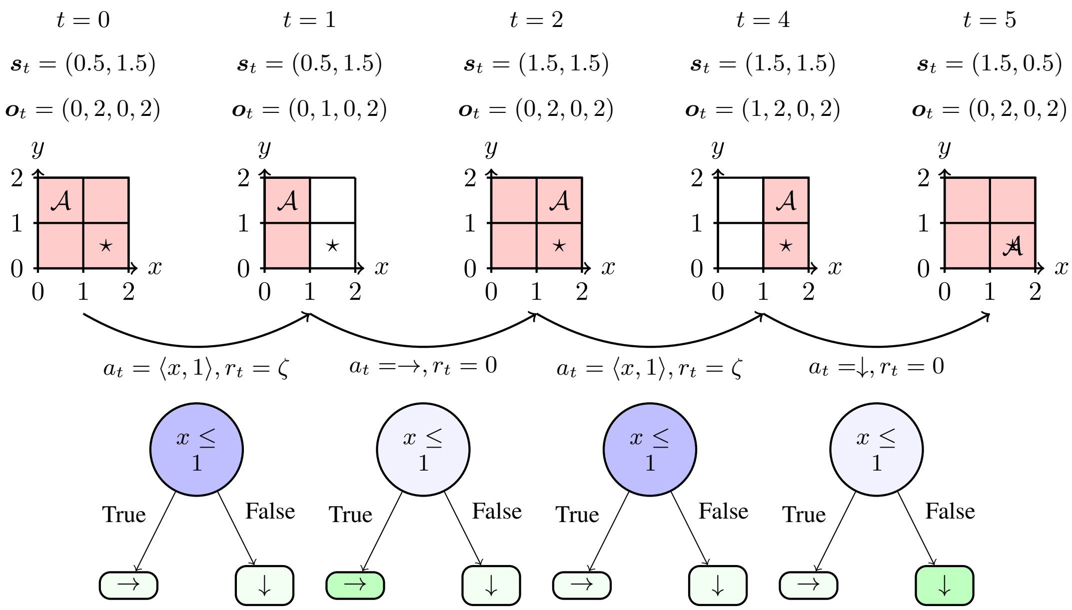


Figure 6. An IBMDP trajectory when the base MDP is 2×2 grid world. In the top row, we write the visited base state features and observations, in the middle row, we graphically represent those, and in the bottom row, we present the corresponding decision tree policy traversal. $\mathcal{A}$ tracks the current state features $\boldsymbol{s}_t$ in the grid. The pink obstructions of the grid represent the current observations $\boldsymbol{o}_t$ of the base state features. When the pink covers the whole grid, the information contained in the observation 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 base state features get. At $t = 0$, the base state features are $\boldsymbol{s}_0 = (0.5, 1.5)$. The initial observation is always the base MDP default state feature bounds, here $\boldsymbol{o}_0 = (0, 2, 0, 2)$ because the base state features are in $[0, 2] \times [0, 2]$. This means that the 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 base state against the value 1 and the reward ζ. 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 base state is below 1, the observation is updated with the refined bounds $\boldsymbol{o}_1 = (0, 1, 0, 2)$, i.e. the pink area shrinks, and the base state features remain unchanged. The agent then takes a base action that is to move right. This gives a reward 0, resets the observation to the original base state feature bounds, and changes the features to $\boldsymbol{s}_2 = (1.5, 1.5)$. And the trajectory continues like this until the absorbing base state $\boldsymbol{s}_5 = (1.5, 0.5)$ is reached.

Algorithm 3 Modified Deep Q-Network. We highlight in green the changes to the standard DQN (cf. algorithm **??**).

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: Partially observable deterministic 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 base state features $\boldsymbol{s}_0 \sim T_0$
  Initialize observation $\boldsymbol{o}_0 = (L_1, U_1, \ldots, L_p, U_p)$

  for *each step t* **do**
    Choose action a_t using ϵ-greedy: $a_t = \text{argmax}_a Q_\theta(\boldsymbol{o}_t, a)$ 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 $(\boldsymbol{s}_i, \boldsymbol{o}_i, a_i, r_i, \boldsymbol{s}_{i+1}) \sim \mathcal{B}$
    $a' = \underset{a}{\text{argmax}} Q_\theta(\boldsymbol{o}_i, a)$
    $y_i = r_i + \gamma Q_{\phi^-}(\boldsymbol{s}_{i+1}, a')$ `// 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
$\pi_{po}(\boldsymbol{o}) = \text{argmax}_a Q_\theta(\boldsymbol{o}, a)$ `// Extract greedy policy`

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.

7.2. Baselines

Modified DQN and Modified PPO as mentioned above, the authors use the modified version of DQN from algorithm 3. 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 (**?**). Authors did not share which activation functions they used so we try both $\tanh$ and relu. For the modified PPO algorithm (cf. algorithm **??**), 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 **??**, **??** and **??**.

Indirect methods We also compare modified RL algorithm to imitation learning (cf. section 4.3). To do so, we use VIPER or Dagger (cf. algorithms **??** and **??**) to imitate greedy neural network policies obtained with standard DQN learning directly on CartPole. We use Dagger to imitate 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 (cf. algorithm **??**) implementation from scikit-learn (**?**) with default hyperparameters and maximum depth of 2 like in (**?**).

7.3. 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 partially observable policy (or Q-function) in an IBMDP, we periodically extract the policy (or Q-function) and use algorithm 2 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 base 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 base MDP is given, i.e. when $a_t \in A$ in the IBMDP (cf. definition 5.1)). 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 base 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 (**?**)table 1]topin2021iterative.

8. Results

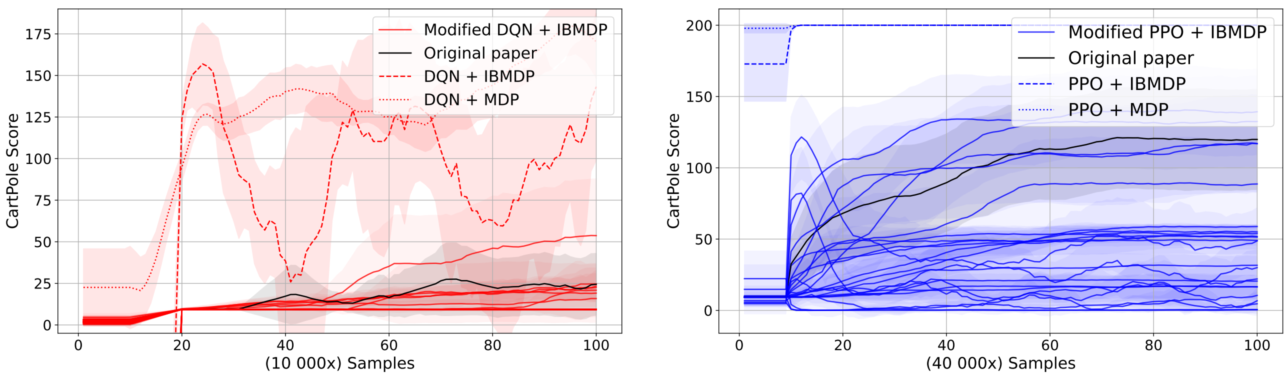
8.1. How well do modified deep RL baselines learn in IBMDPs?

On figure 7a, 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 correspond to the original paper modified DQN, correspond to poor performances on the CartPole MDP. On figure 7b, 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 7a and 7b, we observe that standard RL baselines (RL + IBMDP and RL + MDP), learn better CartPole policies in average than their modified counterparts that learn partially observable policies (cf. proposition 5.2). On figure 7b, 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.

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

On figure 8, 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 (cf. definition 4.4) 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 8 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 9, we present the best decision tree policies for CartPole returned by modified DQN and modified PPO. We used algorithm 2 to extract 20 trees from the 20 partially



(a) DQN variants

(b) PPO variants

Figure 7. Comparison of modified reinforcement learning algorithms on different CartPole IBMDPs. (a) Shows variations of modified DQN and DQN (cf. table **??**), while (b) shows variations of modified PPO and PPO (cf. table **??**). 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.

Figure 8. (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.

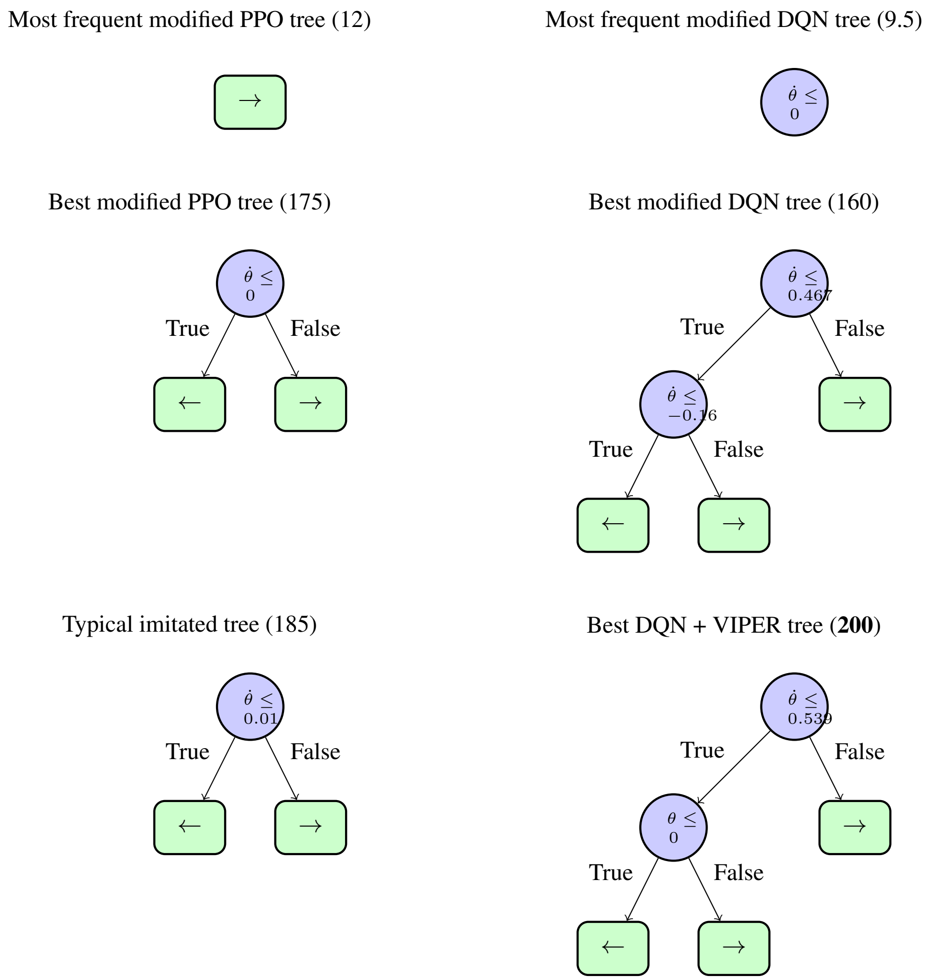


Figure 9. Trees obtained by modified deep RL in IBMDPs against trees obtained with imitation (RL objective value). θ and $\dot{\theta}$ are respectively the angle and the angular velocity of the pole

observable 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 base action forever or repeat the same IGA (cf. definition 5.1) forever.

We have shown that compared to learning non-interpretable but standard Markovian neural network policies for the base MDP or some associated IBMDP, reinforcement learning of partially observable policies in IBMDP is less efficient (cf. figures 7a and 7b). As a consequence, only a handful of modified RL runs are able to learn decision tree policies that are on par with imitated trees (cf. figure 8). In the next chapter, we highlight the connections between direct interpretable RL (cf. definition **??**) and POMDPs to get insights on the hardness of direct reinforcement learning of decision trees.

From the previous chapter, we know that to directly learn decision tree policies that optimize the RL objective (cf. definition 3.3) for an MDP (cf. definition 3.2), one can learn a deterministic partially observable policy that optimizes the interpretable RL objective (cf. definition **??**) in an IBMDP (cf. definition 5.1 and proposition 5.2). Such problems are classical instances of partially observable Markov decision processes (POMDPs) (**??**). This connection with POMDPs brings novel insights to direct reinforcement learning of decision tree policies. In this chapter, all the decision processes have a finite number of vector-valued states and observations. Hence we will use bold fonts for states and observations but we can still use summations rather than integrals when required.

9. Partially observable iterative Markov decision processes

A POMDP is an MDP where the current state is hidden; only some information about the current state is observable.

Definition 9.1 (Partially observable Markov decision process)**.** A partially observable Markov decision process is a tuple $\langle X, A, O, R, T, T_0, \Omega\rangle$ where:

- 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.

Let us define explicitly a partially observable iterative bounding Markov decision process (POIBMDP). It is essentially an IBMDP for which we explicitly define an observation space and an observation function:

Definition 9.2 (Partially observable iterative bounding Markov decision process)**.** a partially observable iterative bounding Markov decision process $\mathcal{M}_{POIB}$ is a tuple:

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

, where $\langle S \times O, A \cup A_{info}, (R, \zeta), (T, T_0, T_{info})\rangle$ is an IBMDP (cf. definition 5.1). The transition function Ω maps concatenation of state features and observations–IBMDP states–to observations, $\Omega : S \times O \to O$, with $P(\boldsymbol{o}|(\boldsymbol{s}, \boldsymbol{o})) = 1$

One can see POIBMDPs as 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 base 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 base MDP state: these are called hidden-mode MDPs (**?**). Following (**?**) we can write the value of a deterministic partially observable policy $\pi : O \to A \cup A_{info}$ in observation $\boldsymbol{o}$.

Definition 9.3 (Partially observable value function)**.** In a POIBMDP (cf. definition 9.2), the expected cumulative discounted reward of a deterministic partially observable 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]learning-pomdp 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 (cf. definition 4.1). We abuse notation and denote both values of observations and values of states by V since the function input is not ambiguous.

The asymptotic occupancy distribution is the probability of a policy π to arrive in $(s, \boldsymbol{o}')$ while observing $\boldsymbol{o}$ in some trajectory. In this chapter, we use reinforcement learning to train decision tree policies for MDPs by seeking deterministic partially observable policies that optimize the interpretable RL objective (cf. definition **??**) in POIBMDPs (cf. definition 9.2).

The goal of the following sections is to see if, unlike indirect approaches tested in section 4.4 (cf. figures **??** and 5), direct approaches can consistently learn the depth-1 decision tree policies that have good interpretability-performance trade-offs for the 2×2 grid world from example 3 (cf. figures 4 and 10). The direct approach optimizes the interpretable RL objective **??** in POIBMDPs (cf. definition 9.2). We will use reinforcement learning to learn deterministic partially observable policies for the IBMDP from example 6 re-written as a POIBMDP. Next we show how to choose γ and ζ in the POIBMDP such that the optimal deterministic partially observable policies w.r.t. the interpretable RL objective correspond exactly to the depth-1 decision tree policies with good interpretability-performance trade-offs. Hence, if we find that RL can consistently find optimal policies w.r.t. interpretable RL objective, it means that this direct approach can consistently find the depth-1 decision tree policies with good interpretability-performance trade-off that indirect approaches could not consistently find.

10. Constructing POIBMDPs whose optimal solutions are depth-1 decision tree policies

Because we know all the base states, all the observations, all the actions, all the rewards and all the transitions of our POIBMDP (cf. example 6), we can compute exactly the values of different deterministic partially observable policies given, ζ the reward for IGAs, and γ the discount factor. Each of those policies can be one of the trees illustrated in figure 10:

- $\pi_{\mathcal{T}_0}$: a depth-0 tree equivalent to always taking the same base action
- $\pi_{\mathcal{T}_1}$: a depth-1 tree equivalent alternating between an IGA and a base action
- $\pi_{\mathcal{T}_u}$: an unbalanced depth-2 tree that sometimes takes two IGAs then a base action and sometimes a an IGA then a base action

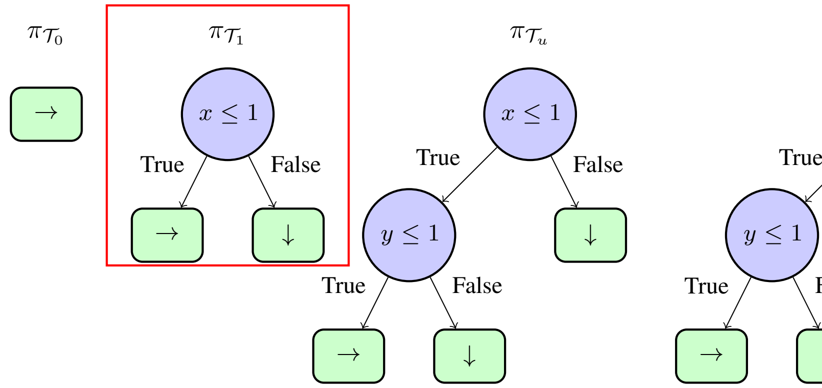


Figure 10. For each decision tree structure, e.g., depth-1 or unbalanced depth-2, we illustrate a decision tree which maximizes the RL objective (cf. definition 3.3) in the grid world MDP.

- $\pi_{\mathcal{T}_2}$: a depth-2 tree that alternates between taking two IGAs and a base action
- an infinite "tree" that only takes IGAs

Furthermore, because from (**?**) we know that for POMDPs, stochastic partially observable policies can sometimes get better expected discounted rewards than deterministic partially observable policies, we also compute the value of the stochastic policy that randomly alternates between two base actions: $\rightarrow$ and $\downarrow$. Those two base actions always lead to the goal state (cf. figure 3).

Proposition 10.1 (Depth-1 decision tree objective value)**.** *The interpretable RL objective value (cf. definition* **??***) of depth-1 decision tree policies with good interpretability–performance trade-offs in the grid world MDP (cf. figures 4 and 10) is* $\frac{4\zeta+\gamma+2\gamma^3+\gamma^5}{4(1-\gamma^2)}$.

We defer the lengthy proofs of this proposition, as well as the interpretable RL objective values of other decision tree policies, to the appendix **??**.

We can now plot, in figure 11, the interpretable RL objective values of the different partially observable policies as functions of ζ when we fix $\gamma = 0.99$. When $\gamma = 0.99$, 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 the optimal one, w.r.t. the interpretable RL objective, deterministic partially observable POIBMDP policy for $0 < \zeta < 1$.

Let us now define a POIBMDP with the grid world (cf. figure 3) as the base MDP, with IGAs as in the IBMDP from example 6, with $\gamma = 0.99$ and $0 < \zeta < 1$ and verify if RL can learn the optimal deterministic partially observable policies w.r.t. the interpretable RL objective, which are equivalent to depth-1 decision tree policies, in this very controlled experiment.

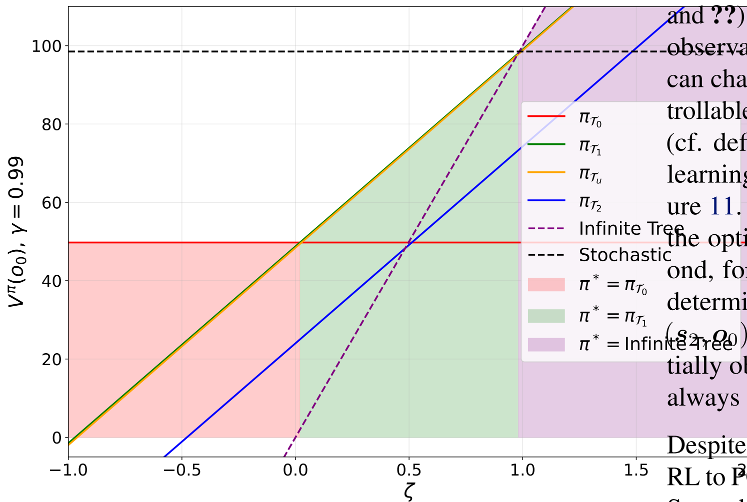


Figure 11. Interpretable RL objective values (cf. definition **??**) of different partially observable policies as functions of ζ. Shaded areas show the optimal *deterministic* partially observable policies in different ranges of ζ values.

11. Reinforcement learning in PO(IB)MDPs

In general, the policy that maximizes the RL objective (cf. definition 3.3) in a POMDP (cf. definition 9.1) maps "belief states" or observation histories (**?**) to actions. Hence, those policies are not solutions to our problem since we require that policies depend only on the current observation. If we did not have this constraint, we could apply any standard RL algorithm to solve POIBMDPs by seeking such policies because both histories and belief states are sufficient statistics for POMDP hidden states (**??**).

In particular, the problem of finding the optimal deterministic partially observable policies for POMDPs is NP-HARD, even with full knowledge of transitions and rewards (**?**)section 3.2]littman1. 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 partially observable policies. Hence it is interesting to study reinforcement learning for finding the best deterministic partially observable policy since it would not search the whole solution space. However applying RL to our interpretable RL objective (cf. definition **??**) is non-trivial.

In (**?**)Fact 2]learning-pomdp, the authors show that the optimal partially observable 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 partially observable policy might not maximize all the values of all observations simultaneously (**?**)Fact 5]learning-pomdp which makes it difficult to use TD-learning (cf. algorithms 1 and **??**). Indeed, doing a TD-learning update of one partially observable value (cf. definition 9.3) with, e.g. Q-learning, can change the value of *all* other observations in an uncontrollable manner because of the dependence in $P^{\pi}((s, \boldsymbol{o}')|\boldsymbol{o})$ (cf. definition 9.3). Interestingly, those two challenges of learning in POMDPs described in (**?**) are visible in figure 11. First, there is a whole range of ζ values for which the optimal partially observable policy is stochastic. Second, for e.g. $\zeta = 0.5$, while a depth-1 tree is the optimal deterministic partially observable policy, the value of state $(s_2, \boldsymbol{o}_0) = (1.5, 1.5, 0, 2, 0, 2)$ is not maximized by this partially observable policy but by the sub-optimal policy that always goes down.

Despite those hardness results, empirical results of applying RL to POMDPs by naively replacing $\boldsymbol{x}$ by $\boldsymbol{o}$ in Q-learning or Sarsa, has already demonstrated successful in practice (**?**). More recently, the framework of Baisero et. al. called asymmetric RL (**??**) has also shown promising results to learn POMDP solutions. 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 algorithm 4 we present asymmetric Q-learning. It is a variant of Q-learning (cf. algorithm 1) that returns a deterministic partially observable policy like modified DQN 3. Given a POMDP, asymmetric Q-learning trains a partially observable Q-function $Q : O \times A \rightarrow \mathbb{R}$ and a Q-function $U : X \times A \rightarrow \mathbb{R}$. The hidden state dependent Q-function U serves as a target in the temporal difference learning update. We also consider an asymmetric version of Sarsa that applies similar modifications to the standard Sarsa (cf. algorithm **??**). We present asymmetric Sarsa in the appendix (cf. algorithm **??**). In (**?**), the authors introduce a policy search algorithm 4.1 that learns a (stochastic) policy $\pi : O \rightarrow \Delta(A)$ and a critic $V : X \rightarrow \mathbb{R}$ using Monte Carlo estimates to guide policy improvement. We also consider this algorithm in our experiments that we call JSJ (for the authors names Jaakkola, Singh, Jordan). We present the JSJ algorithm in the appendix (cf. algorithm **??**). JSJ is equivalent to a tabular asymmetric policy gradient algorithm (cf. algorithm **??**).

Until recently, the benefits of asymmetric RL over standard RL was only shown empirically and only for history-dependent models. The work of Gaspard Lambrechts (**?**) proves that some asymmetric RL algorithms learn better history-dependent **or** partially observable policies for solving POMDPs. This is exactly what we wish for. However, those algorithms are not practical because they require estimations of the asymptotic occupancy distribution

$P^{\pi}((s, \boldsymbol{o}')|\boldsymbol{o})$ (cf. definition 9.3) for candidate policies which in turn would require to gather a lot of on-policy samples. 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.

Note that, in the previous chapter and in the original work of Topin et. al. (?), modified DQN (cf. algorithm 3) and modified PPO (cf. algorithm ??) are respectively asymmetric DQN and asymmetric PPO from (??). In the next section, we use (asymmetric) RL to optimize the interpretable RL objective in POIBMDPs.

Algorithm 4 Asymmetric Q-Learning. We highlight in green the differences with the standard Q-learning (cf. algorithm 1)

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 $U(\boldsymbol{x}, a) = 0$ 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: $a_t = \text{argmax}_a\, Q(\boldsymbol{o}_t, a)$ 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 $\boldsymbol{o}_{t+1} \sim \Omega(\boldsymbol{x}_{t+1})$
    $y \leftarrow r + \gamma U(\boldsymbol{x}_{t+1}, \text{argmax}_{a'}\, Q(\boldsymbol{o}_{t+1}, a'))$ `// TD target`
    $U(\boldsymbol{x}_t, a_t) \leftarrow (1 - \alpha_u) U(\boldsymbol{x}_t, a_t) + \alpha_u y$
    $Q(\boldsymbol{o}_t, a_t) \leftarrow (1 - \alpha_q) Q(\boldsymbol{o}_t, a_t) + \alpha_q y$
    $x_t \leftarrow \boldsymbol{x}_{t+1}$
    $\boldsymbol{o}_t \leftarrow \boldsymbol{o}_{t+1}$
  end
end
$\pi(o) = \text{argmax}_a\, Q(\boldsymbol{o}, a)$ `// Extract greedy policy`

In the next section, we apply asymmetric and standard RL algorithms to the problem of learning the optimal depth-1 tree for the grid world MDP (cf. section 10) by optimizing the interpretable RL objective in POIBMDPs.

12. Results

The results presented in the section show that (asymmetric) reinforcement learning fails for the aforementioned problem. Let us understand why.

12.1. Experimental setup

Baselines: we consider two groups of RL algorithms. The first group is standard tabular RL algorithms to optimize the interpretable RL objective in POIBMDPs; Q-learning, Sarsa, and Policy Gradient with a softmax policy (cf. section 4.1, algorithms 1, ??, and ??). 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 (cf. figure 11).

In addition to the traditional tabular RL algorithms above, we also apply asymmetric Q-learning, asymmetric Sarsa, and JSJ (algorithms 4, ?? and ??). We use at least 200 000 POIBMDP time steps per experiment. Each experiment, i.e an RL algorithm learning in a POIBMDP, is repeated 100 times.

Hyperparameters: For all baselines we use, when applicable, exploration rates $\epsilon = 0.3$ and learning rates $\alpha = 0.1$.

Metrics: We will consider two metrics. First, the sub-optimality gap during training, w.r.t. the interpretable RL objective, between the learned partially observable policy and the optimal deterministic partially observable 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 interpretable RL objective value of the optimal deterministic partially observable policy (cf. figure 11), we can report the *exact* sub-optimality gaps.

Second, 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 partially observable policies (we compute the greedy policy for stochastic policies returned by JSJ and Policy Gradient), from which we can extract the equivalent 100 decision tree policies using algorithm 2 and we can count which one are of e.g. depth 1. This helps understand which trees RL algorithms tend to learn.

12.2. Can (asymmetric) RL learn optimal deterministic partially observable POIBMDP policies?

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 11 that for:

- $\zeta \in [-1, 0]$, the optimal deterministic partially observable policy is a depth-0 tree

- $\zeta \in]0, 1[$, the optimal deterministic partially observable policy is a depth-1 tree
- $\zeta \in [1, 2]$, the optimal deterministic partially observable 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, i.e. not taking the same action forever. In that range, no baseline consistently minimizes the sub-optimality gap.

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 w.r.t. the interpretable RL objective (cf. definition **??**) than e.g. the depth-2 unbalanced trees (cf. figure 11). 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 base action.

However, on the positive side, we observe that asymmetric versions of Q-learning and Sarsa have found the optimal deterministic partially observable 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 partially observable policies in POIBMDPs.

12.3. How difficult is it to learn in POIBMDPs?

In this section we run the same (asymmetric) reinforcement learning algorithms to optimize either the RL objective (cf. definition 3.3) in MDPs (cf. definition 3.2) or IBMDPs (cf. definition 5.1), or the interpretable RL objective in POIBMDPs(cf. definition **??**). This essentially results in three distinct problems:

1. Learning an optimal standard Markovian policy in an MDP, i.e. optimizing the RL objective in an MDP.
2. Learning an optimal standard Markovian policy in an IBMDP, i.e. optimizing the RL objective in an IBMDP.

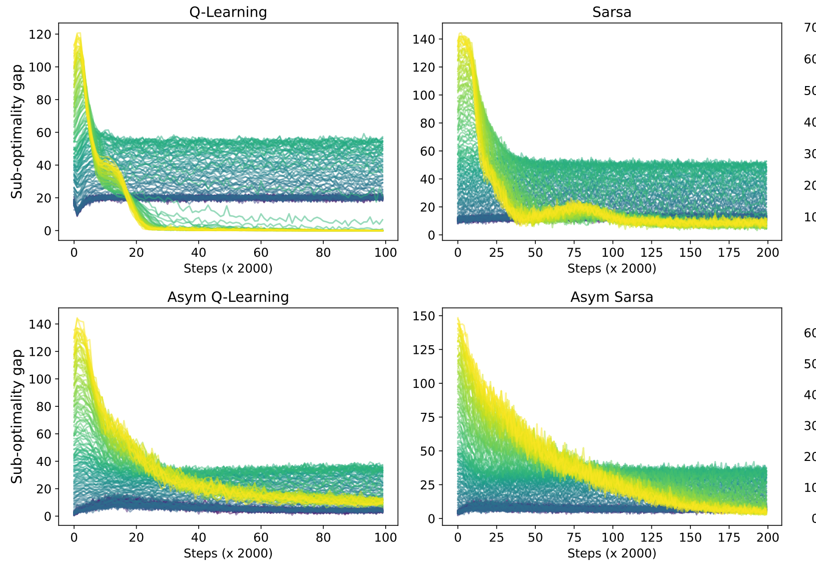


Figure 12. (Asymmetric) reinforcement learning in POIBMDPs. In each subplot, each single line is colored by the value of ζ in the corresponding POIBMDP in which learning occurs. Each single learning curve represent the sub-optimality gap averaged over 100 seeds.

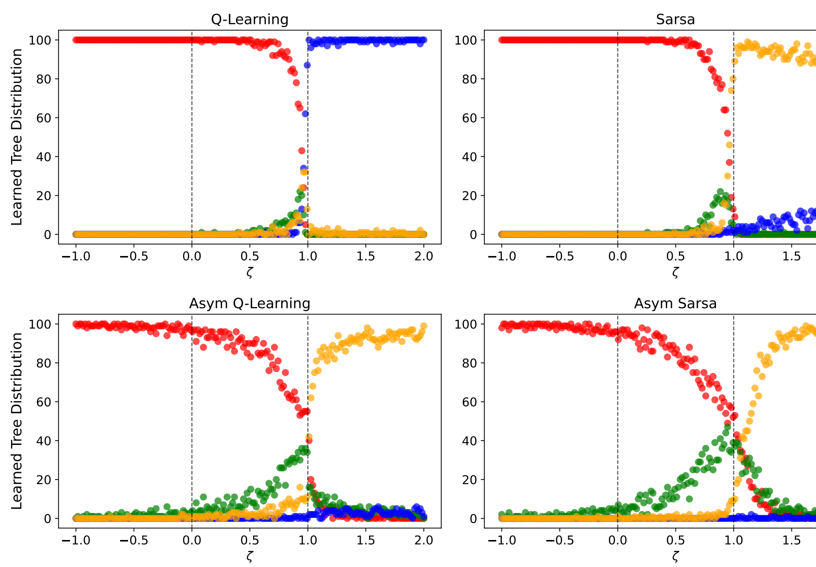


Figure 13. Distributions of final tree policies learned across the 100 seeds. For each ζ value, there are four colored points. Each point represent the share of depth-0 trees (red), depth-1 trees (green), unbalanced depth-2 trees (orange) and depth-2 trees (blue).

3. Learning an optimal deterministic partially observable policy in a POIBMDP.

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 from figure 3 for the grid world from example 3 with $\gamma = 0.99$. Problem 2 is learning one of the optimal standard Markovian deterministic for the IBMDP from figure 6 with $\gamma = 0.99$ and $\zeta = 0.5$. This is similar to the previous chapter experiments where we applied DQN or PPO to an IBMDP for CartPole without constraining the search to partially observable policies (see e.g. figure7b). Problem 3 is what has been done in the previous section to learn deterministic partially observable policies 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 (**?**)). We only provide the detailed hyperparameters for asymmetric Sarsa and an overall summary for all the algorithms in tables 1 and 2. The complete detailed lists of hyperparameters are given in the appendix **??**. 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. For example, for asymmetric Sarsa, we run a total of $10 \times 768 = 7680$ experiments for learning deterministic partially observable policies for a POIBMDP (cf. table 1). To get a success rate, we can simply divide the number of learned depth 1 tree by 7680 (recall that for $\gamma = 0.99$ and $\zeta = 0.5$, the optimal policy is a depth-1 tree (e.g. figure 10) as per figure 11).

The key observations from figure 14 is that reinforcement learning a deterministic partially observable policy in a POIBMDP, is way harder than learning a standard Markovian policy. For example, Q-learning only finds the optimal solution (cf. definition **??**) in only 3% of the experiments while the same algorithms to optimize the standard RL objective (cf. definition 3.3) 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. Another 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?

Table 1. 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 decre
Epsilon Schedules	(0.1, 1), (0.1, 0.99), (0.3, 0.99)	Initial exploration and decre
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 depende
Optimistic	True, False	Optimistic initialization

Table 2. Summary of RL baselines Hyperparameters

algorithm	**Problem**	**Total Hyperparameter Combination**
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

To go even further, on figure 15 we re-run experiments from figure 12 and figure 13 using the top performing hyperparameters for asymmetric Q-learning (given in appendix **??**). While those hyperparameters resulted in asymmetric Q-learning returning 10 of out 10 times an optimal depth 1 tree, the performances didn't transfer. On figure 15 despite higher success rates in the region $\zeta \in]0, 1[$ compared to figure 13.

13. Conclusion

In this chapter, we have shown that direct learning of decision tree policies for MDPs can be reduced to learning deterministic partially observable policies in POMDPs that we called POIBMDPs. By crafting a POIBMDP for which we know exactly the optimal deterministic partially observable policy w.r.t. the interpretable RL objective (cf. definition **??**), we were able to benchmark the sub-optimality of solutions learned with (asymmetric) reinforcement learning.

Across our experiments, we found that no algorithm could consistently learn a depth-1 decision tree policy for a grid world MDP despite it being optimal both w.r.t. the interpretable RL objective and standard RL objective (cf. defi-

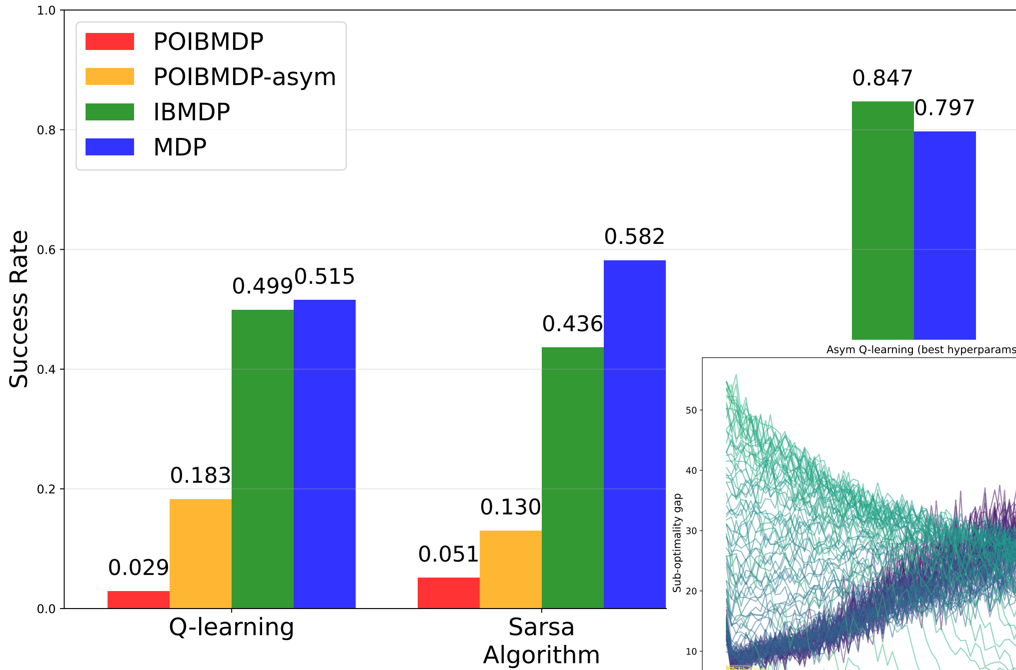
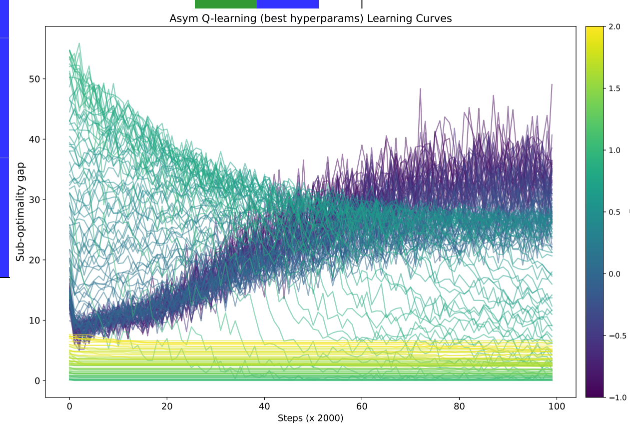


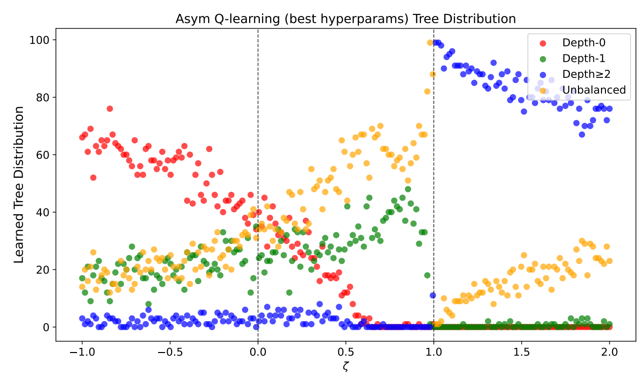
Figure 14. Success rates of different (asymmetric) RL algorithms over thousands of runs when applied to learning deterministic partially observable policies in a POIBMDP or learning deterministic policies in associated MDP and IBMDP.

nition 3.3). When compared to the results of VIPER from figure 5, direct RL is worse at retrieving decision tree policies with good interpretability-performance trade-offs. This echoes the results from the previous chapter in which we saw that direct deep RL performed worse than imitation learning to find decision tree policies for CartPole.

In the next chapter, we find that RL can find optimal deterministic partially observable policies for a special class of POIBMDPs that we believe makes for a convincing argument as to why direct learning of decision tree policies that optimize the RL objective (cf. definition 3.3) is so difficult.



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



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

Figure 15. Analysis of the top-performing asymmetric Q-learning instantiation. (left) Learning curves, and (right) tree distributions across different POIBMDP configurations.

A. You *can* have an appendix here.

You can have as much text here as you want. The main body must be at most 8 pages long. For the final version, one more page can be added. If you want, you can use an appendix like this one.

The `\onecolumn` command above can be kept in place if you prefer a one-column appendix, or can be removed if you prefer a two-column appendix. Apart from this possible change, the style (font size, spacing, margins, page numbering, etc.) should be kept the same as the main body.