Université de Lille

Inria

Numéro d'ordre : 42

Université de Lille
Inria

École doctorale **École Graduée MADIS-631**

Unité de recherche **Centre de Recherche en Informatique, Signal et Automatique de Lille**

Thèse présentée par **Hector Kohler**

Soutenue le **1er décembre 2025**

En vue de l'obtention du grade de docteur de l'Université de Lille et de l'Inria

Discipline **Informatique**

Spécialité **Informatique et Applications**

Interpretabilité via l'Apprentissage Supervisé ou par Renforcement d'Arbres de Décisions

Thèse dirigée par Philippe Preux directeur
Riad Akrour co-directeur

Composition du jury

Rapporteurs	René Descartes	professeur à l'IHP	
	Denis Diderot	directeur de recherche au CNRS	
Examinateurs	Victor Hugo	professeur à l'ENS Lyon	président du jury
	Sophie Germain	mcf à l'Université de Paris 13	
	Joseph Fourier	chargé de recherche à l'INRIA	
	Paul Verlaine	chargé de recherche hdr au CNRS	
Invité	George Sand		
Directeurs de thèse	Philippe Preux	professeur à l'Université de Lille	
	Riad Akrour	Inria	

Colophon

Mémoire de thèse intitulé « Interpretabilité via l'Apprentissage Supervisé ou par Renforcement d'Arbres de Décisions », écrit par Hector Kohler, achevé le 13 août 2025, composé au moyen du système de préparation de document LaTeX et de la classe yathesis dédiée aux thèses préparées en France.

Université de Lille

Inria

Université de Lille
Inria

École doctorale **École Graduée MADIS-631**

Unité de recherche **Centre de Recherche en Informatique, Signal et Automatique de Lille**

Thèse présentée par **Hector Kohler**

Soutenue le **1er décembre 2025**

En vue de l'obtention du grade de docteur de l'Université de Lille et de l'Inria

Discipline **Informatique**

Spécialité **Informatique et Applications**

Interpretabilité via l'Apprentissage Supervisé ou par Renforcement d'Arbres de Décisions

Thèse dirigée par	Philippe Preux	directeur
	Riad Akrour	co-directeur

Composition du jury

Rapporteurs	René Descartes	professeur à l'IHP	
	Denis Diderot	directeur de recherche au CNRS	
Examinateurs	Victor Hugo	professeur à l'ENS Lyon	président du jury
	Sophie Germain	mcf à l'Université de Paris 13	
	Joseph Fourier	chargé de recherche à l'INRIA	
	Paul Verlaine	chargé de recherche hdr au CNRS	
Invité	George Sand		
Directeurs de thèse	Philippe Preux	professeur à l'Université de Lille	
	Riad Akrour	Inria	

Université de Lille

Inria

Université de Lille
Inria

Doctoral School **École Graduée MADIS-631**

University Department **Centre de Recherche en Informatique, Signal et Automatique de Lille**

Thesis defended by **Hector Kohler**

Defended on **December 1, 2025**

In order to become Doctor from Université de Lille and from Inria

Academic Field **Computer Science**

Speciality **Computer Science and Applications**

Interpretability through Supervised or Reinforcement Learning of Decision Trees

Thesis supervised by

Philippe Preux	Supervisor
Riad Akrour	Co-Supervisor

Committee members

Referees	René Descartes	Professor at IHP	
	Denis Diderot	Senior Researcher at CNRS	
Examiners	Victor Hugo	Professor at ENS Lyon	Committee President
	Sophie Germain	Associate Professor at Université de Paris 13	
	Joseph Fourier	Junior Researcher at INRIA	
	Paul Verlaine	HDR Junior Researcher at CNRS	
Guest	George Sand		
Supervisors	Philippe Preux	Professor at Université de Lille	
	Riad Akrour	Inria	

Interpretabilité via l'Apprentissage Supervisé ou par Renforcement d'Arbres de Décisions

Résumé

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Mots clés : apprentissage par renforcement, arbres de décision, interprétabilité, méthodologie

---

Interpretability through Supervised or Reinforcement Learning of Decision Trees

Abstract

In this Ph.D. thesis, we study algorithms to learn decision trees for classification and sequential decision making. Decision trees are interpretable because humans can read through the decision tree computations from the root to the leaves. This makes decision trees the go-to model when human verification is required like in medicine applications. However, decision trees are non-differentiable making them hard to optimize unlike neural networks that can be trained efficiently with gradient descent. Existing interpretable reinforcement learning approaches usually learn soft trees (non-interpretable as is) or are ad-hoc (train a neural network then fit a tree to it) potentially missing better solutions.

In the first part of this manuscript, we aim to directly learn decision trees for a Markov decision process with reinforcement learning. In practice we show that this amounts to solving a partially observable Markov decision process. Most existing RL algorithms are not suited for POMDPs. This parallel between decision tree learning with RL and POMDPs solving help us understand why in practice it is often easier to obtain a non-interpretable expert policy–a neural network–and then distillate it into a tree rather than learning the decision tree from scratch.

The second contribution from this work arose from the observation that looking for a deicison tree classifier (or regressor) can be seen as sequentially adding nodes to a tree to maximize the accuracy of predictions. We thus formulate decision tree induction as sloving a Markov decision problem and propose a new state-of-the-art algorithm that can be trained with supervised example data and generalizes well to unseen data.

Work from the previous parts rely on the hypothesis that decision trees are indeed an interpretable model that humans can use in sensitive applications. But is it really the case? In the last part of this thesis, we attempt to answer some more general questions about interpretability: can we measure interpretability without humans? And are decision trees really more interpretable than neural networks?

Keywords: reinforcement learning, deicision trees, interpretability, methodology

---

Centre de Recherche en Informatique, Signal et Automatique de Lille
Université de Lille-Campus scientifique – Bâtiment ESPRIT – Avenue Henri Poincaré – 59655 Villeneuve d'Ascq

Sommaire

Preliminary Concepts

Interpretable Sequential Decision Making

What is Sequential Decision Making?

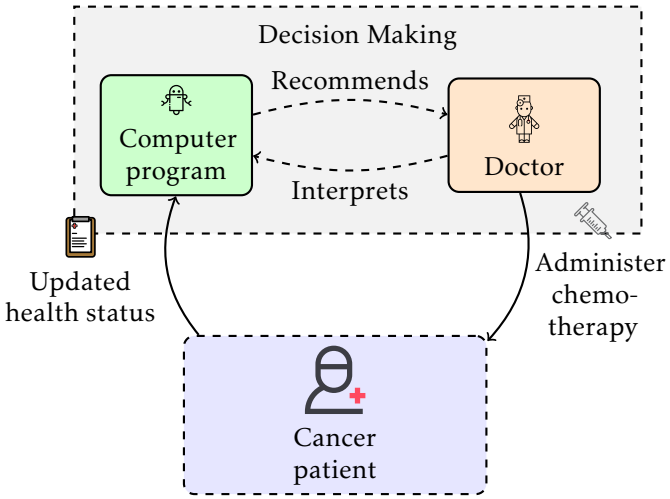


FIGURE 1 – Sequential decision making in cancer treatment. The AI system reacts to the patient's current state (tumor size, blood counts, etc.) and makes a recommendation to the doctor, who administers chemotherapy to the patient. The patient's state is then updated, and this cycle repeats over time.

In this manuscript, we study algorithms for sequential decision making. Humans engage in sequential decision making in all aspects of life. In medicine, doctors have to decide how much chemotherapy to administer based on the patient's current health [33]. In agriculture, agronomists have to decide when to fertilize based on the current soil and weather conditions to maximize plant growth [41]. In automotive settings, the autopilot system has to decide how to steer based on lidar and other sensors to maintain a safe trajectory [58]. These sequential decision making processes exhibit key similarities : an

agent takes actions based on current information to achieve a goal.

As computer scientists, we ought to design computer programs [54] that can help humans during these sequential decision making processes. For example, as depicted in Figure 1, a doctor could benefit from a program that would recommend the "best" treatment given the patient's state. Machine learning algorithms [100] output such helpful programs. For non-sequential decision making, when the doctor only takes one decision and does not need to react to the updated patient's health, e.g. making a diagnosis about cancer type, a program can be fitted to example data : given lots of patient records and the associated diagnoses, the program learns to make the same diagnosis a doctor would given the same patient record, this is *supervised* learning [69]. In the cancer treatment example, the doctor follows the patient over time and adapts treatment to the patient's changing health. In that case, machine learning—and in particular *reinforcement* learning [95]—can be used to teach the program how to take decisions that lead to recovery based on how the patient's health changes from one dose to another. Such machine learning algorithms train increasingly performant models that are deployed to, e.g., identify digits in images [57], control tokamak fusion [24], or write the abstract of a scientific article [30].

However, the key problem behind this manuscript is that the computations performed by these programs cannot be understood and verified by humans : the programs are black-box. Next, we describe the notion of interpretability that is key to ensure safe deployment of computer programs trained with machine learning in critical sectors like medicine.

What is Interpretability ?

Originally, the etymology of "interpretability" is the Latin "interpretabilis", meaning "that can be understood and explained". According to the Oxford English Dictionary, the first recorded use of the English word "interpretability" dates back to 1854, when the British logician George Boole (Figure 2) described the addition of concepts :

> I would remark in the first place that the generality of a method in Logic must very much depend upon the generality of its elementary processes and laws. We have, for instance, in the previous sections of this work investigated, among other things, the laws of that logical process of addition which is symbolized by the sign +. Now those laws have been determined from the study of instances, in all of which it has been a necessary condition, that the classes or things added together in thought should be mutually exclusive. The

> expression x + y seems indeed uninterpretable, unless it be assumed that the things represented by x and the things represented by y are entirely separate ; that they embrace no individuals in common. And conditions analogous to this have been involved in those acts of conception from the study of which the laws of the other symbolical operations have been ascertained. The question then arises, whether it is necessary to restrict the application of these symbolical laws and processes by the same conditions of interpretability under which the knowledge of them was obtained. If such restriction is necessary, it is manifest that no such thing as a general method in Logic is possible. On the other hand, if such restriction is unnecessary, in what light are we to contemplate processes which appear to be uninterpretable in that sphere of thought which they are designed to aid ? [12, p. 48]

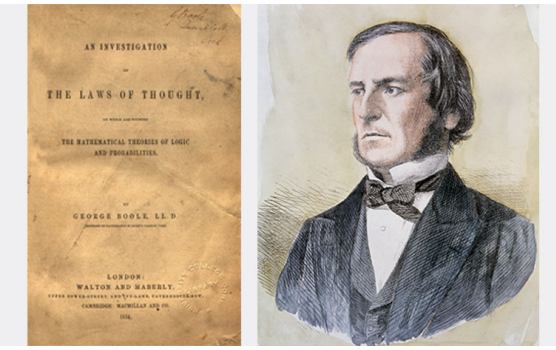
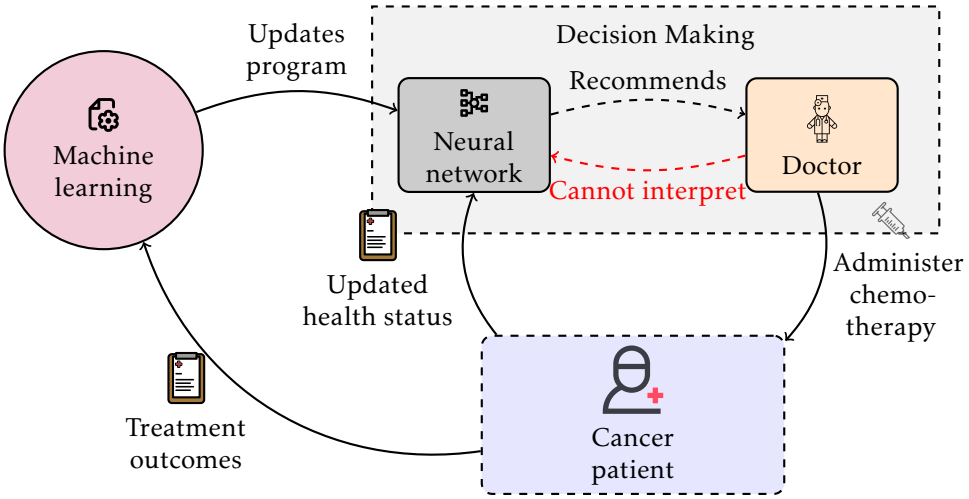


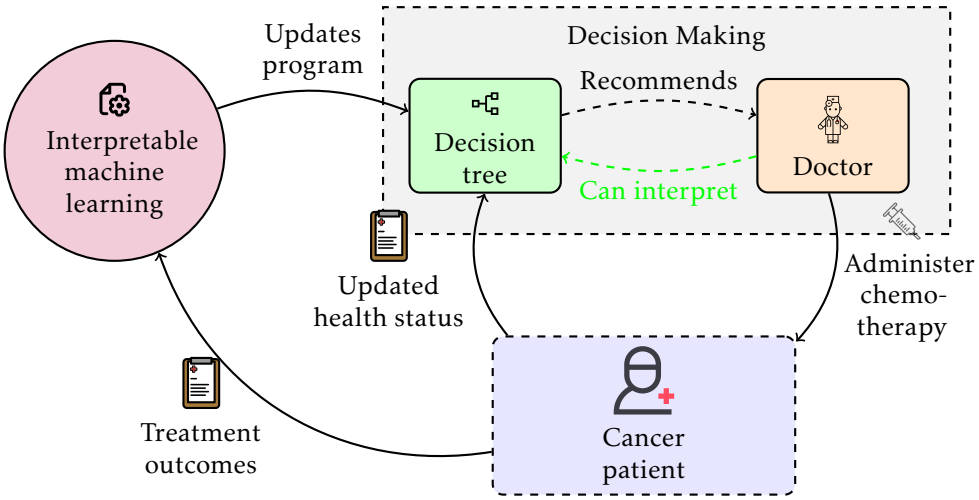
Figure 2 – British logician and philosopher George Boole (1815–1864) next to his book *The Laws of Thought* (1854), the oldest known record of the word "interpretability".

What is remarkable is that the first recorded occurrence of "interpretability" was in the context of logic and computation. Boole asked : *when can we meaningfully apply formal mathematical operations beyond the specific conditions under which we understand them ?* In Boole's era, the concern was whether logical operations like addition could be applied outside their original interpretable contexts—where symbols and their sum represent concepts that humans can understand (e.g., red + apples = red apples). Today, we face an analogous dilemma with machine learning algorithms : black-box programs like neural networks [87], which learn complex, unintelligible representations, are often deployed in contexts where computations should be understood by humans (e.g., in medicine [91]).

In Figure 3a, we illustrate how existing machine learning algorithms *could* be used in principle to help with cancer treatment. In truth, this should be prohibited without



(a) Black-box approach using neural networks



(b) Interpretable approach using decision trees

Figure 3 – Comparison of sequential decision making approaches in cancer treatment. Top : a black-box neural network approach where the doctor cannot interpret the AI's recommendations. Bottom : an interpretable decision tree approach where the doctor can understand and verify the AI's recommendations. Both systems learn from treatment outcomes to improve their recommendations over time.

some kind of transparency in the program's recommendation : why did the program recommend such a dosage ? In Figure 3b, we illustrate how machine learning *should* be used in practice. We would ideally want doctors to have access to computer programs that can recommend "good" treatments and whose recommendations are interpretable.

The key challenge of doing research on interpretability is the lack of formalism ; there is no *formal* definition of what constitutes an interpretable computer program. Hence, unlike for performance objectives, which have well-defined optimization targets (e.g., maximizing accuracy in supervised learning or maximizing rewards over time in reinforcement learning), it is not clear how to design machine learning algorithms to maximize interpretability. Despite this lack of formalism, the necessity of deploying interpretable models has sparked many works, which we present next.

What are existing approaches for learning interpretable programs ?

In this section we follow sections 6 and 7 of Glanois and section 5 of Milani, present next other related works.

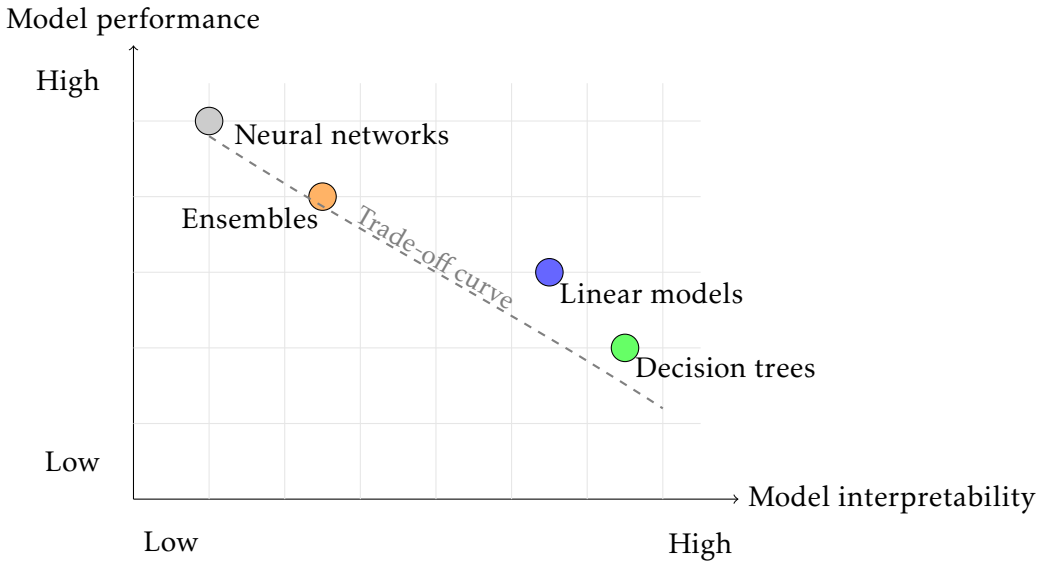


FIGURE 4 – The interpretability–performance trade-off in machine learning. Different model classes are positioned according to their typical interpretability and performance characteristics. The dashed line illustrates the general trade-off between these two properties.

Interpretable machine learning provides either local or global explanations [42]. Global methods output a program whose recommendations can be interpreted without

additional computations, e.g., a decision tree [13]. By contrast, local methods require additional computations but are agnostic to the model class : they can give an *approximate* interpretation of, e.g., neural network recommendations. In Figure 4 we present the popular trade-off between interpretability and performance of different model classes.

The most famous local explanation algorithm is LIME (Local Interpretable Model-agnostic Explanations) [86]. Given a program, LIME works by perturbing the input and learning a simple interpretable model locally to explain that particular prediction. For each individual prediction, LIME provides explanations by identifying which features were most important for that specific decision. Hence, as stated above, LIME needs to learn a local surrogate model per recommendation to be interpreted ; this requires substantial computation. (Figure) HEATMAPS-FEATURE IMPORTANCE : key message is that Global => Local and Local => Global.

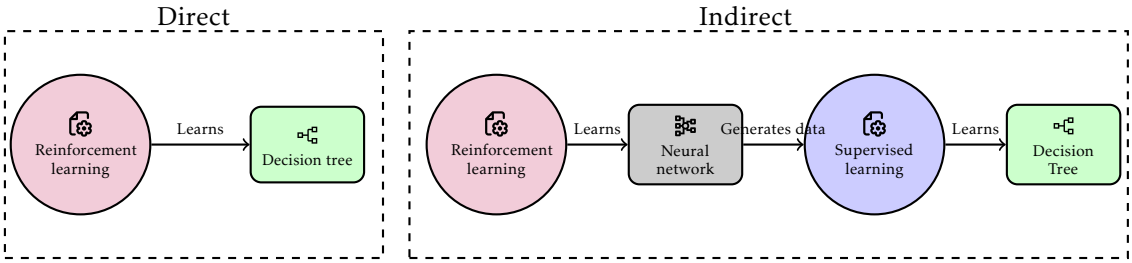


Figure 5 – Comparison of direct and indirect approaches for learning interpretable policies in sequential decision making

Global approaches are either direct or indirect [70]. Direct algorithms, such as decision tree induction [13], directly search a space of interpretable programs (see Figure 4). One key challenge motivating this thesis is that decision tree induction is well-developed for supervised learning but not for reinforcement learning. To directly learn interpretable policies for sequential decision making, one must design new algorithms. Most existing research has focused on working around this limitation and developing indirect methods. Indirect methods for interpretable sequential decision making—sometimes called post hoc—start with reinforcement learning of a non-interpretable program (e.g., a neural network), and then use supervised learning to fit an interpretable model that emulates the black-box program. This approach is called behavior cloning or imitation learning [78, 88], and many works on interpretable sequential decision making use this indirect approach[6, 103]. Figure 5 illustrates the key difference between these two approaches.

Researchers have only recently started to study the advantages of direct over indirect

learning of interpretable programs [97, 99]. In short, direct methods optimize the interpretable program for the actual task (e.g., patient treatment), while indirect methods optimize an interpretable program to match the behavior of a black-box model that was itself optimized for the task. There is no guarantee that optimizing this surrogate objective yields the best interpretability–performance trade-offs. Hence, the ideal solution to interpretable sequential decision making would be global direct algorithms.

Programmatic machine learning

Besides decision trees, programmatic policy representations have been proposed as interpretable alternatives. Programmatically Interpretable Reinforcement Learning (PIRL) [**verma_programmatically_2018**] synthesizes policies in a high-level language via Neurally Directed Program Search (NDPS), typically by imitating a neural oracle. Related work derives programmatic or automata-like abstractions from demonstrations or trained policies [**zhu_inductive_2019**, **burke_explanation_2019**, **Koul_learning_2019**], yielding structured controllers that can sometimes be verified or edited. These approaches share our goal—transparent policies for sequential decision making—but most operate indirectly by distilling from black-box experts, and thus inherit the limitations of imitation.

Learning easy-to-imitate experts

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. Examples include smoothness penalties to reduce erratic behavior [**jia_advanced_2019**], objectives that promote legible and predictable motions [**dragan_legibility_2013**], and representation-shaping losses [**francois-lavet_combined_2019**]. More structured constraints—such as first-order logic [**serafini_logic_2016**] or regional tree regularization [**wu_optimizing_2019**]—can bias decision boundaries toward small trees and improve human simulatability. Because deep models admit many near-optimal solutions, such regularization can often recover policies with similar return but better simulatability. Unlike post hoc distillation, these methods incorporate interpretability early in training, which better aligns optimization with the goal of transparent control.

Conclusion : indirect vs direct approaches Direct and indirect approaches are complementary. Direct methods optimize interpretable policies for the task itself ; indirect methods leverage strong black-box learners and then distill into interpretable mo-

dels. Indirect pipelines are flexible but can miss the best interpretability–performance trade-offs because they optimize a surrogate objective. Hybrids have been proposed [**verma_imitation-projected_2019**], but still inherit some imitation challenges (e.g., non-i.i.d. data). In this thesis, we favor global, direct methods for decision trees whenever possible and use indirect methods (DAgger/VIPER) as baselines.

Misalignment detection

Beyond transparency and readability, interpretability also supports detecting specification or reward misalignment : by exposing the structure and rationales of a learned policy, one can identify goal misspecification or unintended shortcuts (cite ; see also work by Quentin Delfosse, cite).

Explainability

Interpretable RL constrains the policy class so that the computation is transparent by construction. Explainable RL (XRL), by contrast, keeps black-box policies and generates post hoc explanations of their decisions. Typical XRL techniques include visual explanations (t-SNE and saliency maps [**zahavy_graying_2016**, **maaten_visualizing_2008**, **greydanus_visualizing_2018**, **gupta_explain_2020**], attribution such as SHAP [**lundberg_unified_20 wang_attribution-based_2020**], attention-based highlighting [**shi_self-supervised_2020**, **kim_attentional_2020**], and trajectory summaries [**sequeira_interestingness_2020**]) and textual or logic-based explanations [**hayes_improving_2017**, **van_der_waa_contrastive_2018**, **fukuchi_autonomous_2017**]. Causal approaches learn explicit causal models to support contrastive reasoning [**madumal_explainable_2020**, **madumal_distal_2020**]. While useful for insight, these explanations are often subjective, template-bound, or weakly faithful to the underlying computation [**atrey2019exploratory**], and post hoc rationalizations can increase misplaced trust. For safety-critical settings, this motivates our focus on policies that are interpretable by design ; we therefore use XRL primarily for context and comparison, not as our main objective.

Outline of the thesis

In this thesis we study different decision tree learning algorithms in different settings. In the first part of the manuscript, we show that direct decision tree learning methods struggle to find decision trees even for very simple sequential decision making problems. In the second part of the manuscript, we formulate decision tree induction for supervised

learning as solving a sequential decision making problem. Finally, after thoroughly studying decision trees and direct methods, we leverage the diversity and simplicity of *indirect* methods to compare other model classes and show that, in some cases, neural networks can be considered more interpretable than trees, and there exist problems for which there is no need to trade off performance for interpretability. We summarize the outline of the manuscript in Figure 6.

1. Direct reinforcement learning of decision tree policies is hard because it involves POMDPs.
2. One can use Dynamic Programming in MDPs to induce highly performing decision tree classifiers and regressors.
3. In practice, controlling MDPs with interpretable policies does not necessarily decrease performance and has many advantages.

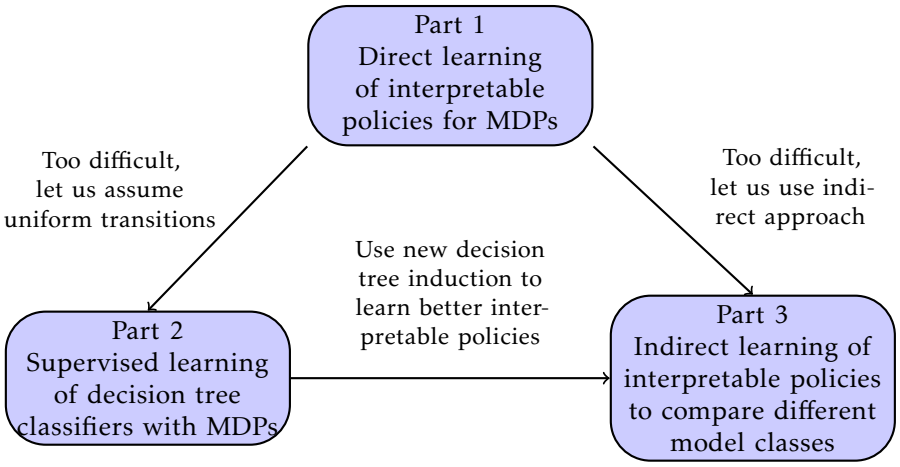


Figure 6 – Thesis structure showing the progression from direct reinforcement learning of decision tree policies (Chapter 1) to simplified approaches : supervised learning with uniform transitions (Chapter 2) and indirect learning methods (Chapter 3).

Technical preliminaries

What are decision trees ?

As mentioned earlier, in contrast to neural networks, decision trees are considered highly interpretable because they apply Boolean operations on the program input without relying on internal complex representations.

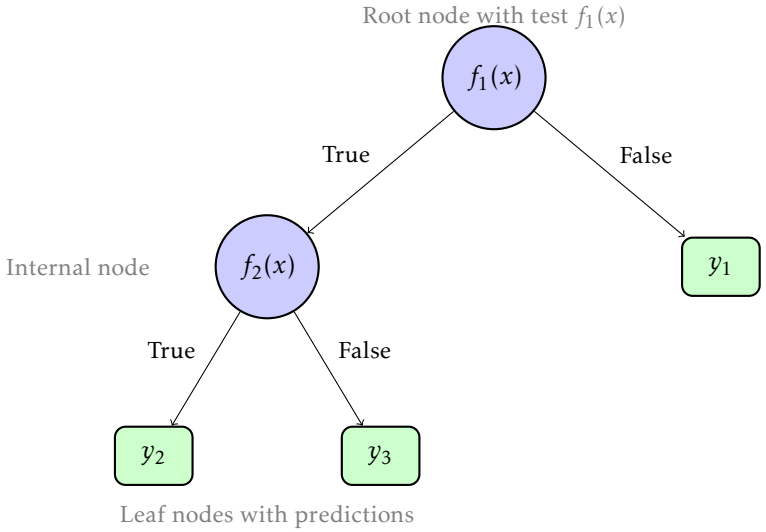


FIGURE 7 – A generic decision tree structure. Internal nodes contain test functions $f_v(x) : \mathcal{X} \to \{0,1\}$ that map input features to boolean values. Edges represent the outcomes of these tests (True/False), and leaf nodes contain predictions $y_\ell \in \mathcal{Y}$. For any input x, the tree defines a unique path from root to leaf.

Definition 1 (Decision tree)**.** *A decision tree is a rooted tree $T = (V, E)$ where :*

- *Each internal node $v \in V$ is associated with a test function $f_v : \mathcal{X} \to \{0,1\}$ that maps input features $x \in \mathcal{X}$ to a boolean.*
- *Each edge $e \in E$ from an internal node corresponds to an outcome of the associated test function.*
- *Each leaf node $\ell \in V$ is associated with a prediction $y_\ell \in \mathcal{Y}$, 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 $T(x) = y_\ell$ where ℓ is the reached leaf.*

How to learn decision trees ?

Training decision trees to optimize the supervised learning objective (cite) has been studied for decades.

Definition 2 (Supervised learning)**.** *We assume that we have access to a set of N examples denoted $\mathcal{E} = \{(x_i, y_i)\}_{i=1}^N$. Each datum x_i is described by a set of p features. $y_i \in \mathcal{Y}$ is the label*

Figure 8 – The american statistician Leo Breiman (1928-2005) author of *Classification and Regression Trees* (1984)

associated with x_i. For a classification task $\mathcal{Y} = \{1, \ldots, K\}$ and for a regression task $\mathcal{Y} \subseteq \mathbb{R}$.

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f(x_i)) + \alpha C(f), \tag{1}$$

where $C : \mathcal{F} \to \mathbb{R}$ is a regularization penalty.

In particular, fitting a decision tree to training data in a greedy manner is fast, and the resulting trees are accurate and generalize well (cite).

The Classification and Regression Trees (CART) algorithm, developed by Leo Breiman and colleagues (Figure 8), is one of the most widely used methods for learning decision trees from supervised data. CART builds binary decision trees through a greedy, top-down approach that recursively partitions the feature space. At each internal node, the algorithm selects the feature and threshold that best splits the data according to a purity criterion such as the Gini impurity for classification or mean squared error for regression. CART uses threshold-based test functions of the form $f_v(x) = \mathbb{I}[x[\text{feature}] \leq \text{threshold}]$, where $\mathbb{I}[\cdot]$ is the indicator function, consistent with the general decision tree definition above. The key idea is to find splits that maximize the homogeneity of the resulting subsets. For classification, this means finding test functions that separate different classes as cleanly as possible. The algorithm continues splitting until a stopping criterion is met, such as reaching a minimum number of samples per leaf or achieving sufficient purity. The complete CART procedure is detailed in Algorithm 1. We use CART as well as other decision tree algorithms in this manuscript and

rely on stable implementations (cite).

In particular, in the second part we will challenge decision tree algorithms that perform better than CART for the supervised learning objective. In the first and third parts, we study CART in conjunction with reinforcement learning as a means to obtain decision trees for sequential decision making.

In the next few sections we present the material related to sequential decision making.

Markov decision processes and problems

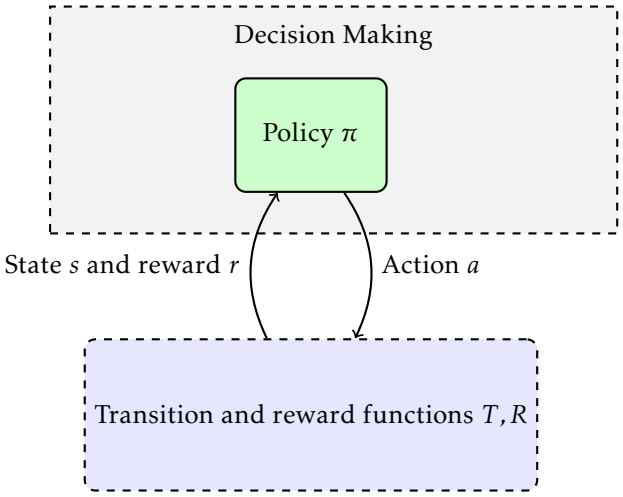


Figure 9 – Markov decision process

Markov decision processes (MDPs) were first introduced in the 1950s by Richard Bellman (cite). 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 (cite), we formally define :

Definition 3 (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* $s \in \mathbb{R}^n$ *representing all possible configurations of the environment.*
- *A is a finite set of actions* $a \in \mathbb{Z}^m$ *available to the agent.*

Algorithme 1 : CART for decision tree induction to optimize the supervised learning objective (cite)

Data : Training data (X, y) where $X \in \mathbb{R}^{n \times p}$ and $y \in \{1, 2, \ldots, K\}^n$
Result : Decision tree T
Function `BuildTree`(X, y) **:**
  if *stopping criterion met* **then**
    return leaf node with prediction MajorityClass(y)
  end
  $(feature, threshold) \leftarrow$ BestSplit(X, y)
  if *no valid split found* **then**
    return leaf node with prediction MajorityClass(y)
  end
  Split data : $X_{left}, y_{left} = \{(x_i, y_i) : x_i[feature] \leq threshold\}$
    $X_{right}, y_{right} = \{(x_i, y_i) : x_i[feature] > threshold\}$
  $left\_child \leftarrow$ BuildTree(X_{left}, y_{left})
  $right\_child \leftarrow$ BuildTree(X_{right}, y_{right})
  return internal node with test function $f_v(x) = \mathbb{I}[x[feature] \leq threshold]$ and children $(left\_child, right\_child)$
Function `BestSplit`(X, y) **:**
  $best\_gain \leftarrow 0$
  $best\_feature \leftarrow None$
  $best\_threshold \leftarrow None$
  for *each feature* $f \in \{1, 2, \ldots, p\}$ **do**
    for *each unique value* v *in* $X[:, f]$ **do**
      $y_{left} \leftarrow \{y_i : X[i, f] \leq v\}$
      $y_{right} \leftarrow \{y_i : X[i, f] > v\}$
      $gain \leftarrow \text{Gini}(y) - \frac{|y_{left}|}{|y|}\text{Gini}(y_{left}) - \frac{|y_{right}|}{|y|}\text{Gini}(y_{right})$
      if $gain > best\_gain$ **then**
        $best\_gain \leftarrow gain$
        $best\_feature \leftarrow f$
        $best\_threshold \leftarrow v$
      end
    end
  end
  return $(best\_feature, best\_threshold)$
Function `Gini`(y) **:**
  return $1 - \sum_{k=1}^{K} \left(\frac{|\{i : y_i = k\}|}{|y|}\right)^2$ `// Gini impurity`
return BuildTree(X, y)

— $R : S \times A \to \mathbb{R}$ *is the reward function that assigns a real-valued reward to each state-action pair.*
— $T : S \times A \to \Delta(S)$ *is the transition function that maps state-action pairs to probability distributions over next states, where* $\Delta(S)$ *denotes the probability simplex over* S*.*
— $T_0 \in \Delta(S)$ *is the initial distribution over states.*

We can also define factored MDPs that represent a special class of MDPs with a continuous state space rather than a finite set of states.

Definition 4 (Factored Markov decision process)**.** *A factored MDP is an MDP (cite) where the state space is a hyperrectangle :* $S \subseteq \mathbb{R}^n$*.*

Now we can also model the goal of the agent. Informally, the goal of an agent is to behave so as to 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 model the agent's goal as an optimization problem as follows.

Definition 5 (Markov decision problem)**.** *Given an MDP* $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$*, the goal of an agent following policy* $\pi : S \to A$ *is to maximize 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 $\gamma \in (0, 1)$ *is the discount factor that controls the trade-off between immediate and future rewards.*

Hence, algorithms presented in this manuscript aim to find solutions to Markov decision problems, i.e., the optimal policy : $\pi^\star = \underset{\pi}{\operatorname{argmax}} J(\pi)$. For the rest of this text, by abuse of notation, we denote both a Markov decision process and the associated Markov decision problem by MDP.

Example : a grid-world MDP

In Figure (cite), we present a very simple MDP (cite). 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 objective (cite). The state space is discrete with state labels representing 2D-coordinates. The actions are to move up, left, right, or down. The MDP transitions to the resulting cell. Only 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. The

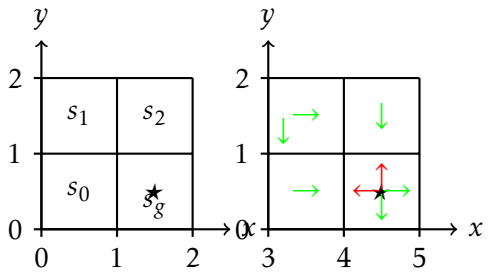


FIGURE 10 – A grid-world MDP (left) and optimal actions w.r.t. the objective (cite) (right).

optimal actions that get to the goal as fast as possible in every state (cell) are presented in green.

Next we present the tools to find solutions to MDPs and retrieve such optimal policies.

Exact solutions for Markov decision problems

It is possible to compute the exact optimal policy $\pi^\star$ using dynamic programming (cite). Indeed, one can leverage the Markov property to find for all states the best action to take based on the reward of upcoming states.

Definition 6 (Value of a state)**.** *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 :

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s')\, V^\pi(s')$$

where $T(s, a, s')$ is the probability of transitioning to state s' when taking action a in state s.

Definition 7 (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 : $V^{\pi^\star}(s)$.*

$$V^\star(s) = V^{\pi^\star}(s) = \max_\pi [J(\pi)]$$

Definition 8 (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.

$$Q^\star(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')\, V^\star(s')$$

Hence, the algorithms we study in the thesis can also be seen as solving the problem : $\pi^\star = \underset{\pi}{\text{argmax}} \mathbb{E}[V^\pi(s_0) | s_0 \sim T_0]$. The well-known Value Iteration algorithm 2 solves this problem exactly (cite).

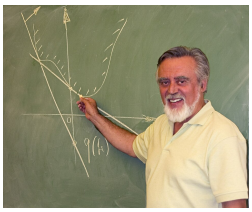
Algorithme 2 : Value Iteration

Data : MDP $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$, convergence threshold θ
Result : Optimal policy π^*
Initialize $V(s) = 0$ for all $s \in S$
repeat
  $\Delta \leftarrow 0$
  for *each state* $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a [R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')]$ `// Bellman optimality update`
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
until $\Delta < \theta$;
for *each state* $s \in S$ **do**
  $\pi^*(s) \leftarrow \arg\max_a [R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')]$ `// Extract optimal policy`
end

More realistically, neither the transition kernel T nor the reward function R of the MDP is 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 (RL), described next.

Reinforcement learning of approximate solutions to MDPs

Reinforcement learning algorithms popularized by Richard Sutton (Figure 11) (cite) don't **compute** an optimal policy but rather **learn** an approximate one based on sequences of observations $(s_t, a_t, r_t, s_{t+1})_t$. RL algorithms usually fall into two categories : value-based (cite) and policy search (cite). The first group of RL algorithms computes an approximation of $V^\star$ using temporal difference learning, while the second class leverages the policy gradient theorem to approximate $\pi^\star$. Examples of these approaches are shown in Algorithms 3 and 4.



(a) D. Bertsekas

(b) M.L. Puterman

(c) A. Barto

(d) R. Sutton

FIGURE 11 – The godfathers of sequential decision making. Andrew Barto and Richard Sutton are the ACM Turing Prize 2024 laureate and share an advisor advisee relationship.

Algorithme 3 : Value-based RL (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 $s \in S, a \in A$
for *each episode* **do**
  Initialize state $s_0 \sim T_0$
  for *each step t* **do**
    Choose action a_t using ϵ-greedy : $a_t = \arg\max_a Q(s_t, a)$ 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
end
$\pi(s) = \arg\max_a Q(s,a)$ `// Extract greedy policy`

Algorithme 4 : Policy Gradient RL (REINFORCE)

Data : MDP $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$, learning rate α, policy parameters θ
Result : Policy π_θ
Initialize policy parameters θ
for *each episode* **do**
  Generate trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$ following π_θ
  for *each time step t in trajectory* **do**
    $G_t \leftarrow \sum_{k=t}^{T} \gamma^{k-t} r_k$ `// Compute return`
    $\theta \leftarrow \theta + \alpha G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$ `// Policy gradient update`
  end
end

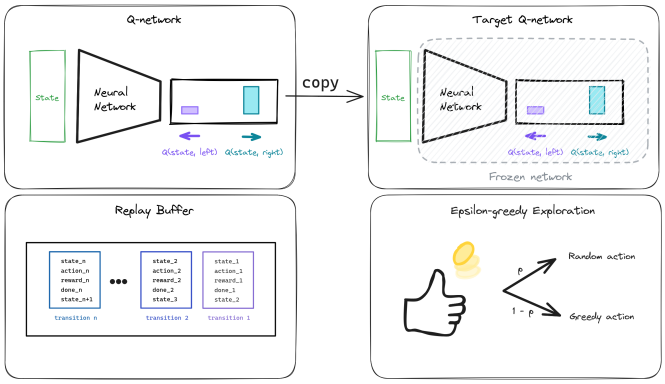


FIGURE 12 – DQN tricks by Antonin Raffin

Both classes of algorithms are known to converge to the optimal value or policy under some conditions (cite) and have known great successes in real-world applications (cite). The books from Puterman, Bertsekas, Sutton and Barto, offer a great overview of MDPs and algorithm to solve them. There are many other ways to learn policies such as simple random search (cite) or model-based reinforcement learning. However, not many algorithms consider the learning of policies that can be easily understood by humans which we discuss next and that is the core of this manuscript.

Deep reinforcement learning for large or continuous state spaces

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

Deep Q-Networks (DQN)(cite)(algo)(fig) was a breakthrough in modern reinforcement learning. Authors successfully extended the Q-learning (algo) to the function approximation setting by introduction target networks to mitigate distributional shift in the td error and replay buffer to increase sample efficiency. DQN achieved super-human performance on a set of Atari games.

Proximal Policy Optimization (PPO)(cite)(algo) is an actor-critic algorithm (cite) optimizing neural network policies. Actor-critic algorithms are instances of policy gradient algorithms (cite) where the returns G_t are also estimated with a neural network. Actor-critic are not as simple efficient as DQN but is known to work well in a variety of domains including robot control in simulation among others (cite).

Algorithme 5 : Deep Q-Network (DQN)

Data : MDP $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$, learning rate α, exploration rate ϵ, Q-network parameters θ, update frequency C
Result : Policy π
Initialize Q-network parameters θ and target network parameters $\theta^- = \theta$
Initialize replay buffer $\mathcal{B} = \emptyset$
for *each episode* **do**
  Initialize state $s_0 \sim T_0$
  for *each step t* **do**
    Choose action a_t using ϵ-greedy : $a_t = \arg\max_a Q_\theta(\boldsymbol{s}_t, a)$ with prob. $1 - \epsilon$
    Take action a_t, observe $r_t = R(s_t, a_t)$ and $\boldsymbol{s}_{t+1} \sim T(\boldsymbol{s}_t, a_t)$
    Store transition $(\boldsymbol{s}_t, a_t, r_t, \boldsymbol{s}_{t+1})$ in $\mathcal{B}$
    Sample random batch $(\boldsymbol{s}_i, a_i, r_i, \boldsymbol{s}_{i+1}) \sim \mathcal{B}$
    $y_i = r_i + \gamma \max_{a'} Q_{\theta^-}(\boldsymbol{s}_{i+1}, a')$ // Compute target
    $\theta \leftarrow \theta - \alpha \nabla_\theta (Q_\theta(\boldsymbol{s}_i, a_i) - y_i)^2$ // Update Q-network
    if t mod $C = 0$ **then**
      $\theta^- \leftarrow \theta$ // Update target network
    end
    $\boldsymbol{s}_t \leftarrow \boldsymbol{s}_{t+1}$
  end
end
$\pi(\boldsymbol{s}) = \arg\max_a Q_\theta(\boldsymbol{s}, a)$ // Extract greedy policy

Algorithme 6 : Proximal Policy Optimization (PPO)

Data : MDP $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$, learning rate α, policy parameters θ, clipping parameter ϵ, value function parameters ϕ
Result : Policy π_θ
Initialize policy parameters θ and value function parameters ϕ
for *each episode* **do**
  Generate trajectory $\tau = (\boldsymbol{s}_0, a_0, r_0, \boldsymbol{s}_1, a_1, r_1, \ldots)$ following π_θ
  for *each time step 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
    $r_t(\theta) \leftarrow \frac{\pi_\theta(a_t|\boldsymbol{s}_t)}{\pi_{\theta_{old}}(a_t|\boldsymbol{s}_t)}$ // 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

In this manuscript we study those two deep reinforcement learning algorithms for various problems and often use their stable implementation (cite).

Imitation learning : a baseline (indirect) global interpretable reinforcement learning method

Unlike PPO or DQN for neural networks, there does not exist an algorithm that trains decision tree policies to optimize the RL objective (cite). 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 with, e.g., DQN or PPO to optimize the RL objective (cite), and then fit, e.g., a decision tree using CART (cite) to optimize the supervised learning objective (cite) with the neural policy's actions as targets. This approach is known as imitation learning and is essentially training a policy to optimize the objective :

Definition 9 (Imitation learning)**.** *Given an expert policy π^* and a policy class Π, the imitation learning objective is to find a policy $\pi \in \Pi$ that minimizes the expected action disagreement with the expert :*

$$\min_{\pi \in \Pi} \mathbb{E}_{s \sim \rho(s)} [\mathcal{L}(\pi(s), \pi^*(s))] \tag{2}$$

where $\rho(s)$ is the state distribution induced by the expert policy and $\mathcal{L}$ is a loss function measuring the disagreement between the learned 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 (cite ; Algorithm 7) is a straightforward way to fit a decision tree policy to optimize the imitation learning objective (cite). VIPER (cite ; Algorithm 7) 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 (cite). 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 the two algorithms extensively throughout the manuscript.

Next we show how to learn a decision tree policy for the Example MDP (cite).

Algorithme 7 : DAgger

Input : Expert policy π^*, MDP M, policy class Π
Output : Fitted student policy $\hat{\pi}_i$
Initialize dataset $\mathcal{D} \leftarrow \emptyset$;
Initialize $\hat{\pi}_1$ arbitrarily from Π;
for $i \leftarrow 1$ **to** N **do**
  if $i = 1$ **then** $\hat{\pi} \leftarrow \pi^*$;
  else $\pi_i \leftarrow \hat{\pi}_i$;
  Sample transitions from M using $\hat{\pi}$;
  Collect dataset $\mathcal{D}_i \leftarrow \{(s, \pi^*(s))\}$ of states visited by $\hat{\pi}$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$;
  Fit classifier/regressor $\hat{\pi}_{i+1}$ on $\mathcal{D}$;
end
return $\hat{\pi}$;

Algorithme 8 : VIPER

Input : Expert policy π^*, Expert Q-function Q^*, MDP M, policy class Π
Output : Fitted student policy $\hat{\pi}_i$
Initialize dataset $\mathcal{D} \leftarrow \emptyset$;
Initialize $\hat{\pi}_1$ arbitrarily from Π;
for $i \leftarrow 1$ **to** N **do**
  if $i = 1$ **then** $\hat{\pi} \leftarrow \pi^*$;
  else $\pi_i \leftarrow \hat{\pi}_i$;
  Sample transitions from M using $\hat{\pi}$;
  Weight each transition by $w(s) \leftarrow V^{\pi^*}(s) - \min_a Q^{\pi^*}(s, a)$;
  Collect dataset $\mathcal{D}_i \leftarrow \{(s, \pi^*(s), w(s))\}$ of states visited by $\hat{\pi}$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$;
  Fit classifier/regressor $\hat{\pi}_{i+1}$ on the weighted dataset $\mathcal{D}$;
end
return $\hat{\pi}$;

Your first decision tree policy

Now the reader should know how to train decision tree classifiers or regressors for supervised learning (cite) using CART. The reader should also know what an MDP is and how to compute or learn policies that optimize the objective (cite) with (deep) reinforcement learning. Finally, the reader should now know how to obtain a decision

tree policy for an MDP through imitation learning by first using RL to get an expert policy and then fitting decision trees to optimize the supervised learning objective (cite), using the expert's actions as labels. Note that, in theory, there is no guarantee that such decision tree policies also maximize the RL objective ; they optimize only the imitation learning objective (cite).

In this section we present the first decision tree policies of this manuscript obtained using DAgger or VIPER after learning an expert policy and expert Q-function for the grid-world MDP (cite) with Q-learning (cite). Recall the optimal policies for the grid-world, taking the green actions in each state in Figure (cite). Among the optimal policies, the ones that take action to 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 a decision tree policy that is optimal and very interpretable (depth 1) : go right if the x-label of the state is greater than 1 and go left otherwise. This tree takes exactly the same actions in the same states as some of the optimal policy from Figure (cite).

In Figure (cite), we present a depth-1 decision tree policy that is optimal w.r.t. the RL objective (cite) and a depth-1 tree that is suboptimal. Indeed, Figure (cite) shows that the optimal depth-1 tree achieves exactly the same RL objective value as the optimal policies from Figure (cite), independently of the discount factor γ.

Now a fair question is : can DAgger or VIPER retrieve such an optimal depth-1 tree given access to an expert optimal policy from Figure (cite)?

We start by running the standard Q-learning algorithm as presented in (cite) with $\epsilon = 0.3$, $\alpha = 0.1$ over 10,000 time steps. The careful reader might wonder how ties are broken in the argmax operation from Algorithm (cite). While Sutton and Barto note (cite) that ties are often broken by index value (the greedy action is the 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 (cite), 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 (cite), 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.

Despite this negative result, we still find that VIPER almost always finds the optimal depth-1 decision tree policy in terms of the RL objective (cite) when ties are broken at random.

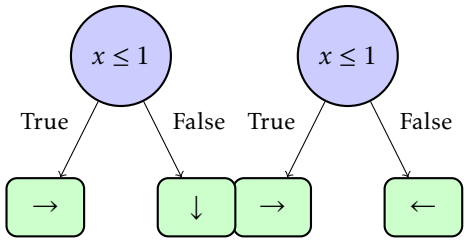


FIGURE 13 – Left, a sub-optimal depth-1 decision tree policy. On the right, an optimal depth-1 decision tree policy.

Now that the reader has seen how to get an interpretable policy for an MDP, we believe it is ready to dive into the contributions of this thesis.

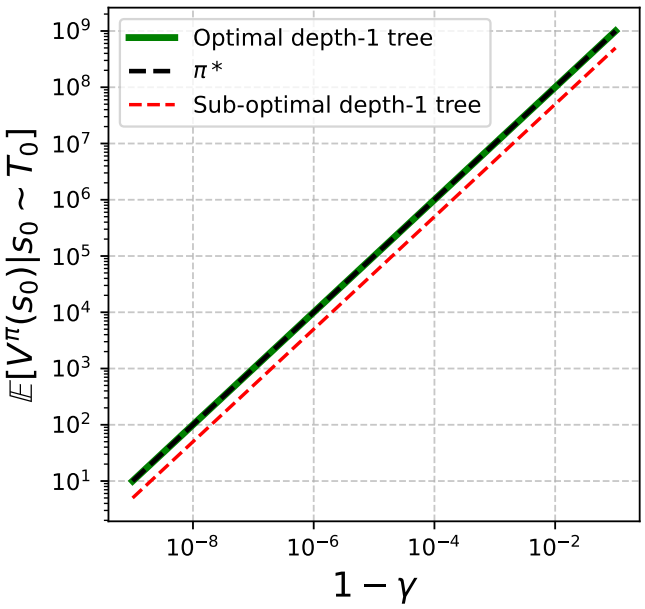


FIGURE 14 – MDP objective values.

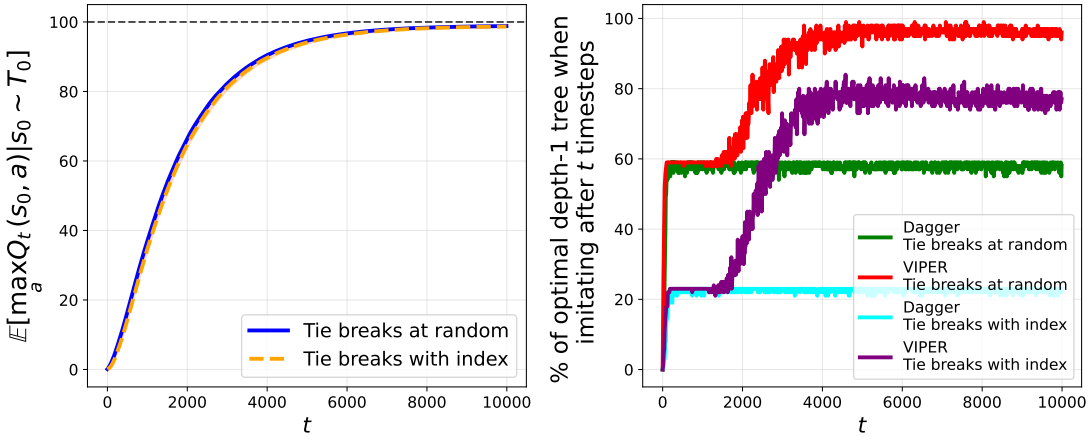


FIGURE 15 – Left, sample complexity curve of Q-learning with default hyperparameters on the 2×2 grid-world MDP over 100 random seeds. Right, performance of indirect interpretable methods when imitating the greedy policy with a tree at different Q-learning stages.

Première partie

A Difficult Problem : Direct Interpretable Reinforcement Learning

Chapitre 1

Introduction

In this part of the manuscript, we show that direct reinforcement learning of decision tree policies for MDPs is often very difficult compared to indirect imitation of a neural network policy with a decision tree.

This first part of the manuscript is organised as follows. In this chapter we present Topin et. al. formal framework for direct reinforcement learning of decision tree policies (cite). In chapter (cite), we reproduce experiments from Topin et. al. where we compare direct deep reinforcement learning of decision tree policies to indirect imitation of a neural networf policy with a decision tree for the simple CartPole MDP. In chapter (cite), we show that the direct approach proposed by Topin et. al. is equivalent to learning deterministic memoryless policy for partially observable MDPs (POMDPs) which is way more difficult that learning the same policies MDPs. In chapter (cite), we show that when doing direct reinforcement learning of decision tree policies for MDPs which transitions are uniform, e.g. when the MDPs encode classification or regression tasks (cite), those POMDP problems are easy to solve.

1.1 Learning Decision Tree policies for MDPs

There already exist algorithms that return decision tree policies for MDPs. 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 depending on the interpretability-performance trade-off to optimize, but are rather “optimized” : the depth, internal nodes arrengement, and state-features to consider in each nodes are fixed *a priori* and only the tested thresholds of each nodes

are optimized similarly to how the weights of a neural network are optimized. As the reader might have guessed, those parametric trees are advantageous in that they can be learned with gradient descent and in the context of decision tree policies, with the policy gradient (cite). The downside of those approaches is that a user cannot know *a priori* what a "good" tree policy structure should be for a particular MDP : either the specified structure is too deep and pruning will be required after training or the tree structure is not expressive enough to encode a good policy. Similar approaches exist in supervised learning exist where a parametric tree is fitted with gradient descent (cite). However their benefit over non-parametric trees have not been shown. When parametric trees are learned for MDPs ; extra stabilizing tricks are required during training such as adaptive batch sizes (cite). Non-parametric trees are the standard model in supervised learning (cite) and can naturally trade-off between interpretability and performances. However, specialized approaches are required since growing a tree from the root in an RL fashion is not possible. Recall that interpretable machine learning algorithms are either indirect. In the introduction (cite), we showed that indirect imitation learning of neural network experts can lead to sub-optimal decision tree policies in terms of the RL objective (cite) because those trees optimize a surrogate imitation loss rather than the RL objective directly.

In 2021, Topin et. al. introduced iterative bouding Markov decision processes (IBMDPs) (cite). Given an MDP for which one wants to learn a decision tree policy, IBMDPs are an augmented version of this base MDP with more state features, more actions, additional rewards and additional transitions. Authors showed that certain IBMDP policies, that can be learned *directly* with RL, are equivalent to decision tree policies for the base MDP.

There also exists more specialized approaches that can return decision tree policies only for small problem classes. In (cite), authors prove that for maze-like MDPs, there always exist an optimal decision tree policy and provide an algorithm to find it. In (cite), authors give an algorithm to return decision tree policies for MDPs that are constrained to have depth less than 3. Finally, in (cite), authors study decision tree policies for planning in MDPs, i.e., when the transitions and rewards are known. In the next section we present IBMDPs as introduced in (cite).

1.2 Iterative Bounding Markov Decision Processes

The key thing to know about IBMDPs is that they are, as their name suggests, MDPs. Hence, IBMPDs inherit MDPs properties such as existence of an optimal deterministic

Markovian policy for problem (cite). The states in an IBMDP are concatenations of the base MDP states and some observations. Those observations are information about the base states that are refined–"iteratively bounded"– at each step and represent a subspace of the base MDP state space. Actions available in an IBMDP are the actions of the base MDP, that change the state of the latter, and *information gathering* actions that change the observation part of the IBMDP state. Now, taking base actions in an IBMDP is rewarded like in the base MDP, this ensures that the base objective, e.g. balancing the pole or treating cancer, is still encoded in the IBMDP reward. When taking an information gathering action, the reward is an arbitrary value supposed to balance performance and interpretability.

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

Definition 10 (Iterative Bounding Markov decision process)**.** *Given a factored (cite) MDP $\mathcal{M}$ (cite) $\langle S, A, R, T, T_0 \rangle$, an associated Iterative Bouding MDP $\mathcal{M}_{IB}$ is a tuple :*

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

- *S the base MDP factored state space. A state $\boldsymbol{s} = (s_1, \dots, s_n) \in S$ has n bounded features $s_i \in [L_i, U_i]$ where $\infty < L_i \leq U_i < \infty \forall 1 \leq i \leq n$.*
- *O are the observations in an IBMDP. They are partial information about the base MDP state features. The set of observations is the bounds of the state features bounds : $O \subsetneq S^2 = [L_1, U_1] \times \cdots \times [L_n, U_n] \times [L_1, U_1] \times \cdots \times [L_n, U_n]$. So the complete IBMDP state space is $S \times O$ the concatenations of state features and observations.*
- *A is the base MDP actions set.*
- *A_{info} are information gathering actions (IGAs) of the form $\langle i, v \rangle$ where i is a state feature index $1 \leq i \leq n$ and v is a real number. 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 that maps base states and actions to a real-valued reward signal.*
- *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 kernel of IBMDPs : Given some observation $\boldsymbol{o}_t = (L'_1, U'_1, \dots, L'_n, U'_n) \in O$ and state features $\boldsymbol{s}_t = (s'_1, s'_2, \dots, s'_n)$ if*

an IGA $\langle i, v \rangle$ is taken, the new observation is :

$$\boldsymbol{o}_{t+1} = \begin{cases} (L'_1, U'_1, \ldots, L'_i, \min\{v, U'_i\}, \ldots, L''_n, U'_n) \text{ if } s_i \leq v \\ (L'_1, U'_1, \ldots, \max\{v, L'_i\}, U'_i, \ldots, L''_n, U'_n) \text{ if } s_i > v \end{cases}$$

If a base action $a_t \in A$ is taken, the new observation is the default state bounds $(L_1, U_1, \ldots, L_n, U_n)$ and the state features change according to the base MDP transitition kernel : $\boldsymbol{s}_{t+1} \sim T(\boldsymbol{s}_t, a_t)$. At initialization, the base state features are drawn from the base MDP T_0 and the observation is always set to the default state features bounds $\boldsymbol{o}_0 = (L_1, U_1, \ldots, L_n, U_n)$.

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

1.2.1 From Policies to Trees

One can notice that information gathering actions resemble the Boolean functions that make up internal decision tree nodes (cite). Indeed, 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.

However, authors from (cite) show that not all IBMDP policies are decision tree policies. In particular they show that only deterministic policies depending solely on the observation part of the IBMDP states are guaranteed to correspond to decision tree policies for the base MDP.

Proposition 1 (Deterministic partially observable IBMDP policies are decision trees)**.** *Given an IBMDP $\mathcal{M}_{IB}$ $\langle S \times O, A \cup A_{info}, (R, \zeta), (T_{info}, T, T_0) \rangle$ 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 factored MDP $\langle S, A, R, T, T_0 \rangle$ (cite).*

Démonstration. (Sketch) Algorithm (cite) that takes as input a deterministic partially observable policy π_{po} for an IBMDP $\mathcal{M}_{IB}$ $\langle S \times O, A \cup A_{info}, (R, \zeta), (T_{info}, T, T_0) \rangle$, returns a decision tree policy $\pi_{\mathcal{T}}$ for $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$ and always terminates unless the deterministic partially observable policy only takes IGAs. □

We defer the connexions with POMDPs to Chapter (cite).

Algorithme 9 : 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 observation IBMDP
$\boldsymbol{o} = (L'_1, U'_1, \ldots, L'_n, U'_n)$
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 $a \in A_{info}$ **then**
    return Leaf_Node(action : a) `// Leaf if base action`
  end
  else
    $\langle i, v \rangle \leftarrow a$ `// Splitting action is feature and value`
    $\boldsymbol{o}_L \leftarrow \boldsymbol{o}$; $\boldsymbol{o}_R \leftarrow \boldsymbol{o}$
    $\boldsymbol{o}_L \leftarrow (L'_1, U'_1, \ldots, L'_i, v, \ldots, L'_n, U'_n)$; $\boldsymbol{o}_R \leftarrow (L'_1, U'_1, \ldots, v, U'_i, \ldots, L'_n, U'_n)$
    $child_L \leftarrow$ 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

1.2.2 Example : an IBMDP for a grid world

Suppose a factored MDP representing a grid world with 4 cells like in Example (cite). The states are the (x, y)-cooridnates $S = \{(0.5, 0.5), (0.5, 1.5), (1.5, 1.5), (1.5, 0.5)\} \subsetneq [0, 2] \times [0, 2]$. The actions are the cardinal directions $A = \{\rightarrow, \leftarrow, \downarrow, \uparrow\}$ 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. Optimal deterministic tabular policies were presented for this MDP in Example (cite).

Suppose an associated IBMDP 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 $\boldsymbol{o}_0 = (0, 2, 0, 2)$. 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 (cite). For example when starting in state $\boldsymbol{s}_0 = (0.5, 1.5)$, 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 by definition (cite).

In Figure (cite) we show a trajectory in this IBMDP.

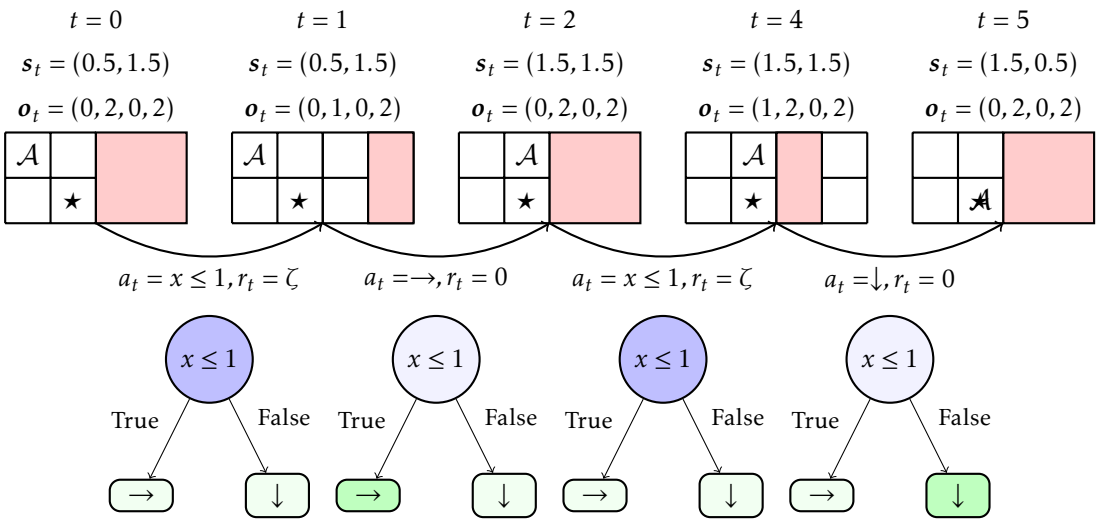


FIGURE 1.1 – An IBMDP trajectory when the base MDP is 2×2 grid world, and equivalent decision tree policy traversal. The pink obstructed squares represent the current bounds over state features. The more IGAs are taking, the more refined the bounds get. At $t = 0$, the state features are $\boldsymbol{s}_0 = (0.5, 1.5)$. The initial observation is always the base MDP default feature bounds, here $\boldsymbol{o}_0 = (0, 2, 0, 2)$ because the base states are in $[0, 2] \times [0, 2]$. The first action is an IGA that tests the feature x of the states against the value 1 and the reward ζ. This transition corresponds to going through an internal node 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 state bounds $\boldsymbol{o}_1 = (0, 1, 0, 2)$ and the base state features remains unchanged. The agent then takes a base action that is to move right. This gives a reward 0, reinitialized the observation to the original bounds, and changes the base state to $\boldsymbol{s}_2 = (1.5, 1.5)$. And the trajectory continues like this until the agent reaches the absorbing base state $\boldsymbol{s}_5 = (1.5, 0.5)$.

1.3 Summary

In this chapter, we presented the approach of Topin et. al. (cite) to find trees that directly optimize the RL objective (cite) rather than the surrogate imitation loss. To achieve that Topin et. al. showed that one can do RL of a partially observable deterministic policy for some IBMDP.

We can thus write an interpretable RL objective as follows :

Definition 11 (Interpretable RL objective)**.** *Given a factored MDP* $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$*, a*

discount factor $\gamma \in (0,1)$*, and an associated IBMDP* $\mathcal{M}_{IB} = \langle S \times O, A \cup A_{info}, (R, \zeta), (T_{info}, T, T_0) \rangle$*, the goal of an interpetable RL agent is to find a partially observable IBMDP policy* $\pi^{\star}_{po} : O \rightarrow A \cup A_{info}$ *such that :*

$$\begin{aligned}\pi^{\star}_{po} &= \underset{\pi_{po}}{argmax} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R((\boldsymbol{s}_t, \boldsymbol{o}_t), a_t) \mid \boldsymbol{s}_0 \sim T_0, a_t = \pi_{po}(\boldsymbol{o}_t), \boldsymbol{s}_{t+1} \sim T(\boldsymbol{s}_t, a_t), \boldsymbol{o}_{t+1} \sim T(\boldsymbol{o}_t, a_t)\right] \\ &= \underset{\pi_{po}}{\text{argmax}} \mathbb{E}[V^{\pi}(s_0, o_0) | s_0 \sim T_0]\end{aligned}$$

In Figure (cite) we summarized the direct reinforcement learning approach of Topin et. al. that we use in the next chapters.

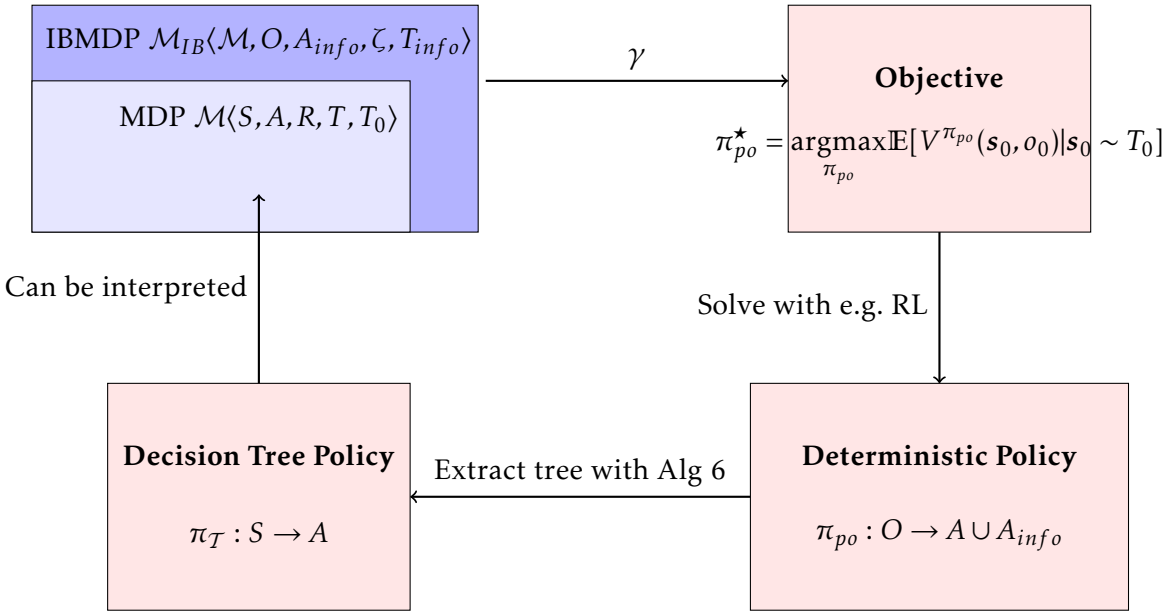


FIGURE 1.2 – A formal framework to learn decision tree policies for MDPs. This include learning a partially observable deterministic policy in a POIBMDP (cite).

Chapitre 2

Direct Deep Reinforcement Learning of Decision Tree Policies

In this chapter, we compare direct deep reinforcement learning of decision tree policies with imitation learning of decision tree policies. We use deep reinforcement learning to optimize the interpretable RL objective from (cite) for the CartPole MDP (cite).

In particular, we attempt to reproduce the results from Table (cite) in which authors use imitation learning or deep reinforcement learning in IBMDPs to learn a detph-2 decision tree for the CartPole MDP. We run the *same* algorithms using the *same* hyperparameters when possible and show that our results differ from (cite) in that we find that direct interpretable reinforcement learning underperforms compared to the indirect imitation learning approach (cite).

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

2.1.1 IBMDP formulation

As described in (cite), given a base MDP $\mathcal{M}\langle S, A, R, T, T_0\rangle$, in order to define an IBMDP $\mathcal{M}_{IB}\langle S \times O, A \cup A, (R, \zeta), (T, T_0, T_{info})\rangle$, the user needs to provide the set of information gathering actions A_{info} and the reward ζ for taking those. Authors propose to parametrize the set of IGAs with $i \times p$ actions $\langle v_k, i\rangle$ with v_k depending on the current observation $\boldsymbol{o}_t = (L'_1, U'_1, \ldots, L'_i, U'_i, \ldots, L'_n, U'_n) : v_k = \frac{k(U'_i - L'_i)}{p+1}$. This parametric IGAs space

keeps the discrete IBMDP action space at a reasonable size while providing a learning agent with varied IGAs to try.

For example, if we define an IBMDP with $p = 3$ for the grid world from Example (cite), 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 agent takes, e.g., IGA $\langle v_2, 2\rangle$, the effective IGA is $\langle v_2 = \frac{k(2-0)}{3+1}, i\rangle = \langle 1, 2\rangle$ 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 $a_t = \langle v_2, 2\rangle$ again, it would be effectively $\langle v_2 = \frac{k(1-0)}{3+1}, i\rangle = \langle 0.5, 2\rangle$. 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, 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 we can either give negative reward for taking D IGAs in a row, or we could terminate the trajectory. The penalization approaches can break the MDP formalism because the reward function now depends on time while it should only depend on states and actions (cite). Similarly, the termination approach requires a transition function that depends on time which also breaks the Makrov property.

We actually find that when $p + 1$, the IBMDP information gathering space parameter, is a prime number, then as a direct consequence of the *Chinese Reminder Theorem* (cite)(proof), the current tree depth is directly encoded in the current observation o_t. Hence, when $p + 1$ is prime, we can control the depth through either transitions or rewards without tracking the time.

We will try various ζ, various p, and various depth control approaches in our experiments but first we describe the reinforcement learning agents.

2.1.2 Modified Deep Reinforcement Learning algorithms

Authors of (cite) use two deep reinforcement learning baselines to which they apply some modifications in order to learn partially observable policies as required by proposition (cite).

The first algorithm is a modified proximal policy optimization algorithm (PPO)(cite)(algo). To satisfy proposition (cite), authors modify the standard PPO and train a neural network policy $O \rightarrow A \cup A_{info}$ while the neural network value function is $S \times O \rightarrow A \cup A_{info}$.

The second deep reinforcement learning algorithm used is the deep Q-networks

algorithm (DQN) (cite)(algo). A similar modification is done to DQN to return a partially observable policy. The trained Q-function is approximated with a neural network $O \to\to \mathbb{R}^{|A \cup A_{info}|}$ 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.

Those two variants of DQN and PPO have first been introduced in (PINTO) for robotics tasks and later studied theoretically to solve POMDPs in Baisero's work (cite) and we defer their connexions to direct interpretable reinforcement learning to the next Chapter.

Next, we present the precise experimental setup we use to reproduce the work of (cite) in order to study direct deep reinforcement learning of decision tree policies for the CartPole MDP.

2.2 Experimental setup

2.2.1 (IB)MDP

We use the exact same MDP and associated IBMDP for our experiments as authors of (cite) except mentioned otherwise.

MDP The problem at hand is the CartPole MDP. At each time step a learning agent observes the cart position velocity and the pole angle and angular velocity, and can take action to push the cart left or right. While the cart is roughly balanced, i.e., while the cart angle remains in some fixed range, the agent gets a positive reward. If the cart is out of balance ; the agent goes to an absorbing terminal state and gets 0 reward forever. In practice, authors use the gymnasium `CartPole-v0` implementation (cite) of the CartPole MDP in which trajectories are truncated after 200 timesteps making the maximum cumulative reward over a trajectory to be 200. Since the IBMDP definition requires the MDP state space to be a factor of bounded segments, authors bound the the state space of the CartPole MDP to be in $[-2, 2] \times [-2, 2] \times [-0.14, 0.14] \times [-1.4, 1.4]$.

IBMDP Authors define the associated IBMDP with $\zeta = -0.01$ and parametric information gathering action space defined by $p = 3$. In addition we also try $\zeta = 0.01$ and $p = 2$. The discount factor used by the authors is $\gamma = 1$.

We potentially differ from the original paper setting in the way we handle maximum

Algorithme 10 : Modified Deep Q-Network (DQN)

Data : IBMDP $\mathcal{M}_{IB}\langle S \times O, A \cup A, (R, \zeta), (T, T_0, T_{info})\rangle$, learning rate α, exploration rate ϵ, partially observable Q-network parameters θ, Fully observable Q-network parameters ϕ, replay buffer $\mathcal{B}$, update frequency C

Result : Partially pbservable deterministic policy π_{po}

Initialize partially observable Q-network parameters θ

Initialize fully observable Q-network parameters ϕ and target network parameters $\phi^- = \phi$

Initialize replay buffer $\mathcal{B} = \emptyset$

for *each episode* **do**

  Initialize state $s_0 \sim T_0$

  Initialize state $o_0 = (L_1, U_1, \ldots, L_n, U_n)$

  for *each step t* **do**

    Choose action a_t using ϵ-greedy : $a_t = \arg\max_a Q_\theta(o_t, a)$ with prob. $1 - \epsilon$

    Take action a_t, observe r_t

    Store transition $(s_t, o_t, a_t, r_t, s_{t+1})$ in $\mathcal{B}$

    Sample random batch $(s_i, o_i, a_i, r_i, s_{i+1}) \sim \mathcal{B}$

    $a' = \underset{a}{\text{argmax}} Q_\theta(o_i, a)$

    $y_i = r_i + \gamma Q_{\phi^-}(s_{i+1}, a')$ // Compute target

    $\phi \leftarrow \phi - \alpha \nabla_\phi (Q_\phi(s_i, a_i) - y_i)^2$ // Update fully observable Q-network

    $\theta \leftarrow \theta - \alpha \nabla_\theta (Q_\theta(o_i, a_i) - y_i)^2$ // Update partially observable Q-network

    if $t \bmod C = 0$ **then**

      $\theta^- \leftarrow \theta$ // Update target network

    end

    $s_t \leftarrow s_{t+1}$

    $o_t \leftarrow o_{t+1}$

  end

end

$\pi_{po}(o) = \arg\max_a Q_\theta(o, a)$ // Extract greedy policy

Algorithme 11 : Proximal Policy Optimization (PPO)

Data : IBMDP $\mathcal{M}_{IB}\langle S \times O, A \cup A, (R, \zeta), (T, T_0, T_{info})\rangle$, learning rate α, policy parameters θ, clipping parameter ϵ, value function parameters ϕ
Result : Partially observable stochastic policy π_{po_θ}
Initialize policy parameters θ and value function parameters ϕ
for *each episode* **do**
  Generate trajectory $\tau = (s_0, o_0, a_0, r_0, s_1, o_1, a_1, r_1, \ldots)$ following π_θ
  for *each timestep t in trajectory* **do**
    $G_t \leftarrow \sum_{k=t}^{T} \gamma^{k-t} r_k$ // Compute return
    $A_t \leftarrow G_t - V_\phi(s_t)$ // Compute advantage
    $r_t(\theta) \leftarrow \frac{\pi_{po_\theta}(a_t|o_t)}{\pi_{po_{\theta\,old}}(a_t|o_t)}$ // 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
    $\phi \leftarrow \phi + \alpha \nabla_\phi (G_t - V_\phi(s_t))^2$ // Value function update
  end
  $\theta_{old} \leftarrow \theta$ // Update old policy
end

depth limiation. Indeed authors restrain the learning of policies to be equivalent to depth-2 trees but don't detail how they do so. We hence try two different approaches as mentioned in the previous secion : terminating trajectories if the agent takes too much information gathering in a row or simply giving a reward of -1 to the agent everytime it takes an information gathering action past the depth limit. We will also try IBMDPs where we do not limit the maximum depth for completeness.

2.2.2 Baselines

Modified DQN as mentioned above, authors use a modified version (cite) of the DQN algorithm (cite). 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 linarly 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 exploartion during training (change the initial ϵ value to 0.9). We use the same optimizer (RMSprop (cite) with hyperparmeter 0.95 and learning rate 2.5×10^{-4}) to update the Q-networks.

Authors did not share what DQN implementation they used so we use the stable-baselines3 one (cite). Authors did not share what activations they used so we try both

tanh() and relu().

Modified PPO for the modified PPO algorithm, we can exactly match the authors hyperparameters since they use the open source stable-baselines3 implementation of PPO.

Similarly to authors of (cite) we train Modified DQN on 1 million timesteps and Modified PPO on 4 million timesteps.

DQN and PPO We also benchmark the non-modified standard DQN and PPO when learning fully observable IBMDP policies $\pi : S \times O \rightarrow A \cup A_{info}$ and when learning standard $\pi : S \rightarrow A$ policies directly in the CartPole MDP.

We summarize hyperpameters for the IBMDP and for the learning algorithms in Tables (cite) and (cite).

Tableau 2.1 – 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 p	2, 3
Information gathering actions rewards ζ	-0.01, 0.01
Depth control	Done signal, negative reward, none

Tableau 2.2 – (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 stard	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()

Tableau 2.3 – (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(), relu()

Indirect methods We also compare Modified RL algorithm to imitation learning. To do so, we use Viper or Dagger (cite) to imitate greedy neural network policies obtained with standard DQN learning directly on CartPole. And 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 greedy tree classifier from scikit-learn with default hyperparameters and maximum depth of 2.

2.2.3 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.

For Modified RL algorithms that learn a partially observable policy (or Q-function) in an IBMPD, we periodically extract the policy (or Q-function) and use Alg 6 (cite) 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 standard RL applied to IBMDPs, since we can't deploy learned policies directly to the base MDP as the state dimensions mismatch, we periodically evaluate those fully observable IBMDP policies periodically in a copy of the training IBMDP in which we fix $\zeta = 0$ ensuring that the cumulative copied IBMDP reward only corresponds to rewards from the base CartPole MDP. Similarly, we do 100 trajectories of the extracted policies in the copied IBMDP and report the average cumulative reward.

For RL applied directly to the base MDP we can just periodically extract the learned policies and evaluate them on 100 trajectories CartPole control trajectories.

Since imitation learning baselines train offline, i.e, on a fixed dataset, their perfor-

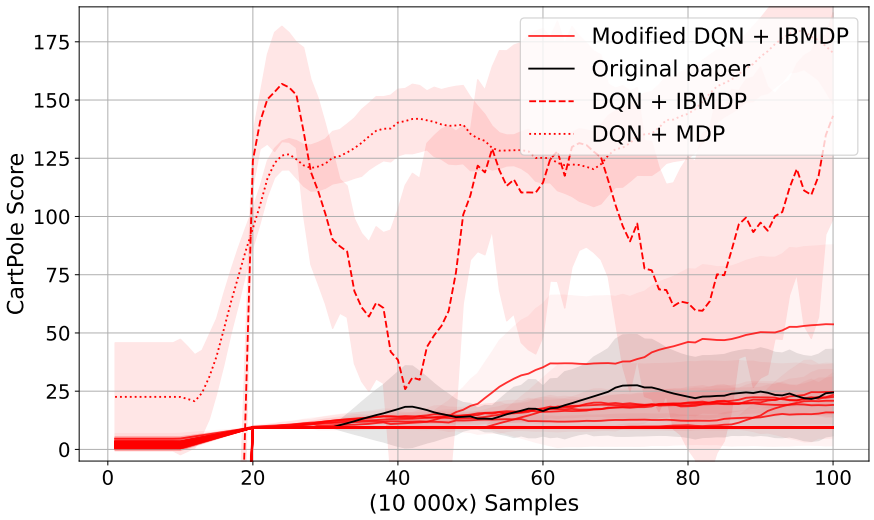


FIGURE 2.1 – Variations of Modified DQN, DQN, on CartPole IBMDP variations. We give different line-styles for the learning curves for DQN applied directly on CartPole and DQN applied on the IBMDP. Since there are multiple possible candidates for the original paper hyperparametrs, we choose to color the (Modified DQN variant, IBMDP variant) pair that resulted in the best decision tree policy on CartPole among the instances that could match the original paper. Shaded areas represent the confidence interval at 95% at each measure on the y-axis.

mances cannot 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 the fitted decision trees are then evaluated on 100 CartPole trajectories.

Every single combination of IBMDP and Modified RL hyperparameters is run 20 times. For standard RL on either an IBMDP or an MDP with use the paper's original hyperparameters when they were spicified, with depth control using negative rewards, tanh() activations, and we repeat this training 20 times.

Next, we present our results and discuss the reproducibility and limitations of the original approach presented in (cite).

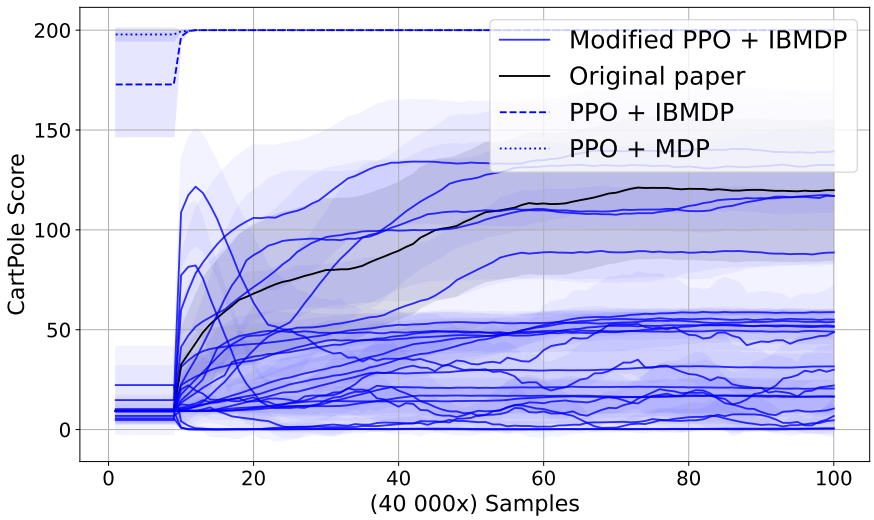


FIGURE 2.2 – Variations of Modified PPO, PPO, on CartPole IBMDP variations. We give different line-styles to the learning curves for PPO applied directly on CartPole and PPO applied on the IBMDP. Since there are multiple possible candidates for the original paper hyperparametrs, we choose to color the (Modified PPO variant, IBMDP variant) pair that resulted in the best decision tree policy on CartPole among the instances that could match the original paper. Shaded areas represent the confidence interval at 95% measure on the y-axis.

2.3 Results

2.3.1 How well do Modified Deep RL baselines learn in IBMDPs ?

On Figure (cite), 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 CartPole in average–the curves flatten at the end of the x-axis and have low y-values–. In, particular, among all the learning curves that could possibly correspond to the original paper Modified DQN, the learning curve with highest final y-value is converging to decision tree policies for CartPole high poor performances.

On Figure (cite) we observe that Modified PPO finds decision tree policies with 150 cumulative rewards towrads the end of training. The performance difference with Modified DQN could be because we trained longer, like in the original paper.

However it could also be because DQN-like algorithm with those hyperpameters struggle in general CartPole MDP or IBMDPs.

Indeed, on Figures (cite) and (cite), we observe that baselines seeking fully observable policies (RL + IBMDP and RL + MDP), learn better CartPole policies in average for both DQN and PPO-like baselines. We do notice that for DQN-like baselines, learning seems difficult in general indepedently of the setting while on Figure (cite) it is clear that for PPO baselines seeking fully observable, learning is super efficient and agent find optimal policies with reward 200.

In Tables (cite) and (cite) 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 CartPole control.

Tableau 2.4 – Top 5 Hyperparameter Configurations for Modified DQN + IBMDP, bold font represent the original paper hyperparameters.

Rank	p	Depth control	Activation	Exploration	ζ	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

Tableau 2.5 – Top 5 Hyperparameter Configurations for Modified PPO + IBMDP, bold font represent the original paper hyperparameters.

Rank	p	Depth Control	Activation	ζ	Final Performance
1	3	reward	relu()	0.01	139
2	3	done	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

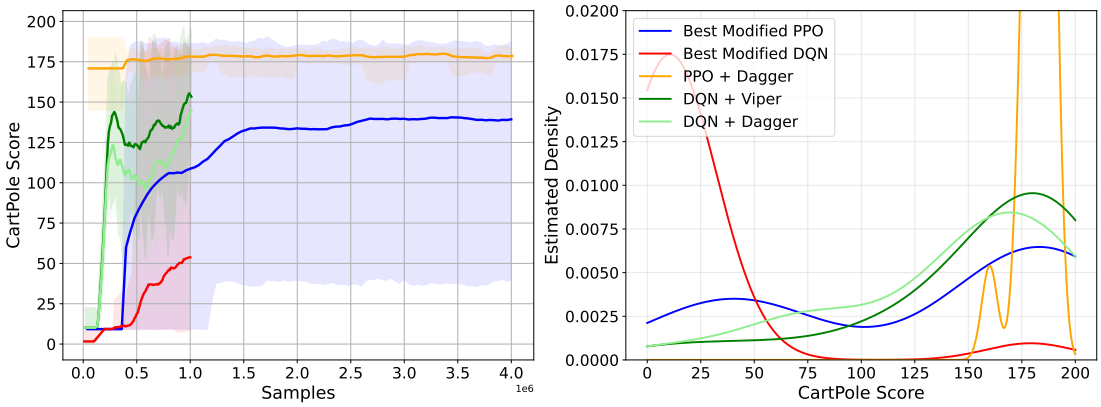


Figure 2.3 – (left) Mean performance of the best Modified PPO on the best IBMDP with shaded areas representing the min and max performance over the 20 seeds during training. (right) Histogram of the final decision tree policies performances.

2.3.2 How does direct interpretable reinforcement learning perform compared to the indirect approach?

On Figure (cite), we isolate the best performing baselines that learn decision tree policies for CartPole using either Modified DQN or Modified PPO and compare them with imitation learning baselines that use the surrogate objective (cite) to find CartPole decision tree policies. We find that despite having poor performances in *average* (c.f. Figure (cite)), the deep reinforcement learning baselines find very good decision tree policies and also extremly poor ones as shown by the min-max shaded areas on the left of Figure (cite) and the estimated density of final trees performances on Figure (cite). In fact, when compared with imitation learning baselines, that fit the policy of a PPO + MDP or fit the greedy policy w.r.t to the Q-network of a DQN + MDP, when running e.g., 20 seeds of direct reinforcement learning baselines, there are some seeds that can find good decision tree policies as fast as the indirect imitation learning baselines : if

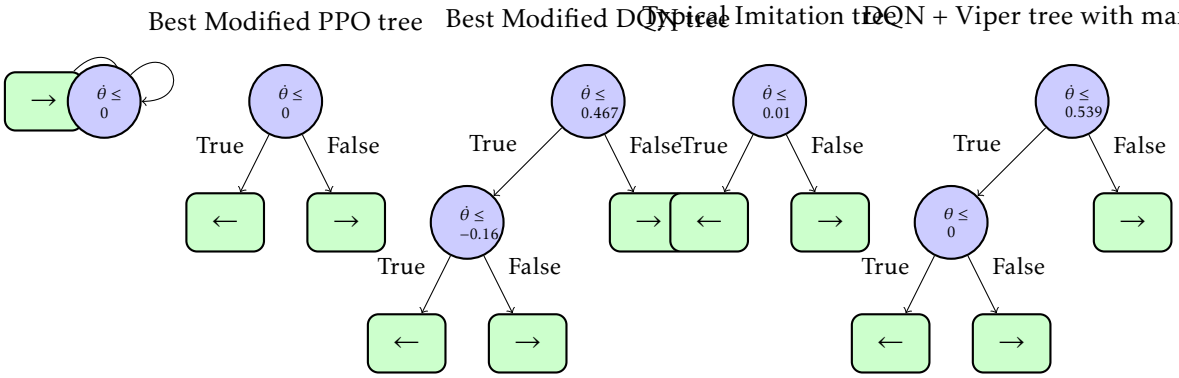


FIGURE 2.4 – Trees obtained by DRL against trees obtained with imitation.

we look at the shaded area of Modified DQN on Figure (cite), we can observe that it is consitently overlapping with an inderect method curve. However this is not desirable, a user want an agent that can consistantly find good decision tree policy. As shown by the estimated densities (cite), 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 final trees returned by direct RL methods seem equally distributed on both extremes of the scores.

On Figure (cite), we compare the best trees returned by Modified DQN and Modified PPO. We used Algorithm 6 to extract 20 trees from the 20 partially observable policies returned by deeop reinforcement learning over 20 seeds. We then plot the best tree for each baseline. Those trees get an everage reward of roughly 175. Similarly, we plot a representative tree for imitation learning baseline as well as a tree that has max reward obtained with Viper. However unlike for direct methods, the tree returned are extremely similar across seed. 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 tree across seed returned by modified RL baselines is the "infinite" tree that corresponds to only taking an information gathering action.

2.4 Discussion

We have shown that compared to learning non-interpretable policies for both the base MDP and some associated IBMDP, learning partially observable policies in IBMDP is difficult (c.f. Figures and (cite)). As a consequence, only a handful of Modified RL runs are able to learn decision tree policies that are on par with imitated trees (c.f. Figure).

In the next chapter, we highlkight the conncexions between RL in IBMDPs and

POMDPs to get insights on the hardness of direct reinforcement learning of decision trees.

Chapitre 3

Understanding the Limits of Direct Reinforcement Learning

From the previous Chapter (cite) we know that to directly learn decision trees for an MDP, one can learn a partially observable policy for an IBMDP (cite). Such problems are classical instances of Partially Observable Markov Decision Processes (POMDPs) (cite). This connexion with POMDP was not done by the authors of IBMDPs.

3.0.1 Partially Observable IBMDPs

Definition 12 (Partially Observable Markov Decision Processes)**.** *A Partially Observable Markov Decision Process (POMDP) is a tuple* $\langle X, A, O, T, T_0, \Omega, R\rangle$ *where :*

- X *is the state space (like in the definition of MDPs (cite)).*
- A *is a finite set of actions (like in the definition of MDPs(cite)).*
- O *is a set of observations.*
- $T : X \times A \to \Delta X$ *is the transition kernal, where* $T(boldsymbolx_t, a, \boldsymbol{x}_{t+1}) = P(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}, a)$ *is the probability of transitioning to state* $\boldsymbol{x}_t$ *when taking action* a *in state* $\boldsymbol{x}$
- T_0 *: is the intial distribution over states.*
- $\Omega : X \to \Delta O$ *is the observation kernel, 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 (cite).*

We can simply extend the definition of Iterative Bounding MDPs (cite) with an

observation kernel to get Partially Observable IBMDPs :

Definition 13 (Partially Observable Iterative Bounding Markov Decision Processes)**.** *a Partially Observable Iterative Bounding Markov Decision Process (POIBMDP) is an IBMDP (cite) extended with an observation kernel*

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

The transition kernel Ω *maps state features and observations to observations,* $\Omega : S \times O \to O$*, with* $P(\boldsymbol{o}|(\boldsymbol{s}, \boldsymbol{o})) = 1$

POIBMDPs are a particular instance of POMDPs with observations being some indices of the fully-observable states. This setting has other names in the litterature : Mixed Observability MDPs (cite), Block MDPs (cite) N-MDPs (cite). POIBMDPs can also be seen as non-stationary MDPs in which there is one different transition kernel per base MDP state : these are called Hidden-Mode MDPs (cite).

Following (cite) we can write the definition of the value of a deterministic partially observable policy $\pi : O \to A$ in observation o.

Definition 14 (Partial observable value function)**.** *In a POIBMDP, the expected cumulative discounted reward of a deterministic partially observable policy* $\pi : O \to A \cup A_{info}$ *starting from observation o is* $V^{\pi}(o)$ *:*

$$V^{\pi}(o) = \sum_{(s,o') \in S \times O} P^{\pi}((s,o')|o) V^{\pi}((s,o'))$$

with $P^{\pi}((s,o')|o)$ *the asymptotic occupancy distribution (see cite for definition) of the fully observable state* (s,o') *given the partial observation o and* $V^{\pi}((s,o'))$ *the classical state-value function defined in (cite).*

The asymptotic occupancy distribution is the probability of a policy π to be in (s,o') while observing o and having taken actions given by π.

The problem that we solve is to find the deterministic partially observable policy that maximizes the excpeted value in the initial observation :

Definition 15 (Revised Direct interpretable RL objective)**.**

$$\pi^{\star}_{po} = \underset{\pi_{po}}{\operatorname{argmax}} J(\pi_{po}) = \underset{\pi_{po}}{\operatorname{argmax}} V^{\pi}(\boldsymbol{o}_0) \tag{3.1}$$

With π_{po} a determintic partially observable policy $\pi_{po} : O \to A \cup A_{info}$. There is no expectation over possible initial observation in the above objective function as there is only one initial observation in a POIBMDP : $o_0 = (L_1, U_1, \ldots, L_n, U_n)$.

In general, the policy that maximizes the expected discounted cumulative reward in a POMDP maps "belief states" or observations histories (cite) to actions, i.e., they are not solutions to our problem (cite). If this would have been the case, any standard RL algorithm would find solutions to problem (cite) because both histories and belief states are sufficient statistic for the POMDP hidden states (cite).

The particular problem of finding the optimal deterministic partially observable policies for POMDPs is NP-HARD (cite) (Memoryless Littman section 3.2). It means there is not necessarily an algorithman that can find the optimal deterministic policy and if there were one, it would be slow.

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.

In (cite), authors show that the optimal partially observable policy can be stochastic (cite precise section), hence policy gradient algoriothms (cite) are to avoid since we want the best deterministic policy. Furthermore, the optimal deterministic patially observable policy might not maximize all the values of all observations simulataneously (cite precise section) which makes difficult the use of the Bellman optimality equation (cite) to compute policies.

Despite those hardness results, empirical results of applying RL to POMDPs by naively setting the states to be observations has shown promising results (cite). More recently, the framework of Baisero et. al. (cite) called asymmetric RL has also shown promising empirical and theoretical results when leveraging fully-observable state information during training of a partially observable policy. In this Chapter, we use reinforcement learning to train decision tree policies for MDPs by seeking deterministic partially observable policies for POIBMDPs (cite).

We will attempt to learn a depth-1 tree policy for the 2×2 grid world from Example (cite) as the base MDP by soving the example POIBMDP with similar parameters as the Example IBMDP from the previous Chapter (cite). We carefully craft POIBMDPs such that the *optimal* partially observable deterministic policy corresponds to a decision tree policy of detph 1. To do so, we present next how the POIBMDP objective values (cite) from different decision tree policies change with ζ for some fixed discount factor γ.

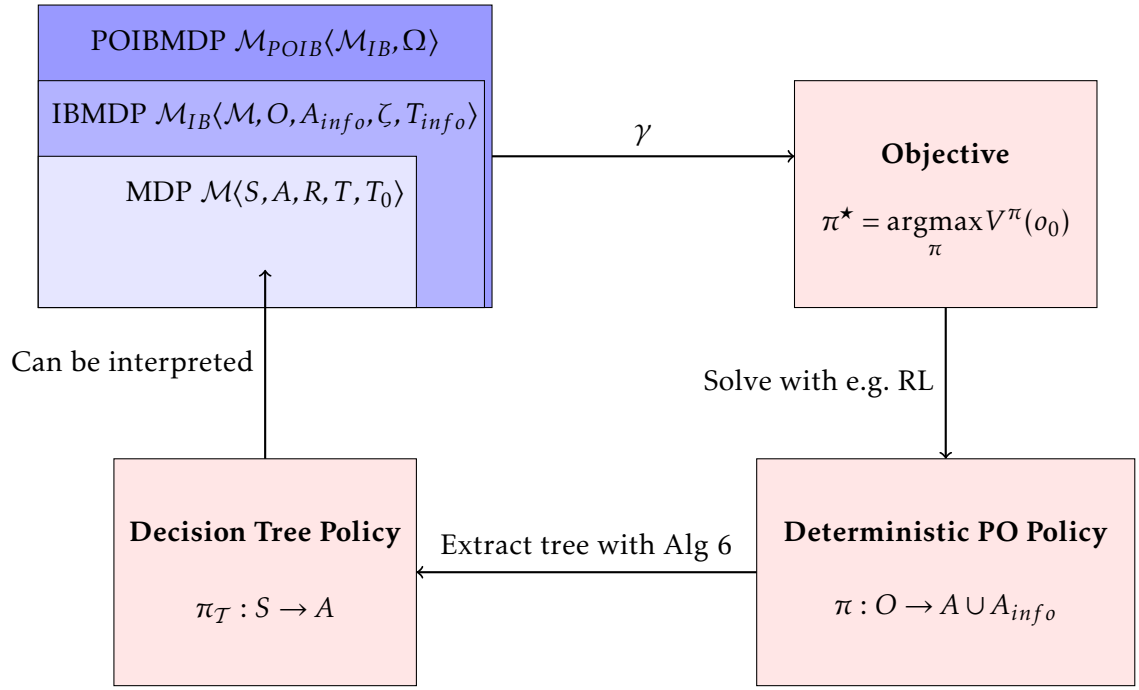


FIGURE 3.1 – A formal framework to learn decision tree policies for MDPs. This include learning a partially observable deterministic policy in a POIBMDP (cite).

3.1 Constructing POIBMDPs which optimal solutions are the depth-1 tree

Because we know all the base states, all the observations, all the actions, all the rewards and all the transitions of our POIBMDP, we can compute exactly the values of different partially observable deterministic policies given ζ the reward for IGAs and γ the discount factor.

Each of those policies can be one of the following trees illustrated in Figure (cite) :

— $\pi_{\mathcal{T}_0}$: 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
— $\pi_{\mathcal{T}_2}$: a depth-2 tree that alternates between taking two IGAs and a base action
— an inifinite "tree" that only takes IGAs

Furthermore, because from (cite) we know that for POMDPs, stochastic policies can sometimes get better expected discounted rewards than deterministic policies, we also

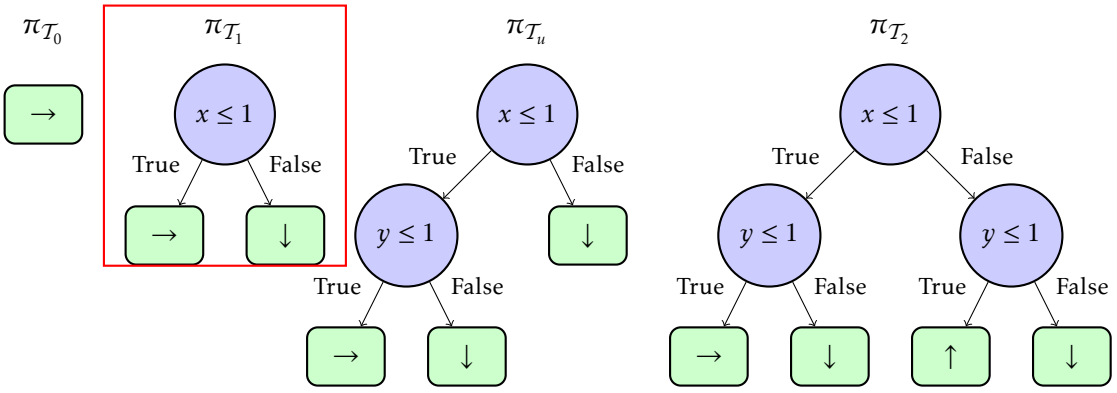


FIGURE 3.2 – For each decision tree structure, e.g., depth-1 or unbalanced depth-2, in the space of deterministic partially observable POIBMDP policies, we illustrate the decision tree which gets the highest base rewards.

compute the value of the stochastic policy that alternates between two base actions : → and ↓. Those two base actions always lead to the goal state in expectation.

We detail the calculations for the detpth-1 decision tree objective value (cite) and defer the calculations for the other policies to the Appendix (cite).

Proposition 2 (Depth-1 decision tree objective value)**.** *The objective value of the best depth-1 decision tree from Figure (cite) is* $V^{\pi_{\mathcal{T}_1}}(o_0) = \frac{4\zeta+\gamma+2\gamma^3+\gamma^5}{4(1-\gamma^2)}$.

Démonstration. $\pi_{\mathcal{T}_1}$ has one root node that tests $x \leq 1$ (respectively $y \leq 1$) and two leaf nodes → and ↓. To compute $V^{\pi}_{\mathcal{T}_1}(o_0)$, we compute the values of $\pi_{\mathcal{T}_1}$ in each of the possible startin states $(s_0, o_0), (s_1, o_0), (s_2, o_0), (s_g, o_0)$ and compute the expectation over those. At inititalization, when the base state is $s_g = (1.5, 0.5)$, 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 :

$$\begin{aligned} V^{\pi_{\mathcal{T}_1}}(s_g, o_0) &= \zeta + \gamma + \gamma^2\zeta + \gamma^3 \dots \\ &= \sum_{t=0}^{\infty} \gamma^{2t}\zeta + \sum_{t=0}^{\infty} \gamma^{2t+1} \\ &= \frac{\zeta + \gamma}{1-\gamma^2} \end{aligned}$$

At inititialization, in either of the base states $s_0 = (0.5, 0.5)$ and $s_2 = (1.5, 1.5)$, 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

s_g :

$$\begin{aligned} V^{\pi_{\mathcal{T}_1}}(s_0, o_0) &= \zeta + \gamma 0 + \gamma^2 V^{\pi_{\mathcal{T}_1}}(s_g, o_0) \\ &= \zeta + \gamma^2 V^{\pi_{\mathcal{T}_1}}(s_g, o_0) \\ &= V^{\pi_{\mathcal{T}_1}}(s_2, o_0) \end{aligned}$$

Similarly, the value of the best depth-1 decision tree policy in state $s_1 = (0.5, 1.5)$ is the value of taking one information gathering action then moving right to s_2 then following the policy in s_2 :

$$\begin{aligned} V^{\pi_{\mathcal{T}_1}}(s_1, o_0) &= \zeta + \gamma 0 + \gamma^2 V^{\pi_{\mathcal{T}_1}}(s_2, o_0) \\ &= \zeta + \gamma^2 V^{\pi_{\mathcal{T}_1}}(s_2, o_0) \\ &= \zeta + \gamma^2 (\zeta + \gamma^2 V^{\pi_{\mathcal{T}_1}}(s_g, o_0)) \\ &= \zeta + \gamma^2 \zeta + \gamma^4 V^{\pi_{\mathcal{T}_1}}(s_g, o_0) \end{aligned}$$

Since the probability of being in any base states at initialization given that the agent observe o_0 is the probability of being in any base states at initialization, we can write :

$$\begin{aligned} V^{\pi_{\mathcal{T}_1}}(o_0) &= \frac{1}{4} V^{\pi_{\mathcal{T}_1}}(s_g, o_0) + \frac{2}{4} V^{\pi_{\mathcal{T}_1}}(s_2, o_0) + \frac{1}{4} V^{\pi_{\mathcal{T}_1}}(s_1, o_0) \\ &= \frac{1}{4} \frac{\zeta + \gamma}{1 - \gamma^2} + \frac{2}{4} (\zeta + \gamma^2 \frac{\zeta + \gamma}{1 - \gamma^2}) + \frac{1}{4} (\zeta + \gamma^2 \zeta + \gamma^4 \frac{\zeta + \gamma}{1 - \gamma^2}) \\ &= \frac{1}{4} \frac{\zeta + \gamma}{1 - \gamma^2} + \frac{2}{4} (\frac{\zeta + \gamma^3}{1 - \gamma^2}) + \frac{1}{4} (\frac{\zeta + \gamma^5}{1 - \gamma^2}) \\ &= \frac{4\zeta + \gamma + 2\gamma^3 + \gamma^5}{4(1 - \gamma^2)} \end{aligned}$$

□

We can now plot the POIBMDP objective values of the different policies corresponding to trees for the grid world MDP 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 that a depth-1 tree is the optimal deterministic partially observable POIBMDP policy for $0 < \zeta < 1$.

Interestingly, two POMDP challenges described in (cite) can also be observed in Figure (cite). First, there is a whole range of ζ values for which the stochastic policy is optimal. Second, for e.g. $\zeta = 0.5$, while the depth-1 tree is the optimal deterministic partially observable policy, the value of state $(s_2, o_0) = (1.5, 1.5, 0, 2, 0, 2)$ is not maximized

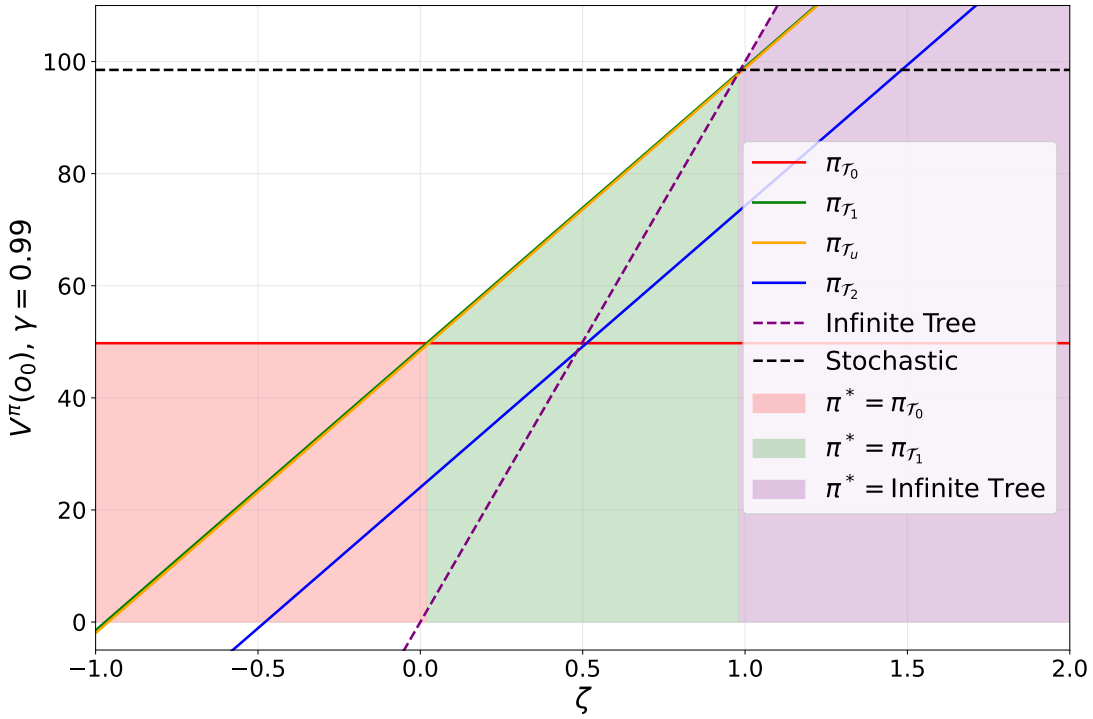


FIGURE 3.3 – POIBMDP objective values of different policies as functions of ζ. Shaded areas show the optimal policies in different ranges of ζ values.

by this policy but by the sub-optimal policy that always goes down.

Now that we know to what values to set ζ in our POIBMDPs, we can apply reinforcement learning algorithms to learn partially observable deterministic policies and see if we can retrieve the optimal depth-1 decision tree.

3.1.1 Asymmetric Reinforcement Learning

3.2 Results

Unfortunately, our results are negative and show that reinforcement learning fails for the aforementioned problem. Let us understand why.

3.2.1 Experimental Setup

Baselines : we consider two groups of RL algorithms. The first group is standard tabular RL naively applied to POIBMDPs ; Q-learning (cite)(cite), Sarsa (cite)(cite), and Policy Gradient (cite)(cite). (cite) and (cite) already studied the theoretical and practical implications of applying standard RL directly to POMDP by considering that partial observations are full MDP states. (cite) proved that Q-Learning will converge but without optimality guarantees. (cite) showed empirically that Sarsa-λ (cite), a version of Sarsa with some sort of memory, can learn good deterministic solutions to POMDP which makes it a good candiate for our problem.

We also use a vanilla tabular Policy Gradient baseline (cite) with softmax policy, which to the best of our knowledge, nobody studied in our setting. 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 seeked depth-1 decision tree policy (c.f. Figure (cite)).

More recently, RL algorithms were developped specially for learning policies in POMDPs. Those algorithms are called asymmetric RL which authors of the original IBMDP paper (cite) use without knowing. Asymmetric algorithms leverage the fact that, even if the deployed policy should depend only on partial observation, nothing forbids the use of full state information during training if the latter is available. Asymmetric Q-learning (cite) makes use of full-state-action value function $U : S \times O \to \mathbb{R}$ to use as a temporal difference error target (cite) when updating the $Q : O \times A \to \mathbb{R}$ of intereset. In Algorithm (cite), we write the Asymmetric Q-learning algorithm.

In addition to the traditional tabular RL algorithms, we also apply Asymmetric Q-learning and Asymmetric Sarsa and JSJ, the tabular algorithm from (cite)(cite) which

is equivalent to a tabular Asymmetric Policy Gradient algorithm.

We use at least 200 000 time steps to train each agent. Each agent is trained on 100 seeds on each POIBMDP.

Algorithme 12 : Asymmetric Q-Learning (cite)

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 $o_0 \sim \Omega(x_0)$
  for *each step* t **do**
    Choose action a_t using ϵ-greedy : $a_t = \arg\max_a Q(o_t, a)$ with prob. $1 - \epsilon$
    Take action a_t, observe $r_t = R(x_t, a_t)$, $x_{t+1} \sim T(x_t, a_t)$, and $o_{t+1} \sim \Omega(x_{t+1})$
    $y \leftarrow r + \gamma U(x_{t+1, a'} Q(o_{t+1}, a'))$ // `TD target`
    $U(x_t, a_t) \leftarrow (1 - \alpha_u) U(x_t, a_t) + \alpha_u y$
    $Q(o_t, a_t) \leftarrow (1 - \alpha_q) Q(o_t, a_t) + \alpha_q y$
    $x_t \leftarrow x_{t+1}$
    $o_t \leftarrow o_{t+1}$
  end
end
$\pi(o) = \arg\max_a Q(o, a)$ // `Extract greedy policy`

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 of the learned policy POIBMDP value with respect to the optimal deterministic partially observable POIBMP policy : $|V^{\pi\star}(o_0) - V^{\pi}(o_0)|$ Because we know the whole POIBMDP model that we can represent exactly as tables ; and because we know for each ζ value the POIBMDP objective value of the optimal partially observable policy (c.f. Figure) ; we can report the *exact* sub-optimality gaps.

Second, we consider the distribution of the learned trees structure over the 100 training seeds. Indeed, since for every POIBMDP we train each baseline 100 times, we obtain 100 partially observable deterministic policies from which we can extract the equivalent 100 decision tree policies using Algorithm (cite). This helps understand which trees RL tend to learn.

Algorithme 13 : JSJ algorithm. Uses Monte Carlo estimates of the average reawrd value functions to perform policy imporvements (cite)

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 $\pi_\theta : O \to \Delta A$
Initialize policy parameters θ
Initialize $Q(o,a) = 0$ for all observations o and actions a
for *each episode* **do**
  for $i = 1$ *to* N **do**
    Generate trajectory $\tau_i = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T)$ 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 (o_t, a_t, G_t) for later averaging
    end
  end
  for *each unique observation-action pair* (o,a) **do**
    $Q(o,a) \leftarrow \frac{1}{|\{(o,a)\}|} \sum_{(o,a,G)} G$ // Monte Carlo estimate
  end
  for *each observation o* **do**
    for *each action a* **do**
      $\pi_1(a|o) \leftarrow 1.0$ if $a =_{a'} Q(o,a')$, 0.0 otherwise // Deterministic policy from Q-values
      $\pi(a|o) \leftarrow (1-\alpha)\pi(a|o) + \alpha \pi_1(a|o)$ // Policy improvement step
    end
  end
  Reset $Q(o,a) = 0$ for all observations o and actions a // Reset for next episode
end

3.2.2 Can RL baselines find the optimal deterministic partially observable POIBMDP policies?

In Figure (cite), 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 ζ uniformily in $[-1, 2]$. For (asymmetric) Q-learning and (asymmetric) Sarsa, we extract the greedy policies from the learned Q-functions and evaluate them every 2000 steps. For the Policy Gradient and the JSJ baselines we directly evaluate the learned stochastic policies.

In Figure (cite), we plot the distributions over the final learned trees in function of ζ from the above runs. For the Policy Gradient and JSJ baselines, we extract the tree from the policy that is greedy w.r.t to the action probabilities as ALgorithm 6 requires a deterministic partially observable policy to return a tree policy.

We observe that, despite all runs converging, independently of the ζ values, not all runs fully minimize the sub-optimality gap. In particular all RL algorithms seem to consistantly minimze the gap, i.e. learn the optimal policy or Q-function, for $\zeta \in [-1, 0]$, where the optimal policy is the detph-0 tree. For $\zeta \in [1, 2]$, where the optimal policy is to repeat taking information gathering actions, only the Q-learning baseline learns the optimal policy. In $\zeta \in]0, 1[$, where the depth-1 tree is optimal, no baseline can consistently learn the optimal solution. However, we observe that asymmetric versions of Q-learning and Sarsa have found the optimal policy more frequently than other baselines. 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.

Next, we quantify how difficult it is to do RL to learn policies in POIBMDPs.

3.2.3 How difficult is it to learn in POIBMPDs?

To show that learning the optimal deterministic partially observable POIBMDP policy is very difficult, even though there are only a handful of states, observations, and actions (16, 9, and 6 respectively), we compare the success rate of RL baselines when applied to a POIBMDP with $\gamma = 0.99$, $\zeta = 0.5$ and when applied to the corresponding fully observable IBMDP, and MDP.

Because we know that for those values of ζ and γ, the optimal deterministic partially observable policy is a depth-1 tree and the optimal fully observable policy is a tabular policy from Figure (cite); we can compute the success rate of an algorithm as the proportion of learned policies that match their optimal counterparts.

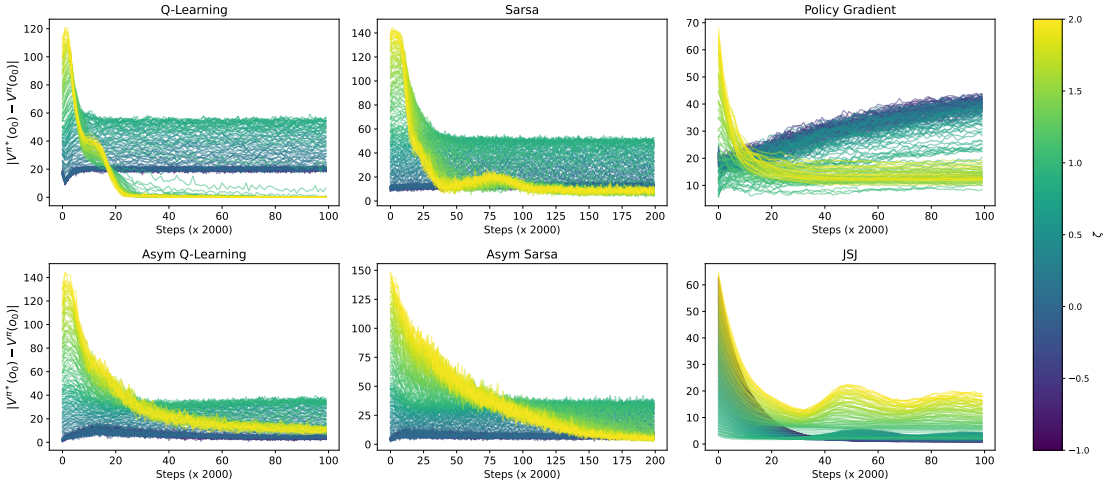


FIGURE 3.4 – Baselines learning curves of partially observable policies in POIBMDPs. In each subplot, each single learning curve is colored by the value of ζ in the corresponding POIBMDP in which learning occurs. Each single learning curve represent the sub-optimality gap averaged over 100 seeds. So for each baseline we ran a total of 200×100 single training run.

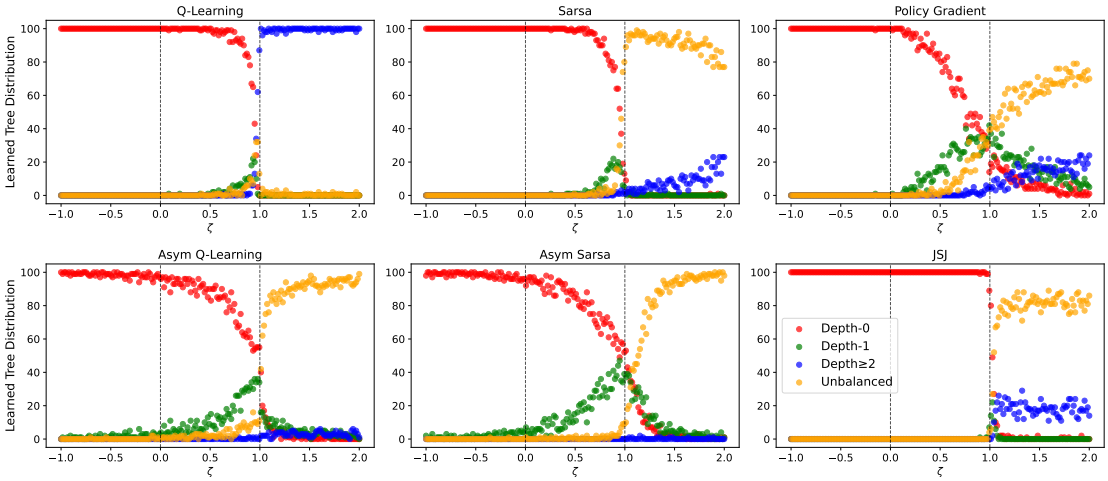


FIGURE 3.5 – Distributions of final tree policies learned across the 100 seeds. For each ζ value, there are four color points each represeting the share of depth-0 trees (red), depth-1 trees (green), unbalanced depth-2 trees (orange) and depth-2 trees (blue). Note that those tres do not necessary correspond to tres in Figure (cite). We simply check the structure of trees and not if they are the best-in-class tree.

Furhtermore ; we also consider advanced tricks for tabular RL in order to properly assess the difficulty of learning in POIBMDPs compared to learning in (IB)MDPs :

1. Optimistic Q-function (cite)
2. Entropy regularization (cite)
3. Eligibility traces (cite)
4. ϵ-decay (cite)

In particular, for each of the aforementioned RL baselines ; we learn policies using a wide variety of hyperparamters. For example, in Table (cite) we provide the detailed hyperparameter space of Asymmetric Sarsa, which induce a total of 1152 instances of Asymmetric Sarsa, and in Table (cite) we provide the hyperparameter space sizes for all baselines. We run each baseline using each hyperparamters combination 10 times on 200000 timesteps.

Tableau 3.1 – Asymmetric Sarsa Hyperparameter Space (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	Q learning rate
Learning Rate Q	0.001, 0.005, 0.01, 0.1	U learning rate
Optimistic	True, False	Optimistic initialization

Tableau 3.2 – Summary of RL baselines Hyperparameters

Algorithm	**Problem**	**Total Hyperparameter Combinations**
Policy Gradient	PO/IBMDP	420
JSJ	POIBMDP	15
Q-learning	PO/IBMDP	192
Asym Q-learning	POIBMDP	768
Sarsa	PO/IBMDP	192
Asym Sarsa	POIBMDP	768

The key observations from Figure (cite) is that POIBMDPs are way harder than their IBMDPs counterparts. Even though asymmetry seems to increase performances ; learning a decision tree policy for a simple grid world directly with RL using the

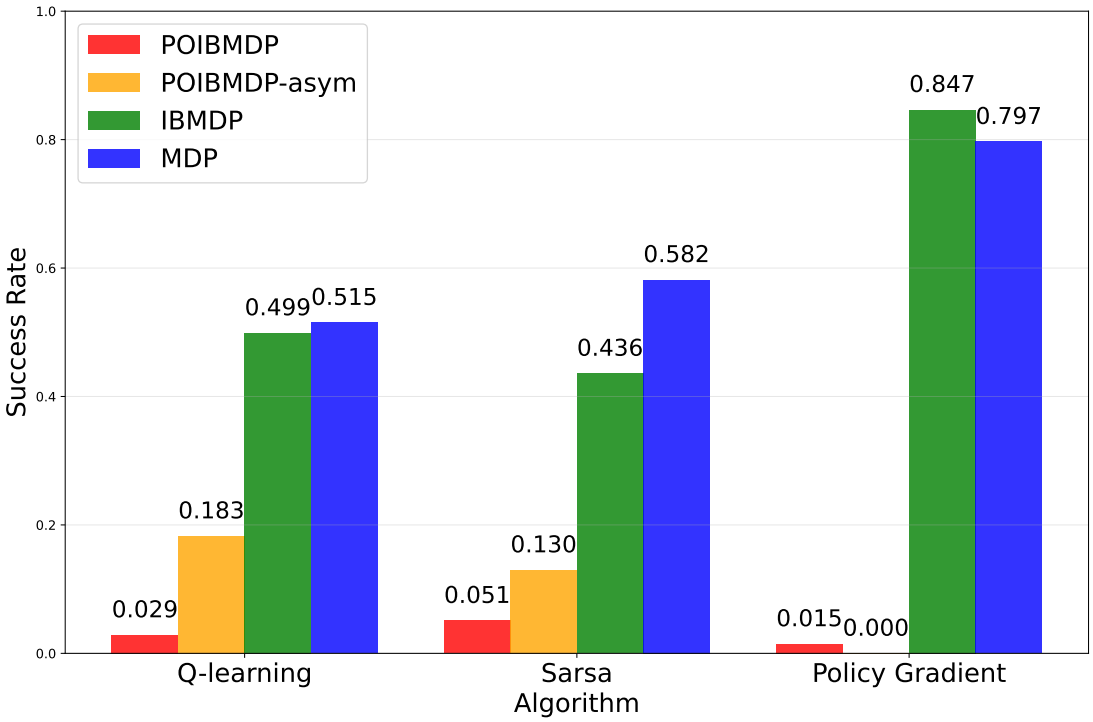
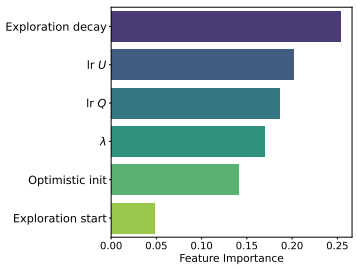
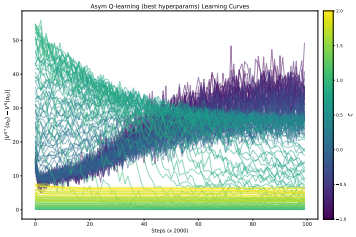


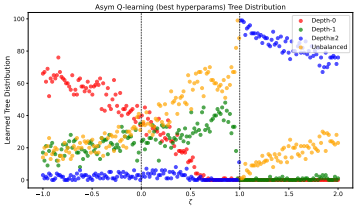
FIGURE 3.6 – Success rates of different RL algorithms over thousands of runs when applied to a POIBMDP and its fully observable IBMDP and MDP counterparts



(a) Hyperparameter importance for Asymmetric Q-learning



(b) Best learning curves for Asymmetric Q-learning



(c) Best tree distributions for Asymmetric Q-learning

FIGURE 3.7 – Asymmetric Q-learning performance analysis : hyperparameter importance, learning curves, and tree distributions across different POIBMDP configurations.

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

Tableau 3.3 – Best hyperparameters for each algorithm on the POIBMDP problem

framework of POIBMDP seem way to 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 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 ? Even after choosing candidate IGAs ; RL with large action space is known to be an already difficult problem.

It is important to note that some hyperparameter sets gave a 10/10 success rate for asymmetric baselines (c.f. Table (cite)). However, when we try those top performing hyperparameters on more seeds, as shown in Figure (cite), even though we observe an increase in the proportion of depth-1 trees learned, the performance are still not satisfactory for such a simple task.

3.3 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. We once again showed the difficulty of the latter by crafting a POIBMDP for which we know exactly the optimal deterministic partially observable policy. This allowed to show 1) that existing (asymmetric) RL algorithms fail to consistently learn the optimal deterministic partially observable policy (c.f. Figure (cite)) and 2) to show that for similar transitions and rewards, learning a partially observable deterministic policy is way harder than learning a fully observable one (c.f. Figure (cite)).

So interesting future work could be to design POIBMDP-specific algorithm or to

find another framework for the direct learning of decision trees for MDPs. Overall, in the two last Chapter, it seems fair to say that indirect interpretability is more suited for MDPs Despite those negative conclusion, there is still some positive directions coming out of this work. In the next Chapter, we show that when the base MDP's transitions are independent of the action, then POIBMDPs are actually *fully* observable which can lead to new decision tree induction for supervised learning.

Chapitre 4

When transitions are uniform POIBMDPs are fully observable

In this section we show that decision tree induction for classification problems can be formulated as solving POIBMDPs. Indeed, a supervised classification problem (cite) can be formulated as an MDP where actions are class labels and states are training data. The reward at every step is 1 if the correct label was predicted and 0 otherwise. In this MDP, the transitions are independent of the actions : the next state is given by uniformly sampling a new training datum. This implies that the resulting POIBMDP is actually *fully* observable and that traditional RL algorithms should perform well.

Definition 16 (Classification Markov Decision Process)**.** *Given a set of N examples denoted* $\mathcal{E} = \{(x_i, y_i)\}_{i=1}^{N}$ *where each datum* x_i *is described by a set of p features and* $y_i \in \mathbb{Z}^m$ *is the label associated with* x_i*, a Classification Markov Decision Process is an MDP (cite)* $\langle S, A, R, T, T_0 \rangle$ *where :*

- *the state space is* $S = \{x_i\}_{i=1}^{N}$*, the set of 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* $R(s = x_i, a) = 1_{a=y_i}$
- *the transition function is* $T : S \times A \to \Delta S$ *with* $T(s, a, s') = \frac{1}{N} \quad \forall s, a, s'$
- *the initial distribution is* $T_0(s_0 = s) = \frac{1}{N}$

It is easy to see that policies that maximize the expected discounted cumulative reward objective (cite) are classifiers that maximize the prediction accuracy because $\sum_{i=1}^{N} 1_{\pi(x_i)=y_i} = \sum_{i=1}^{N} R(x_i, \pi(x_i))$. We defer the formal proof in the next part of the manuscript in which we extensively study supervised learning problems. Learning a classifier with a reward signal can also be done with contextual bandits (cite).

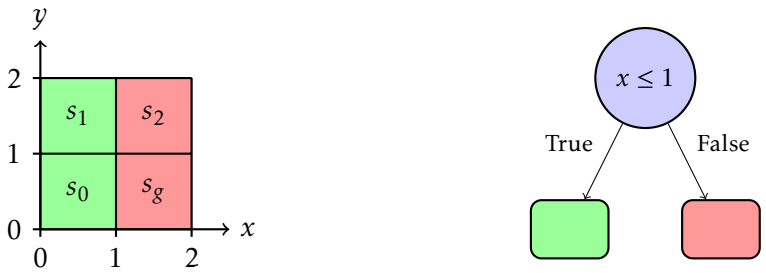


FIGURE 4.1 – Classification MDP optimal actions. In this Classification MDP, there are four data to assign either a green or red label. On the right, there is the unique optimal depth-1 tree for this particular Classification MDP. This depth-1 tree also maximizes the accuracy on the corresponding classification tasks.

To learn a decision tree *classifier*, we use the POIBMDP formalism and show, in the particular case where the base MDP is a Classification MDP (cite), that associated POIBMDPs are in fact MDPs with stochastic transitions.

Definition 17 (Classification POIBMDP)**.** *Given a Classification MDP (cite)* $\langle \{x_i\}_{i=1}^N, \mathbb{Z}^m, R, T, T_0 \rangle$*, and an associated POIBMDP (cite)* $\langle S, O, A, A_{info}, R, \zeta, T_{info}, T, T_0 \rangle$*, a Classification POIBMDP is an MDP :*

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

- O *is the set of possible observations in* $[L_1, U_1] \times \cdots \times [L_d, U_d] \times [L_1, U_1]$ *times* $\cdots \times [L_d, U_d]$ *where* L_j *is the minimum value of feature* j *over all data* 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 refinement in* A_{info}
- *The reward for assigning label* $a \in \mathbb{Z}^m$ *when observing some observation* $o = (L'_1, U'_1, \ldots, L'_d, U'_d)$ *is the proportion of training data satistifying the bounds and having label* a *:* $R(o, a) = \frac{|\{x_i : L'_j \leq x_{ij} \leq U'_j \forall i,j\} \cap \{x_i : y_i = a \forall i\}|}{|\{x_i : L'_j \leq x_{ij} \leq U'_j \forall i,j\}|}$*. The reward for taking an information gathering action that refines bounds is* ζ
- *The transition kernel is* $\mathcal{P} : O \times (\mathbb{Z}^m \cup A_{info}) \to \Delta O$ *where :*
  - *For* $a \in \mathbb{Z}^m$ *:* $\mathcal{P}(o, a, (L_1, U_1, \ldots, L_d, U_d)) = 1$ *(reset to full bounds)*
  - *For* $a = (k, v) \in A_{info}$ *: from* $o = (L'_1, U'_1, \ldots, L'_d, U'_d)$*, the MDP will transit to* $o_{left} = (L'_1, U'_1, \ldots, L_k, v, dots, L'_d, U'_d)$ *(resp.* $o_{right} = (L'_1, U'_1, \ldots, U'_k, v, dots, L'_d, U'_d)$*) with*

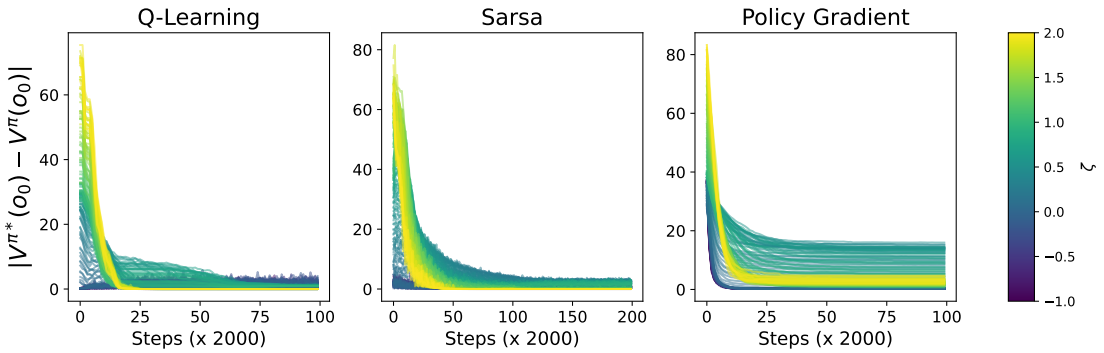


FIGURE 4.2

$$probability\ \frac{|\{x_i : L'_j \leq x_{ij} \leq U'_j \forall j \wedge x_{ik} \leq v\}|}{|\{x_i : L'_j \leq x_{ij} \leq U'_j \forall j\}|}\ (resp.\ \frac{|\{x_i : L'_j \leq x_{ij} \leq U'_j \forall j \wedge x_{ik} > v\}|}{|\{x_i : L'_j \leq x_{ij} \leq U'_j \forall j\}|})$$

Now we can learn a decision tree *classifier* for a supervised learning task as follows. Given a set of examples and lables, we can formulate the associated Classification MDP (cite). Then we can provide some information gathering actions and a reward ζ for those to define a Classification POIBMDP (cite). Finally, we can learn a deterministic policy $\pi : O \to A \cup A_{info}$ to obtain a decision tree classifier. Since we have shown that Classification POIBMDPs are MDPs, we that RL baselines should perform well to solve Classification POIBMDPs. We expect that, compared to general POIBMDPs studied in the previous chapter where some information was hidden to the agent, RL baselines will consistently learn the optimal decision tree classifiers.

4.1 How well can RL baselines learn in Classification POIBMDPs ?

Similarly to the previous chapter, we are interested in a very simple Classification POIBMDP that corresponds to building a tree for the supervised problem :

$$\mathcal{X} = \{(0.5, 0.5), (0.5, 1.5), (1.5, 1.5), (1.5, 0.5)\}$$
$$y = \{0, 0, 1, 1\}$$

We illustrate the associated Classification MDP in Figure (cite). We construct Classification POIBMDPs with $\gamma = 0.99$, 200 values of $\zeta \in [0, 1]$ and IGAs $x \leq 1$ and $y \leq 1$. Since Classification POIBMDPs are MDPs, we do not need to analyze asymmetric and JSJ (cite) baselines.

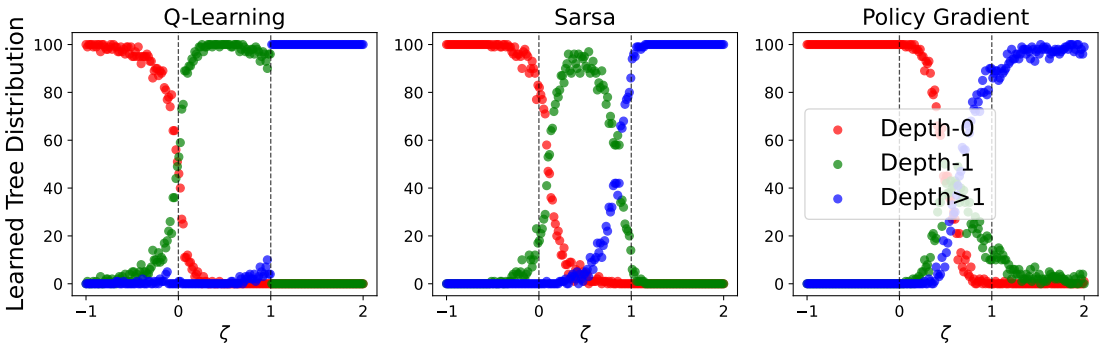


FIGURE 4.3

Fortunately this time, compared to general POIBMDPs, RL can be used to retrieve optimal policies in Classification POIBMDPs equivalent to decision tree classifiers. We observe on Figure (cite) that both Q-learning and Sarsa consistently minimize the sub-optimality gap. Hence they are able to retrieve the optimal detph-1 decision tree classifier from Figure (cite).

In this part of the manuscript, we have highlighted the challenges of reinforcement learning of decision tree policies for MDPs. We showed that this task was particularly challenging, even when an optimal decision tree policy exists. We saw that RL algorithms are unable to find such optimal trees, even for very simple problems such as a 2×2 grid world. However, we also showed that for some MDPs, such as ones that encode a supervised learning task, RL algorithms could retrieve optimal decision tree classifiers. This was shown only on a very simple classification task with four data and two classes. Furthermore, good candidate nodes–information gathering actions–were provided to the learning algorithms.

Now remains the question of scalability and competitiveness of such approaches to supervised learning for which strong baselines have existed for decades. In the next part of this thesis ; we present a new decision tree induction that relies on solving MDPs exactly and choosing information gathering actions adaptively.

A key future work is to prove that any direct reinforcement learning of decision tree policies algorithm is equivalent to RL in POIBMDPs hence generalizing and formally proving that direct reinforcement learning of interpretable policies is hard. We explore in the next part of this manuscript another natural continuation of this part which is to solve classification MDPs for decision tre induction in the supervised learning setting.

Deuxième partie

An easier problem : Learning Decision Trees for MDPs that are Classification tasks

Chapitre 5

DPDT-intro

In supervised learning, decision trees are valued for their interpretability and performance. While greedy decision tree algorithms like CART remain widely used due to their computational efficiency, they often produce sub-optimal solutions with respect to a regularized training loss. Conversely, optimal decision tree methods can find better solutions but are computationally intensive and typically limited to shallow trees or binary features. We present Dynamic Programming Decision Trees (DPDT), a framework that bridges the gap between greedy and optimal approaches. DPDT relies on a Markov Decision Process formulation combined with heuristic split generation to construct near-optimal decision trees with significantly reduced computational complexity. Our approach dynamically limits the set of admissible splits at each node while directly optimizing the tree regularized training loss. Theoretical analysis demonstrates that DPDT can minimize regularized training losses at least as well as CART. Our empirical study shows on multiple datasets that DPDT achieves near-optimal loss with orders of magnitude fewer operations than existing optimal solvers. More importantly, ex-

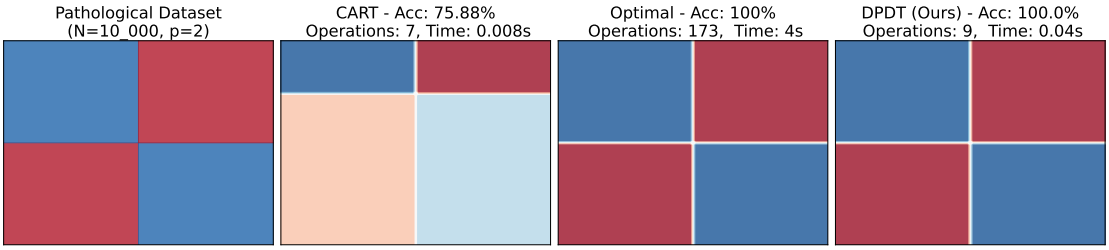


FIGURE 5.1 – Pathological dataset and learned depth-2 trees with their scores, complexities, runtimes, and decision boundaries.

tensive benchmarking suggests statistically significant improvements of DPDT over both CART and optimal decision trees in terms of generalization to unseen data. We demonstrate DPDT practicality through applications to boosting, where it consistently outperforms baselines. Our framework provides a promising direction for developing efficient, near-optimal decision tree algorithms that scale to practical applications.

5.1 Introduction

Decision trees [83, 82, 13] are at the core of various machine learning applications. Ensembles of decision trees such as tree boosting [39, 38, 20, 79] are the state-of-the-art for supervised learning on tabular data [44]. Human users can make sense of decision trees predictions [31, 62, 15] which allows for real-world applications when safety or trust is critical [89]. More recently, decision trees have been used to model sequential decision policies with imitation learning [**kohler2024interpretable**, 6] or directly with reinforcement learning (RL) [**marton2024sympolsymbolictreebasedonpolicy**, 97, 107, 106].

To motivate the design of new decision tree induction algorithms, Figure 5.1 exhibits a dataset for which existing greedy algorithms are suboptimal, and optimal algorithms are computationally expensive. The dataset is made up of $N = 10^4$ samples in $p = 2$ dimensions that can be perfectly labeled with a decision tree of depth 2. When running CART [13], greedily choosing the root node yields a suboptimal tree. This is because greedy algorithms compute locally optimal splits in terms of information gain. In our example, the greedy splits always give two children datasets which themselves need depth 2 trees to be perfectly split. On the other hand, to find the root node, an optimal algorithm such as [68] iterates over all possible splits, that is, $N \times p = 20,000$ operations to find one node of the solution tree.

In this work, we present a framework for designing non-greedy decision tree induction algorithms that optimize a regularized training loss nearly as well as optimal methods. This is achieved with orders of magnitude less operations, and hence dramatic computation savings. We call this framework "Dynamic Programming Decision Trees" (DPDT). For every node, DPDT heuristically and dynamically limits the set of admissible splits to a few good candidates. Then, DPDT optimizes the regularized training loss with some depth constraints. Theoretically, we show that DPDT minimizes the empirical risk at least as well as CART. Empirically, we show that on all tested datasets, DPDT can reach 99% of the optimal regularized train accuracy while using thousands times less operations than current optimal solvers. More importantly, we follow [44] methodology

to benchmark DPDT against both CART and optimal trees on hard datasets. Following the same methodology, we compare boosted DPDT [37] to boosted CART and to some deep learning methods and show clear superiority of DPDT.

5.2 Related Work

To learn decision trees, greedy approaches like CART [13] iteratively partition the training dataset by taking splits optimizing a local objective such as the Gini impurity or the entropy. This makes CART suboptimal with respect to training losses [74]. But CART remains the default decision tree algorithm in many machine learning libraries such as [77, 20, 52, 113] because it can scale to very deep trees and is very fast. To avoid overfitting, greedy trees are learned with a maximal depth or pruned a posteriori [13, chapter 3]. In recent years, more complex optimal decision tree induction algorithms have shown consistent gains over CART in terms of generalization capabilities [9, 104, 28].

Optimal decision tree approaches optimize a regularized training loss while using a minimal number of splits [9, 3, 105, 68, 28, 29, 60, 19]. However, direct optimization is not a convenient approach, as finding the optimal tree is known to be NP-Hard [50]. Despite the large number of algorithmic tricks to make optimal decision tree solvers efficient [28, 68], their complexity scales with the number of samples and the maximum depth constraint. Furthermore, optimal decision tree induction algorithms are usually constrained to binary-features dataset while CART can deal with any type of feature. When optimal decision tree algorithms deal with continuous features, they can usually learn only shallow trees, e.g. Quant-BnB [68] can only compute optimal trees up to depth 3. `PySTreeD`, the latest optimal decision tree library [60], can compute decision trees with depths larger than three but uses heuristics to binarize a dataset with continuous features during a pre-processing step. Despite their limitations to binary features and their huge computational complexities, encouraging practical results for optimal trees have been obtained [73, 59, 22, 61]. Among others, they show that optimal methods under the same depth constraint (up to depth four) find trees with 1–2% greater test accuracy than greedy methods.

In this work, we only consider the induction of nonparametric binary depth-constrained axis-aligned trees. By nonparametric trees, we mean that we only consider tree induction algorithms that optimize both features and threshold values in internal nodes of the tree. This is different from the line of work on Tree Alternating Optimization (TAO) algorithm [17, 112, 18] that only optimizes tree nodes threshold values for fixed nodes

features similarly to optimizing neural network weights with gradient-based methods.

There exist many other areas of decision tree research [63] such as inducing non-axis parallel decision trees [75, 51], splitting criteria of greedy trees [61], different optimization of parametric trees [76, 79], or pruning methods [34, 72].

Our work is not the first to formulate the decision tree induction problem as solving a Markov decision process (MDP) [32, 40, 97, 19]. Those works formulate tree induction as solving a partially observable MDP and use approximate algorithms such as Q-learning [40] or deep Q-learning [97] to solve them in an online fashion one datum from the dataset at a time. In a nutshell, our work, DPDT that we present next, is different in that it builds a stochastic and fully observable MDP that can be explicitly solved with dynamic programming. This makes it possible to solve exactly the decision tree induction problem.

Chapitre 6

DPDT-paper

6.1 Decision Trees for Supervised Learning

Let us briefly introduce some notations for the supervised classification problem considered in this paper. We assume that we have access to a set of N examples denoted $\mathcal{E} = \{(x_i, y_i)\}_{i=1}^{N}$. Each datum x_i is described by a set of p features. $y_i \in \mathcal{Y}$ is the label associated with x_i.

A decision tree is made of two types of nodes : split nodes that are traversed, and leaf nodes that finally assign a label. To predict the label of a datum x, a decision tree T sequentially applies a series of splits before assigning it a label $T(x) \in \mathcal{Y}$. In this paper, we focus on binary decision trees with axis-aligned splits as in [13], where each split compares the value of one feature with a threshold.

Our goal is to learn a tree that generalizes well to unseen data. To avoid overfitting, we constrain the maximum depth D of the tree, where D is the maximum number of splits that can be applied to classify a data. We let $\mathcal{T}_D$ be the set of all binary decision trees of depth $\leq D$. Given a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$, we induce trees with a regularized training loss defined by :

$$
\begin{aligned}
T^* &= \underset{T \in \mathcal{T}_D}{\operatorname{argmin}} \; \mathcal{L}_\alpha(T), \\
T^* &= \underset{T \in \mathcal{T}_D}{\operatorname{argmin}} \; \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, T(x_i)) + \alpha C(T),
\end{aligned}
\tag{6.1}
$$

where $C : \mathcal{T} \to \mathbb{R}$ is a complexity penalty that helps prevent or reduce overfitting

such as the number of nodes [13, 68], or the expected number of splits to label a data[73]. The complexity penalty is weighted by $\alpha \in [0,1]$. For supervised classification problems, we use the 0–1 loss : $\ell(y_i, T(x_i)) = 1_{\{y_i \neq T(x_i)\}}$. Please note while we focus on supervised classification problems in this paper, our framework extends naturally to regression problems.

We now formulate the decision tree induction problem 6.1 as finding the optimal policy in an MDP.

6.2 Decision Tree Induction as an MDP

Given a set of examples $\mathcal{E}$, the induction of a decision tree is made of a sequence of decisions : at each node, we must decide whether it is better to split (a subset of) $\mathcal{E}$, or to create a leaf node.

This sequential decision-making process corresponds to a Markov Decision Problem (MDP) [80] $\mathcal{M} = \langle S, A, R_\alpha, P, D \rangle$. A state is a pair made of a subset of examples $X \subseteq \mathcal{E}$ and a depth d. Then, the set of states is $S = \{(X,d) \in P(\mathcal{E}) \times \{0,\dots,D\}\}$ where $P(\mathcal{E})$ denotes the power set of $\mathcal{E}$. $d \in \{0,\dots,D\}$ is the current depth in the tree. An action A consists in creating either a split node, or a leaf node (label assignment). We denote the set of candidate split nodes $\mathcal{F}$. A split node in $\mathcal{F}$ is a pair made of one feature i and a threshold value $x_{ij} \in \mathcal{E}$. So, we can write $A = \mathcal{F} \cup \{1,\dots,K\}$. From state $s = (X,d)$ and a splitting action $a \in \mathcal{F}$, the transition function P moves to the next state $s_l = (X_l, d+1)$ with probability $p_l = \frac{|X_l|}{|X|}$ where $X_l = \{(x_i, y_i) \in X : x_i \leq x_{ij}\}$, or to state $s_r = (X \setminus X_l, d+1)$ with probability $1 - p_l$. For a class assignment action $a \in \{1,\dots,K\}$, the chain reaches an absorbing terminal state with probability 1. The reward function $R_\alpha : S \times A \to \mathbb{R}$ returns $-\alpha$ for splitting actions and the proportion of misclassified examples of X $-\frac{1}{|X|}\sum_{(x_i,y_i)\in X} \ell(y_i, a)$ for class assignment actions. $\alpha \in [0,1]$ controls the accuracy-complexity trade-off defined in the regularized training objective 6.1. The horizon D limits tree depth to D by forbidding class assignments after D MDP transitions.

The solution to this MDP is a deterministic policy $\pi : S \to A$ that maximizes $J_\alpha(\pi) = \mathbb{E}\left[\sum_{t=0}^{D} R_\alpha(s_t, \pi(s_t))\right]$, the expected sum of rewards where the expectation is taken over transitions $s_{t+1} \sim P(s_t, \pi(s_t))$ starting from initial state $s_0 = (\mathcal{E}, 0)$. Any such policy can be converted into a binary decision tree through a recursive extraction function $E(\pi, s)$ that returns, either a leaf node with class $\pi(s)$ if $\pi(s)$ is a class assignment, or a tree with root node containing split $\pi(s)$ and left/right sub-trees $E(\pi, s_l)/E(\pi, s_r)$ if $\pi(s)$ is a split. The final decision tree T is obtained by calling $E(\pi, s_0)$ on the initial state s_0.

Proposition 3 (Objective Equivalence)**.** *Let π be a deterministic policy of the MDP and π^* be an optimal deterministic policy. Then $J_\alpha(\pi) = -\mathcal{L}_\alpha(E(\pi, s_0))$ and $T^* = E(\pi^*, s_0)$ where T^* is a tree that optimizes Eq. 6.1.*

This proposition is key as it states that the return of any policy of the MDP defined above is equal to the regularized training accuracy of the tree extracted from this policy. A consequence of this proposition is that when all possible splits are considered, the optimal policy will generate the optimal tree in the sense defined by Eq. (6.1). The proof is given in the Appendix 6.6.

6.3 Algorithm

We now present the Dynamic Programming Decision Tree (DPDT) induction algorithm. The algorithm consists of two essential steps. The first and most computationally expensive step constructs the MDP presented in Section 6.2. The second step solves it to obtain a policy that maximizes Eq.6.2 and that is equivalent to a decision tree. Both steps are now detailed.

6.3.1 Constructing the MDP

An algorithm constructing the MDP of section 6.2 essentially computes the set of all possible decision trees of maximum depth D which decision nodes are in $\mathcal{F}$. The transition function of this specific MDP is a directed acyclic graph. Each node of this graph corresponds to a state for which one computes the transition and reward functions. Considering all possible splits in $\mathcal{F}$ does not scale. We thus introduce a state-dependent action space A_s, much smaller than A and populated by a splits generating function. In Figure **??**, we illustrate the MDP constructed for the classification of a toy dataset using some arbitrary splitting function.

6.3.2 Heuristic splits generating functions

A split generating function is any function ϕ that maps an MDP state, i.e., a subset of training examples, to a split node. It has the form $\phi : S \to P(\mathcal{F})$, where $P(\mathcal{F})$ is the power set of all possible split nodes in $\mathcal{F}$. For a state $s \in S$, the state-dependent action space is defined by $A_s = \phi(s) \cup \{1, \dots, K\}$.

When the split generating function does not return all the possible candidate split nodes given a state, solving the MDP with state-dependent actions A_s is not guaranteed to yield the minimizing tree of Eq. 6.1, as the optimization is then performed on the

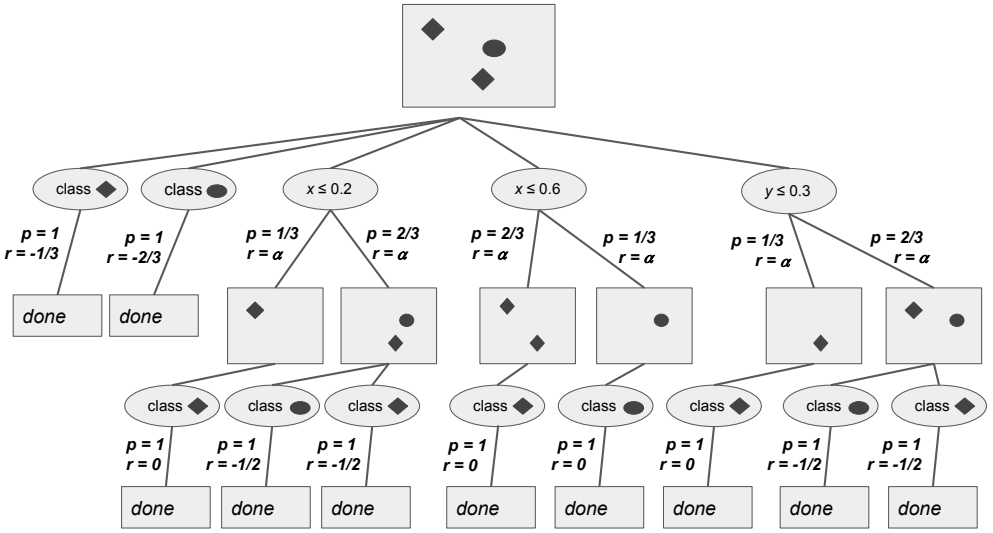


FIGURE 6.1 – Schematics of the MDP to learn a decision tree of depth 2 to classify a toy dataset with three samples, two features (x,y), and two classes (oval, diamond) and using an arbitrary splits generating function.

subset of trees of depth smaller or equal to D, $\mathcal{T}_D$. We now define some interesting split generating functions and provide the time complexity of the associated decision tree algorithms. The time complexity is given in big-O of the number of candidate split nodes considered during computations.

Exhaustive function. When $\mathcal{F} \subseteq \phi(s), \forall s \in S$, the MDP contains all possible splits of a certain set of examples. In this case, *the optimal MDP policy is the optimal decision tree of depth at most D*, and the number of states of the MDP would be $O((2Np)^D)$. Solving the MDP for $A_s = \phi(s)$ is equivalent to running one of the optimal tree induction algorithms [104, 9, 60, 68, 105, 28, 3, 29, 59, 19]

Top B most informative splits. [11] proposed to generate splits with a function that returns, for any state $s = (X, d)$, the B most informative splits over X with respect to some information gain measure such as the entropy or the Gini impurity. The number of states in the MDP would be $O((2B)^D)$. *When $B = 1$, the optimal policy of the MDP is the greedy tree.* In practice, we noticed that the returned set of splits lacked diversity and often consists of splits on the same feature with minor changes to the threshold value.

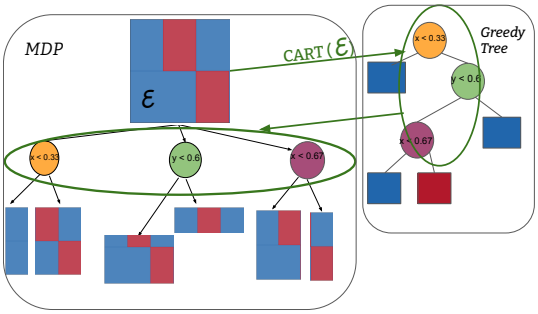


FIGURE 6.2 – How CART is used in DPDT to generate candidate splits given the example data in the current state.

Calls to CART Instead of returning the most informative split at each state $s = (X, d)$, we propose to find the most discriminative split, i.e. the feature splits that best predicts the class of data in X. We can do this by considering the split nodes of the greedy tree. In practice, we run CART on s and use the returned nodes as $\phi(s)$. We control the number of MDP states by constraining CART trees with a maximum number of nodes $B : \phi(s) = nodes(\text{CART}(s, max\_nodes = B))$ The number of MDP states would be $O((2B)^D)$.*When* $B = 1$, *the MDP policy corresponds to the greedy tree.* The process of generating split nodes with calls to CART is illustrated in Figure 6.2.

6.3.3 Dynamic Programming to solve the MDP

Algorithme 14 : DPDT

Data : Dataset $\mathcal{E}$, max depth D, split function $\phi()$,
split function parameter B, regularizing term α
Result : Tree T
$\mathcal{M} \leftarrow build\_mdp(\mathcal{E}, D, \phi(), B)$
// Backward induction
$Q^*(s,a) \leftarrow R_\alpha^{\mathcal{M}}(s,a) + \sum_{s'} P^{\mathcal{M}}(s,a,s') \max_{a' \in A_{s'}^{\mathcal{M}}} Q^*(s',a') \forall s, a \in \mathcal{M}$
// Get the optimal policy
$\pi^*(s) =_{a \in A_s^{\mathcal{M}}} Q^*(s,a) \forall s \in \mathcal{M}$
// Extracting tree from policy
$T \leftarrow E(\pi^*, s_0^{\mathcal{M}})$

After constructing the MDP with a chosen splits generating function, we solve for the optimal policy using dynamic programming. Starting from terminal states and working backward to the initial state, we compute the optimal state-action values using Bellman's

optimality equation [**BELLMAN1958228**], and then deducing the optimal policy.

From now on, we write DPDT to denote Algorithm 14 when the split function is a call to CART. We discuss key bottlenecks when implementing DPDT in subsequent sections. We now state theoretical results when using DPDT with the CART heuristic.

6.3.4 Performance Guarantees of DPDT

We now show that : 1) DPDT minimizes the loss from Eq. 6.1 at least as well as greedy trees and 2) there exists problems for which DPDT has strictly lower loss than greedy trees. As we restrict the action space at a given state s to a subset of all possible split nodes, DPDT is not guaranteed to find the tree minimizing Eq. 6.1. However, we are still guaranteed to find trees that are better or equivalent to those induced by CART :

Theorem 1 (MDP solutions are not worse than the greedy tree)**.** *Let π^* be an optimal deterministic policy of the MDP, where the action space at every state is restricted to the top B most informative or discriminative splits. Let T_0 be the tree induced by CART and $\{T_1, \dots, T_M\}$ all the sub-trees of T_0,* [1] *then for any $\alpha > 0$,*

$$\mathcal{L}_\alpha(E(\pi^*, s_0)) \leq \min_{0 \leq i \leq M} \mathcal{L}_\alpha(T_i)$$

Démonstration. Let us first define $C(T)$, the expected number of splits performed by tree T on dataset $\mathcal{E}$. Here T is deduced from policy π, i.e. $T = E(\pi, s_0)$. $C(T)$ can be defined recursively as $C(T) = 0$ if T is a leaf node, and $C(T) = 1 + p_l C(T_l) + p_r C(T_r)$, where $T_l = E(\pi, s_l)$ and $T_r = E(\pi, s_r)$. In words, when the root of T is a split node, the expected number of splits is one plus the expected number of splits of the left and right sub-trees of the root node. □

It is known that the greedy tree of depth 2 fails to perfectly classify the XOR problem as shown in Figure 5.1 and in [74, 73]. We aim to show that DPDT is a cheap way to alleviate the weaknesses of greedy trees in this type of problems. The following theorem states that there exist classification problems such that DPDT optimizes the regularized training loss strictly better than greedy algorithms such as CART, ID3 or C4.5.

Theorem 2 (DPDT can be strictly better than greedy)**.** *There exists a dataset and a depth D such that the DPDT tree T_D^{DPDT} is strictly better than the greedy tree T_D^{greedy} , i.e, $\mathcal{L}_{\alpha=0}(T_D^{greedy}) > \mathcal{L}_{\alpha=0}(T_D^{DPDT})$.*

1. These sub-trees are interesting to consider since they can be returned by common postprocessing operations following a call to CART, that prune some of the nodes from T_0. Please see [34] for a review of pruning methods for decision trees.

The proof of this theorem is given in the next section.

6.3.5 Proof of Improvement over CART

In this section we construct a dataset for which the greedy tree of depth 2 fails to accurately classify data while DPDT with calls to CART as a splits generating function guarantees a strictly better accuracy. The dataset is the XOR pattern like in Figure 5.1. We will first show that greedy tree induction like CART chooses the first split at random and the second split in between the two columns or rows. Then we will quantify the misclassification of the depth-2 greedy tree on the XOR gate. Finally we will show that using the second greedy split as the root of a tree and then building the remaining nodes greedily, i.e. running DPDT with the CART heuristic, strictly decreases the misclassification.

Definition 18 (XOR dataset)**.** *Let us defined the XOR dataset as* $\mathcal{E}_{XOR} = \{(X_i, Y_i)\}_{i=1}^{N}$. $X_i = (x_i, y_i) \sim \mathcal{U}([0,1]^2)$ *are i.i.d 2-features samples.* $Y_i = f(X_i)$ *are alternating classes with* $f(x,y) = (\lfloor 2x \rfloor + \lfloor 2y \rfloor) \bmod 2$.

Lemma 1. *The first greedy split is chosen at random on the XOR dataset from definition 18.*

Démonstration. Let us consider an arbitrary split $x = x_v$ parallel to the y-axis. The results apply to splits parallel to the x-axis because the XOR pattern is the same when rotated 90 degrees. The split x_v partitions the dataset into two regions R_{left} and R_{right}. Since the dataset has two columns and two rows, any rectangular area that spans the whole height $[0,1)$ has the same proportion of class 0 samples and class 1 samples from definition 18. So in both R_{left} and R_{right} the probabilities of observing class 0 or class 1 at random are $\frac{1}{2}$. Since the class distributions in left and right regions are independent of the split location, all splits have the same objective value when the objective is a measure of information gain like the entropy or the Gini impurity. Hence, the first split in a greedily induced tree is chosen at random. □

Lemma 2. *When the first split is greedy on the XOR dataset from definition 18, the second greedy splits are chosen perpendicular to the first split at* $y = \frac{1}{2}$

Démonstration. Assume without loss of generality due to symmetries, that the first greedy split is vertical, at $x = x_v$, with $x_v <= \frac{1}{2}$. This split partitions the unit square into $R_{left} = [0, x_v) \times [0,1)$ and $R_{right} = [x_v, 1) \times [0,1)$. The split $y = \frac{1}{2}$ further partitions R_{left} into $R_{left-down}$ and $R_{left-up}$ with same areas $x_v \times y = \frac{x_v}{2}$. Due to the XOR pattern, there

are only samples of class 0 in $R_{left-down}$ and only samples of class 1 in $R_{left-up}$. Hence the the split $y = \frac{1}{2}$ maximizes the information gain in R_{left}, hence the second greedy split given an arbitrary first split $x = x_v$ is necessarily $y = \frac{1}{2}$. □

Definition 19 (Forced- Tree)**.** *Let us define the forced-tree as a greedy tree that is forced to make its first split at* $y = \frac{1}{2}$.

Lemma 3. *The forced-tree of depth 2 has a 0 loss on the XOR dataset from definition 18 while, with probability* $1 - \frac{1}{|\mathcal{E}_{XOR}|}$*, the greedy tree of depth 2 has strictly positive loss.*

Démonstration. This is trivial from the definition of the forced tree since if we start with the split $y = \frac{1}{2}$, then clearly CART will correctly split the remaining data. If instead the first split is some $x_v \neq \frac{1}{2}$ then CART is bound to make an error with only one extra split allowed. Since the first split is chosen at random, from Lemma 6.3.5, there are only two splits ($x = \frac{1}{2}$ and $y = \frac{1}{2}$) out of $2|\mathcal{E}_{XOR}|$ that do not lead to sub-optimality. □

We can now formally prove theorem 2.

Démonstration. By definition of DPDT, all instances of DPDT with the CART nodes parameter $B \geq 2$ include the forced-tree from definition 19 in their solution set when applied to the XOR dataset (definition 18). We know from lemma 3 that with high probability, the forced-tree of depth 2 is strictly more accurate than the greedy tree of depth 2 on the XOR dataset. Because we know by proposition 3 that DPDT returns the tree with maximal accuracy from its solution set, we can say that DPDT depth-2 trees are strictly better than depth-2 greedy trees returned by e.g. CART on the XOR dataset. □

6.3.6 Practical Implementation

The key bottlenecks lie in the MDP construction step of DPDT (Section 6.2). In nature, all decision tree induction algorithms have time complexity exponential in the number of training subsets per tree depth D : $O((2B)^D)$, e.g., CART has $O(2^D)$ time complextiy. We already saw that DPDT saves time by not considering all possible tree splits but only B of them. Using state-dependent split generation also allows to generate more or less candidates at different depths of the tree. Indeed, the MDP state $s = (X, d)$ contains the current depth during the MDP construction process. This means that one can control DPDT's time complexity by giving multiple values of maximum nodes : given $(B_1, B_2, ..., B_D)$, the splits generating function in

Algorithm 14 becomes $\phi(s_i) = \phi(X_i, d = 1) = nodes(\text{CART}(s, max\_nodes = B_1))$ and $\phi(s_j) = \phi(X_j, d = 2) = nodes(\text{CART}(s, max\_nodes = B_2))$.

Similarly, the space complexity of DPDT is exponential in the space required to store training examples $\mathcal{E}$. Indeed, the MDP states that DPDT builds in Algorithm 14 are training samples $X \subseteq \mathcal{E}$. Hence, the total space required to run DPDT is $O(Np(2B)^D)$ where Np is the size of $\mathcal{E}$. In practice, one should implement DPDT in a depth first search manner to obtain a space complexity linear in the size of training set : $O(DNp)$. In practice DPDT builds the MDP from Section 6.2 by starting from the root and recursively splitting the training set while backpropagating the Q-values. This is possible because the MDP we solve has a (not necessarily binary) tree structure (see Figure **??**) and because the Q-values of a state only depend on future states.

We implemented DPDT [2] following scikit-learn API [16] with depth-first search and state-depth-dependent splits generating.

6.4 Empirical Evaluation

In this section, we empirically demonstrate strong properties of DPDT trees. The first part of our experiments focuses on the quality of solutions obtained by DPDT for objective Eq.6.1 compared to greedy and optimal trees. We know by theorems 1 and 2 that DPDT trees should find better solutions than greedy algorithms for certain problems ; but what about real problems ? After showing that DPDT can find optimal trees by considering much less solutions and thus performing orders of magnitude less operations, we will study the generalization capabilities of the latter : do DPDT trees label unseen data accurately ?

6.4.1 DPDT optimizing capabilities

From an empirical perspective, it is key to evaluate DPDT training accuracy since optimal decision tree algorithms against which we wish to compare ourselves are designed to optimize the regularized training loss Eq.6.1.

Setup

Metrics : we are interested in the regularized training loss of algorithms optimizing Eq.6.1 with $\alpha = 0$ and a maximum depth D. We are also interested in the number of key operations performed by each baseline, namely computing candidate split nodes for

2. https://github.com/KohlerHECTOR/DPDTreeEstimator

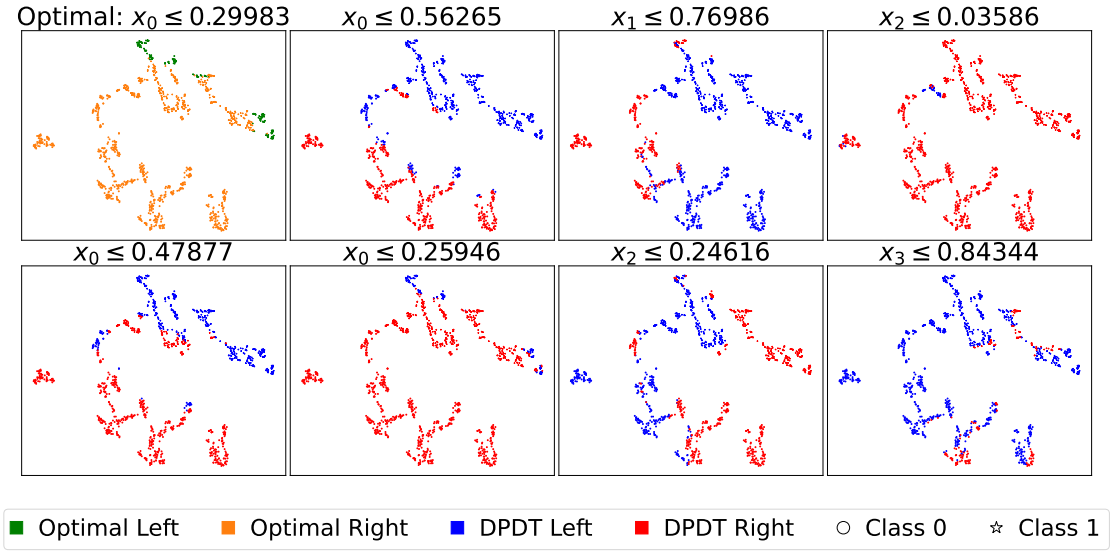


FIGURE 6.3 – Root splits candidate obtained with DPDT compared to the optimal root split on the Bank dataset. Each split creates a partition of p-dimensional data that we projected in the 2-dimensional space using t-SNE.

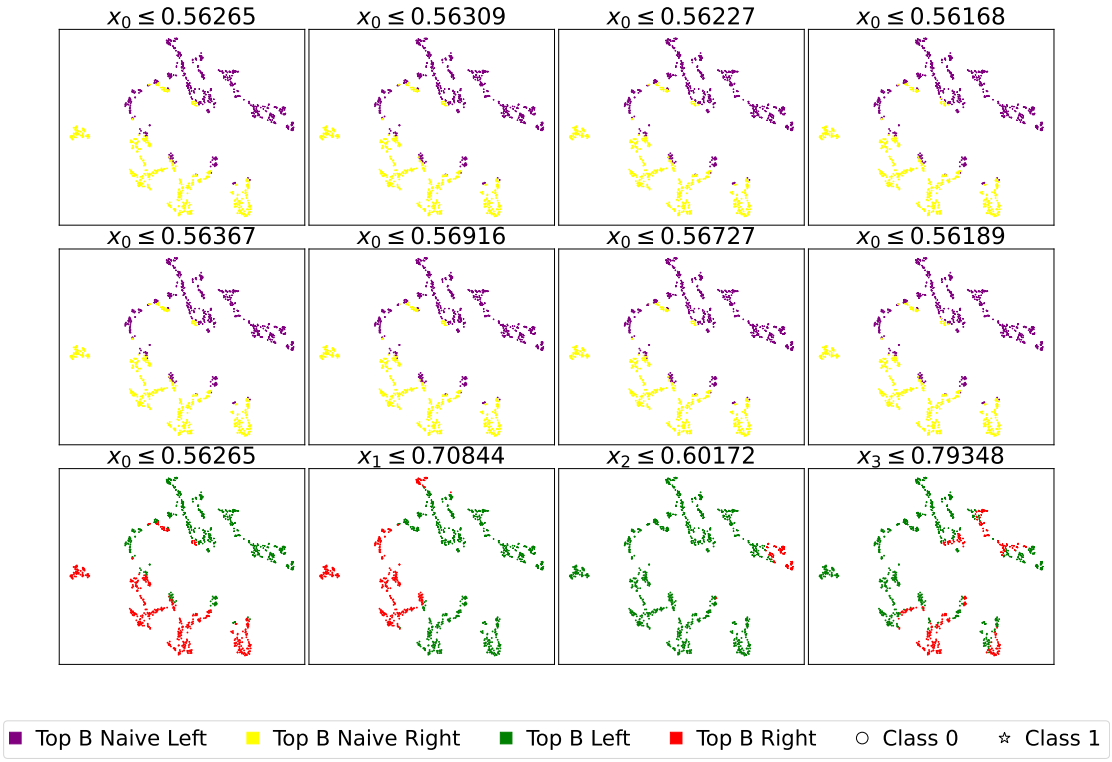


FIGURE 6.4 – Root splits candidate obtained with Top-B[11] on the Bank dataset. Each split creates a partition of p-dimensional data that we projected using t-SNE.

Tableau 6.1 – Comparison of train accuracies of depth-3 trees and number of operations on classification tasks. For DPDT and Top-B, "light" configurations have split function parameters (8, 1, 1) "full" have parameters (8, 8, 8). We also include the mean train accuracy over 5 deep RL runs. **Bold** values are optimal accuracies and blue values are the largest non-optimal accuracies.

			Accuracy								
			Opt	Greedy	DPDT		Top-B		Deep RL	Opt	Greedy
Dataset	N	p	Quant-BnB	CART	light	full	light	full	Custard	Quant-BnB	CART
room	8103	16	**0.992**	0.968	0.991	**0.992**	0.990	**0.992**	0.715	10^6	15
bean	10888	16	**0.871**	0.777	0.812	0.853	0.804	0.841	0.182	$5{\cdot}10^6$	15
eeg	11984	14	**0.708**	0.666	0.689	0.706	0.684	0.699	0.549	$2{\cdot}10^6$	13
avila	10430	10	**0.585**	0.532	0.574	**0.585**	0.563	0.572	0.409	$3{\cdot}10^7$	9
magic	15216	10	**0.831**	0.801	0.822	0.828	0.807	0.816	0.581	$6{\cdot}10^6$	15
htru	14318	8	**0.981**	0.979	0.979	0.980	0.979	0.980	0.860	$6{\cdot}10^7$	15
occup.	8143	5	**0.994**	0.989	0.991	**0.994**	0.990	0.992	0.647	$7{\cdot}10^5$	13
skin	196045	3	**0.969**	0.966	0.966	0.966	0.966	0.966	0.612	$7{\cdot}10^4$	15
fault	1552	27	**0.682**	0.553	0.672	0.674	0.672	0.673	0.303	$9{\cdot}10^8$	13
segment	1848	18	**0.887**	0.574	0.812	0.879	0.786	0.825	0.137	$2{\cdot}10^6$	7
page	4378	10	**0.971**	0.964	0.970	0.970	0.964	0.965	0.902	10^7	15
bidding	5056	9	**0.993**	0.981	0.985	**0.993**	0.985	**0.993**	0.810	$3{\cdot}10^5$	13
raisin	720	7	**0.894**	0.869	0.879	0.886	0.875	0.883	0.509	$4{\cdot}10^6$	15
rice	3048	7	**0.938**	0.933	0.934	0.937	0.933	0.936	0.519	$2{\cdot}10^7$	15
wilt	4339	5	**0.996**	0.993	0.994	0.995	0.994	0.994	0.984	$3{\cdot}10^5$	13
bank	1097	4	**0.983**	0.933	0.971	0.980	0.951	0.974	0.496	$6{\cdot}10^4$	13

subsets of the training data. We disregard running times as solvers are implemented in different programming languages and/or using optimized code : operations count is more representative of an algorithm efficiency. We also qualitatively compare different decision trees root splits to some optimal root split.

Baselines : we benchmark DPDT against greedy trees and optimal trees. For greedy trees we compare DPDT to CART [13]. For optimal trees we compare DPDT to Quant-BnB [68] which is the only solver specialized for depth 3 trees and continuous features. We also consider the non-greedy baseline Top-B [11]. Ideally, DPDT should have training accuracy close to the optimal tree while performing a number of operations close to the greedy algorithm. Furthermore, comparing DPDT to Top-B brings answers to which heuristic splits are better to consider.

We use the CART algorithm implemented in `scikit-learn` [77] in `CPython` with a maximum depth of 3. Optimal trees are obtained by running the `Julia` implementation of the Quant-BnB solver from [68] specialized in depth 3 trees for datasets with conti-

nuous features. We use a time limit of 24 hours per dataset. DPDT and Top-B trees are obtained with Algorithm 14 implemented in pure `Python` and the calls to CART and Top-B most informative splits generating functions from Section 6.2 respectively. We also include Custard, a deep RL baseline [97]. Custard fits a neural network online one datum at a time rather than solving exactly the MDP from Section 6.2 which states are sets of data. Similarly to DPDT, Custard neural network policy is equivalent to a decision tree. We implement Custard with the DQN agent from `stable-baselines3` [85] and train until convergence.

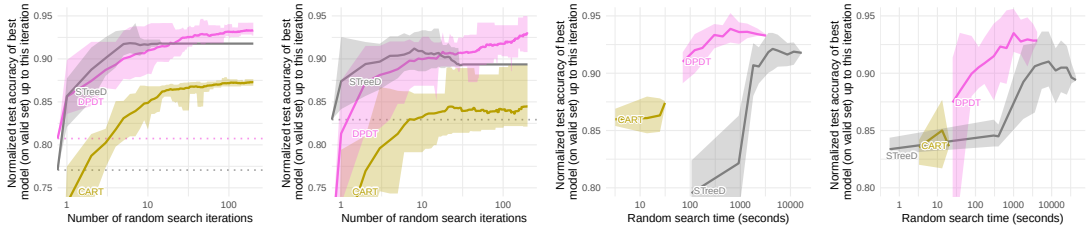
Datasets : we us the same datasets as the Quant-BnB paper [68].

Observations

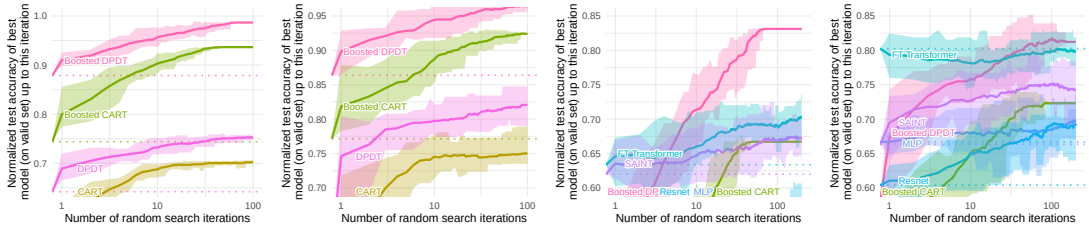
Near-optimality. Our experimental results demonstrate that unlike Deep RL, DPDT and Top-B approaches consistently improve upon greedy solutions while requiring significantly fewer operations than exact solvers. Looking at Table 6.1, we observe several key patterns : first, light DPDT with 16 candidate root splits consistently outperforms the greedy baseline in all datasets. This shows that in practice DPDT can be strictly netter than CART outside of theorem 2 assumptions. Second, when comparing DPDT to Top-B, we see that DPDT generally achieves better accuracy for the same configuration. For example, on the bean dataset, full DPDT reaches 85.3% accuracy while full Top-B achieves 84.1%. This pattern holds on most datasets, suggesting that DPDT is more effective than selecting splits based purely on information gain.

Third, both approaches achieve impressive computational efficiency compared to exact solvers. While optimal solutions require between 10^4 to 10^8 operations, DPDT and Top-B typically need only 10^2 to 10^4 operations, a reduction of 2 to 4 orders of magnitude. Notably, on several datasets (room, avila, occupancy, bidding), full DPDT matches or comes extremely close to optimal accuracy while requiring far fewer operations. For example, on the room dataset, full DPDT achieves the optimal accuracy of 99.2% while reducing operations from 1.34×10^6 to 1.61×10^4. These results demonstrate that DPDT provides an effective middle ground between greedy approaches and exact solvers, offering near-optimal solutions with reasonable computational requirements. While both DPDT and Top-B improve upon greedy solutions, DPDT CART-based split generation strategy appears to be particularly effective at finding high-quality solutions.

DPDT splits To understand why the CART-based split generation yields more accurate DPDT trees than the Top-B heuristic, we visualize how splits partition the feature space



(a) Single Tree Numerical (b) Single Tree Categorical (c) Single Tree Numerical (d) Single Tree Categorical



(e) Boosting vs Single Tree Num. (f) Boosting vs Single Tree Cat. (g) Boosting vs Neural Numerical (h) Boosting vs Neural Categorical

FIGURE 6.5 – Benchmark on medium-sized datasets. Dotted lines correspond to the score of the default hyperparameters, which is also the first random search iteration. Each value corresponds to the test score of the best model (obtained on the validation set) after a specific number of random search iterations (a, b) or after a specific time spent doing random search (c, d), averaged on 15 shuffles of the random search order. The ribbon corresponds to the minimum and maximum scores on these 15 shuffles.

(Figures 6.3, 6.4). We run both DPDT with splits from CART and DPDT with the Top-B most informative splits on the bank dataset. We use t-SNE to create a two-dimensional representations of the dataset partitions given by candidates root splits from CART and Top-B. The optimal root split for the depth-3 tree for bank–obtained with Quant-BnB–is shown on Figure 6.3 in the top-left subplot using green and orange colors for the resulting partitions. On the same figure we can see that the DPDT split generated with CART $x_0 \leq 0.259$ is very similar to the optimal root split. However, on Figure 6.4 we observe that no Top-B candidate splits resemble the optimal root and that in general Top-B split lack diversity : they always split along the same feature. We tried to enforce diversity by getting the most informative split *per feature* but no candidate split resembles the optimal root.

6.4.2 DPDT generalization capabilities

The goal of this section is to have a fair comparison of generalization capabilities of different tree induction algorithms. Fairness of comparison should take into account the number of hyperparameters, choice of programming language, intrinsic purposes of each algorithms (what are they designed to do?), the type of data they can read (categorical features or numerical features). We benchmark DPDT using [44]. We choose this benchmark because it was used to establish XGBoost [20] as the SOTA tabular learning model.

Setup

Metrics : We re-use the code from [44] [3]. It relies on random searches for hyper-parameter tuning [8]. We run a random search of 100 iterations per dataset for each benchmarked tree algorithms. To study performance as a function of the number n of random search iterations, we compute the best hyperparameter combination on the validation set on these n iterations (for each model and dataset), and evaluate it on the test set. Following [44], we do this 15 times while shuffling the random search order at each time. This gives us bootstrap-like estimates of the expected test score of the best tree found on the validation set after each number of random search iterations. In addition, we always start the random searches with the default hyperparameters of each tree induction aglorithm. We use the test set accuracy (classification) to measure model performance. The aggregation metric is discussed in details in [44, Section 3].

Datasets : we use the datasets curated by [44]. They are available on `OpenML` [101] and described in details in [44, Appendix A.1]. The attributes in these datasets are either numerical (a real number), or categorical (a symbolic values among a finite set of possible values). The considered datasets follow a strict selection [44, Section 3] to focus on core learning challenges. Some datasets are very large (millions of samples) like Higgs or Covertype [109, 10]. To ensure non-trivial learning tasks, datasets where simple models (e.g. logistic regression) performed within 5% of complex models (e.g. ResNet [43], HistGradientBoosting [77]) are removed. We use the same data partitioning strategy as [44] : 70% of samples are allocated for training, with the remaining 30% split between validation (30%) and test (70%) sets. Both validation and test sets are capped at 50,000 samples for computational efficiency. All algorithms and hyperparameter combinations were evaluated on identical folds. Finally, while we focus on classification

3. https ://github.com/leogrin/tabular-benchmark

Tableau 6.2 – Hyperparameters importance comparison. A description of the hyperparameters can be found in the scikit-learn documentation : https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

Hyperparameter	DPDT (%)	CART (%)	STreeD (%)
min_samples_leaf	35.05	33.50	50.50
min_impurity_decrease	24.60	24.52	-
cart_nodes_list	15.96	-	-
max_features	11.16	18.06	-
max_depth	7.98	10.19	0.00
max_leaf_nodes	-	7.84	-
min_samples_split	2.67	2.75	-
min_weight_fraction_leaf	2.58	3.14	-
max_num_nodes	-	-	27.51
n_thresholds	-	-	21.98

datasets in the main text, we provide results for regression problems in table 6.5 in the appendix.

Baselines : we benchmark DPDT against CART and STreeD when inducing trees of depth at most 5. We use hyperparameter search spaces from [55] for CART and DPDT. For DPDT we additionally consider eight different splits functions parameters configurations for the maximum nodes in the calls to CART. Surprisingly, after computing the importance of each hyperparameter of DPDT, we found that the maximum node numbers in the calls to CART are only the third most important hyperparametrer behind classical ones like the minimum size of leaf nodes or the minimum impurity decrease (Table 6.2). We use the CPython implementation of STreeD [4]. All hyperparameter grids are given in table 6.7 in the appendix.

Hardware : experiments were conducted on a heterogeneous computing infrastructure made of AMD EPYC 7742/7702 64-Core and Intel Xeon processors, with hardware allocation based on availability and algorithm requirements. DPDT and CART random searches ran for the equivalent of 2-days while PySTreeD ran for 10-days.

Observations

Generalization In Figure 6.5, we observe that DPDT learns better trees than CART and STreeD both in terms of generalization capabilities and in terms of computation

4. PySTreeD : https://github.com/AlgTUDelft/pystreed

Tableau 6.3 – Depth-10 decision trees for the KDD 1999 cup dataset.

Model	Test Accuracy (%)	Time (s)	Memory (MB)
DPDT-(4,)	**91.30**	339.85	5054
DPDT-(4,4,)	**91.30**	881.07	5054
CART	91.29	25.36	1835
GOSDT-$\alpha = 0.0005$	65.47	5665.47	1167
GOSDT-$\alpha = 0.001$	65.45	5642.85	1167

cost. On Figures 6.5a and 6.5b, DPDT obtains best generalization scores for classification on numerical and categorical data after 100 iterations of random hyperparameters search over both CART and STreeD. Similarly, we also present generalization scores as a function of compute time (instead of random search iterations). On Figures 6.5c and 6.5d, despite being coded in the slowest language (`Python` vs. `CPython`), our implementation of DPDT finds the best overall model before all STreeD random searches even finish. The results from Figure 6.5 are appealing for machine learning practitioners and data scientists that have to do hyperparameters search to find good models for their data while having computation constrains.

Now that we have shown that DPDT is extremely efficient to learn shallow decision trees that generalize well to unseen data, it is fair to ask if DPDT can also learn deep trees on very large datasets.

Deeper trees on bigger datasets. We also stress test DPDT by inducing deep trees of depth 10 for the KDD 1999 cup dataset [5]. The training set has 5 million rows and a mix of 80 continuous and categorical features representing network intrusions. We fit DPDT with 4 split candidates for the root node (DPDT-(4,)) and with 4 split candidates for the root and for each of the internal nodes at depth 1 (DPDT-(4,4,)). We compare DPDT to CART with a maximum depth of 10 and to GOSDT [6] [**McTavish_Zhong_Achermann_Karimalis_Chen**
with different regularization values α. GOSDT first trains a tree ensemble to binarize a dataset and then solve for the optimal decision tree of depth 10 on the binarized problem. In Table 6.3 we report the test accuracy of each tree on the KDD 1999 cup test set. We also report the memory peak during training and the training duration (all experiments are run on the same CPU). We observe that DPDT can improve over CART even for deep trees and large datasets while using reasonable time and memory. Furthermore, Table 6.3 highlights the limitation of optimal trees for practical problems when the dataset is not binary. We observed that GOSDT could not find a good binarization of the dataset

5. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`
6. Code available at : `https://github.com/ubc-systopia/gosdt-guesses`

even when increasing the budget of the tree ensemble up to the point where most of the computations are spent on fitting the ensemble (see more details about this phenomenon in [**McTavish_Zhong_Achermann_Karimalis_Chen_Rudin_Seltzer_2022**]). In table 6.6 in the appendix, we also show that DPDT performs better than optimal trees for natively binary datasets. In the next section we study the performance of boosted DPDT trees.

6.5 Application of DPDT to Boosting

In the race for machine learning algorithms for tabular data, boosting procedures are often considered the go-to methods for classification and regression problems. Boosting algorithms [37, 39, 38] sequentially add weak learners to an ensemble called strong learner. The development of those boosting algorithms has focused on what data to train newly added weak learners [39, 38], or on efficient implementation of those algorithms [20, 79]. We show next that Boosted-DPDT (boosting DPDT trees with AdaBoost [37]) improves over recent deep learning algorithms.

6.5.1 Boosted-DPDT

We benchmark Boosted-DPDT with the same datasets, metrics, and hardware as in the previous section on single-tree training. Second, we verify the competitiveness of Boosted-DPDT with other models such as deep learning ones (SAINT [94] and other deep learning architectures from [43]).

On Figures 6.5e and 6.5f we can notice 2 properties of DPDT. First, as in any boosting procedure, Boosted-DPDT outperforms its weak counterpart DPDT. This serves as a sanity check for boosting DPDT trees. Second, it is clear that boosting DPDT trees yields better models than boosting CART trees on both numerical and categorical data. Figures 6.5g and 6.5h show that boosting DPDT trees using the default AdaBoost procedure [37] is enough to learn models outperforming deep learning algorithms on datasets with numerical features and models in the top-tier on datasets with categorical features. This show great promise for models obtained when boosting DPDT trees with more advanced procedures.

6.5.2 (X)GB-DPDT

We also boost DPDT trees with Gradient Boosting and eXtreme Gradient Boosting [38, 39, 20](X(GB)-DPDT). For each dataset from [44], we trained (X)GB-DPDT

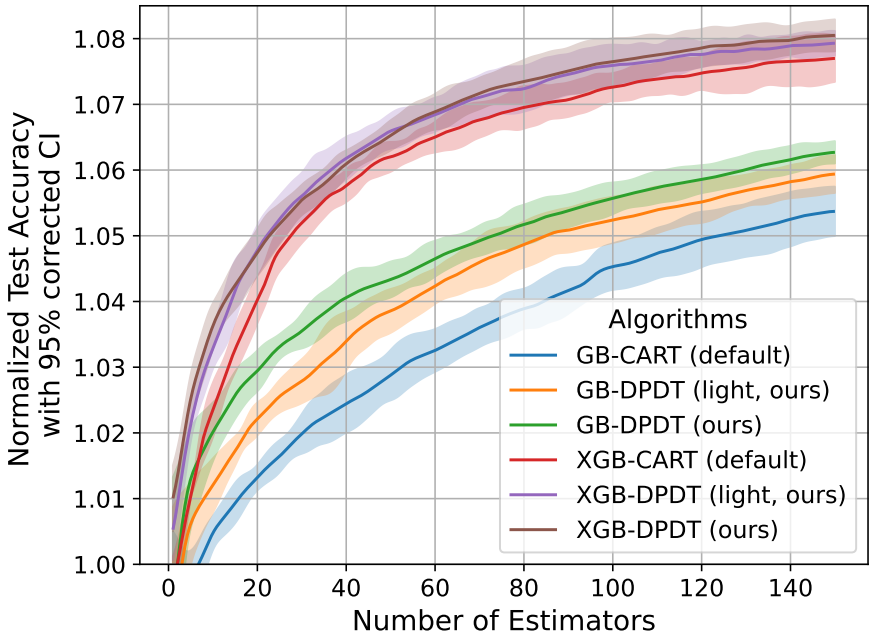


FIGURE 6.6 – Aggregated mean test accuracies of Gradient Boosting models as a function of the number of single trees.

models with 150 boosted single DPDT trees and a maximum depth of 3 for each. We evaluate two DPDT configurations for the single trees : light (DPDT-(4, 1, 1)) and the default (DPDT-(4,4,4)). We compare (X)GB-DPDT to (X)GB-CART : 150 boosted CART trees with maximum depth of 3 and default hyperparameters for each. All models use a learning rate of 0.1. For each each dataset, we normalize all boosted models scores by the accuracy of a single depth-3 CART decision tree and aggregate the results : the final curves represent the mean performance across all datasets, with confidence intervals computed using 5 different random seeds.

Figure 6.6 shows that similarly to simple boosting procedures like AdaBoost, more advanced ones like (eXtreme) Gradient Boosting yields better models when the weak learners are DPDT trees rather than greedy trees. This is a motivation to develop efficient implementation of (eXtreme) Gradient Boosting with DPDT as the weak learning algorithm to perform extensive benchmarking following [44] and potentially claim the state-of-the-art.

6.6 Proof of Propostion 3

For the purpose of the proof, we overload the definition of J_α and $\mathcal{L}_\alpha$, to make explicit the dependency on the dataset and the maximum depth. As such, $J_\alpha(\pi)$ becomes $J_\alpha(\pi, \mathcal{E}, D)$ and $\mathcal{L}_\alpha(T)$ becomes $\mathcal{L}_\alpha(T, \mathcal{E})$. Let us first show that the relation $J_\alpha(\pi, \mathcal{E}, 0) = -\mathcal{L}_\alpha(T, \mathcal{E})$ is true. If the maximum depth is $D = 0$ then $\pi(s_0)$ is necessarily a class assignment, in which case the expected number of splits is zero and the relation is obviously true since the reward is the opposite of the average classification loss. Now assume it is true for any dataset and tree of depth at most D with $D \geq 0$ and let us prove that it holds for all trees of depth $D+1$. For a tree T of depth $D+1$ the root is necessarily a split node. Let $T_l = E(\pi, s_l)$ and $T_r = E(\pi, s_r)$ be the left and right sub-trees of the root node of T. Since both sub-trees are of depth at most D, the relation holds and we have $J_\alpha(\pi, X_l, D) = \mathcal{L}_\alpha(T_l, X_l)$ and $J_\alpha(\pi, X_r, D) = \mathcal{L}_\alpha(T_r, X_r)$, where X_l and X_r are the datasets of the "right" and "left" states to which the MDP transitions—with probabilities p_l and p_r—upon application of $\pi(s_0)$ in s_0, as described in the MDP formulation. Moreover, from the definition of the policy return we have

$$
\begin{aligned}
J_\alpha(\pi, \mathcal{E}, D+1) &= -\alpha + p_l * J_\alpha(\pi, X_l, D) + p_r * J_\alpha(\pi, X_r, D) \\
&= -\alpha - p_l * \mathcal{L}_\alpha(T_l, X_l) - p_r * \mathcal{L}_\alpha(T_r, D) \\
&= -\alpha - p_l * \left(\frac{1}{|X_l|} \sum_{(x_i, y_i) \in X_l} \ell(y_i, T_l(x_i)) + \alpha C(T_l) \right) \\
&\quad - p_r * \left(\frac{1}{|X_r|} \sum_{(x_i, y_i) \in X_r} \ell(y_i, T_r(x_i)) + \alpha C(T_r) \right) \\
&= -\frac{1}{N} \sum_{(x_i, y_i) \in X} \ell(y_i, T(x_i)) - \alpha(1 + p_l C(T_l) + p_r C(T_r)) \\
&= -\mathcal{L}(T, \mathcal{E})
\end{aligned}
$$

6.7 Additional Experiments and Hyperparameters

In this section we provide additional experimental results. In Table 6.5, we compare DPDT trees to CART and STreeD trees using 50 train/test splits of regression datasets from [44]. All algorithms are run with default hyperparameters.

The configuration of DPDT is (4, 4, 4) or (4,4,4,4,4). STreeD is run with a time limit of 4 hours per tree computation and on binarized versions of the datasets. Both for depth-3 and depth-5 trees, DPDT outperforms other baselines in terms of train and test accuracies. Indeed, because STreeD runs on "approximated" datasets, it performs pretty poorly.

In Table 6.6, we compare DPDT(4, 4, 4, 4, 4) to additional optimal decision tree baselines on datasets with **binary features**. The optimal decision tree baselines run with default hyperparameters and a time-limit of 10 minutes. The results show that even on binary datasets that optimal algorithms are designed to handle well ; DPDT outperforms other baselimes. This is likely because optimal trees are slow and/or don't scale well to depth 5.

In Table 6.4 compare DPDT to lookahead depth-3 trees when optimizing Eq.6.1. Unlike the other greedy approaches, lookahead decision trees [**norton**] do not pick the split that optimizes a heuristic immediately. Instead, they pick a split that sets up the best possible heuristic value on the following split. Lookahead-1 chooses nodes at depth $d < 3$ by looking 1 depth in the future : it looks for the sequence of 2 splits that maximizes the information gain at depth $d + 1$. Lookahead-2 is the optimal depth-3 tree and Lookahead-0 would be just building the tree greedily like CART. The conclusion are roughly the same as for Table 6.1. Both lookahead trees and DPDT [7] are in `Python` which makes them slow but comparable.

We also provide the hyperparameters necessary to reproduce experiments from section 6.4.2 and 6.5.1 in Table 6.7.

7. `https://github.com/KohlerHECTOR/DPDTreeEstimator`

Tableau 6.4 – Train accuracies of depth-3 trees (with number of operations). Lookahead trees are trained with a time limit of 12 hours.

Dataset	**DPDT**	**Lookahead-1**
avila	57.22 (1304)	OoT
bank	97.99 (699)	96.54 (7514)
bean	85.30 (1297)	OoT
bidding	99.27 (744)	98.12 (20303)
eeg	69.38 (1316)	69.09 (10108)
fault	67.40 (1263)	67.20 (32514)
htru	98.01 (1388)	OoT
magic	82.81 (1451)	OoT
occupancy	99.31 (1123)	99.01 (15998)
page	97.03 (1243)	96.44 (16295)
raisin	88.61 (1193)	86.94 (9843)
rice	93.44 (1367)	93.24 (37766)
room	99.23 (1196)	99.04 (5638)
segment	87.88 (871)	68.83 (24833)
skin	96.60 (1300)	96.61 (1290)
wilt	99.47 (862)	99.31 (36789)

Tableau 6.5 – Mean train and test scores (with standard errors) for regression datasets over 50 cross-validation runs.

	Depth 3								
Dataset	DPDT		Optimal		CART		DPDT		
	Train	Test	Train	Test	Train	Test	Train	Test	
nyc-taxi	39.0 ± 0.0	38.9 ± 0.2	33.8 ± 0.0	33.8 ± 0.1	39.0 ± 0.0	38.9 ± 0.2	45.8 ± 0.0	45.7 ± 0.2	33.
medical_charges	95.2 ± 0.0	95.2 ± 0.0	90.1 ± 0.0	90.1 ± 0.1	95.2 ± 0.0	95.2 ± 0.0	97.7 ± 0.0	97.7 ± 0.0	90.
diamonds	93.0 ± 0.0	92.9 ± 0.1	90.1 ± 0.0	90.1 ± 0.1	92.7 ± 0.0	92.6 ± 0.1	94.2 ± 0.0	94.0 ± 0.1	90.
house_16H	39.9 ± 0.1	38.1 ± 2.5	32.8 ± 0.1	29.4 ± 1.6	35.8 ± 0.1	35.8 ± 1.9	59.4 ± 0.1	35.2 ± 4.1	32.
house_sales	67.0 ± 0.0	66.0 ± 0.4	65.1 ± 0.0	64.4 ± 0.4	66.8 ± 0.0	66.1 ± 0.4	77.6 ± 0.0	76.1 ± 0.3	65.
superconduct	73.1 ± 0.0	72.7 ± 0.5	70.9 ± 0.0	70.5 ± 0.5	70.4 ± 0.0	69.7 ± 0.5	83.0 ± 0.0	81.7 ± 0.4	70.
houses	51.7 ± 0.0	50.7 ± 0.7	48.5 ± 0.1	47.3 ± 0.7	49.5 ± 0.0	48.4 ± 0.7	69.1 ± 0.0	67.6 ± 0.5	48.
Bike_Sharing	55.2 ± 0.0	54.7 ± 0.5	45.1 ± 0.1	44.8 ± 0.7	48.1 ± 0.0	47.9 ± 0.5	65.2 ± 0.0	63.3 ± 0.5	45.
elevators	48.0 ± 0.0	46.8 ± 1.1	40.2 ± 0.1	38.2 ± 1.0	46.8 ± 0.0	45.5 ± 1.2	65.6 ± 0.0	61.2 ± 1.0	40.
pol	72.2 ± 0.0	71.3 ± 0.6	67.8 ± 0.1	67.5 ± 0.9	72.0 ± 0.0	71.2 ± 0.8	93.3 ± 0.0	92.4 ± 0.3	67.
MiamiHousing2016	62.3 ± 0.0	60.4 ± 0.8	60.8 ± 0.0	58.3 ± 0.8	62.3 ± 0.0	60.6 ± 0.8	79.8 ± 0.0	77.5 ± 0.5	60.
Ailerons	63.5 ± 0.0	62.6 ± 0.7	61.6 ± 0.0	60.3 ± 0.7	63.5 ± 0.0	62.6 ± 0.7	76.0 ± 0.0	72.9 ± 0.6	61.
Brazilian_houses	90.7 ± 0.0	90.3 ± 0.8	89.2 ± 0.0	89.4 ± 0.8	90.7 ± 0.0	90.4 ± 0.8	97.6 ± 0.0	96.6 ± 0.4	89.
sulfur	72.5 ± 0.1	66.6 ± 2.2	35.7 ± 0.1	19.1 ± 6.7	72.0 ± 0.1	68.0 ± 2.2	89.0 ± 0.0	68.4 ± 6.7	35.
yprop_41	6.3 ± 0.0	2.3 ± 0.7	3.6 ± 0.0	1.5 ± 0.4	6.2 ± 0.0	2.1 ± 0.8	13.2 ± 0.0	1.2 ± 1.7	3.
cpu_act	93.4 ± 0.0	92.0 ± 0.6	89.0 ± 0.0	86.5 ± 1.9	93.4 ± 0.0	92.0 ± 0.6	96.5 ± 0.0	94.7 ± 0.5	89.
wine_quality	27.9 ± 0.0	23.3 ± 0.9	25.2 ± 0.0	23.7 ± 0.8	27.7 ± 0.0	24.5 ± 0.8	37.4 ± 0.0	26.7 ± 1.0	25.
abalone	46.3 ± 0.0	39.6 ± 1.6	42.5 ± 0.0	40.4 ± 1.4	43.3 ± 0.0	39.2 ± 1.2	58.6 ± 0.0	44.7 ± 1.8	42.

TABLEAU 6.6 – Train/test accuracies of different decision tree induction algorithms. All algorithms induce trees of depth at most 5 on 8 classification datasets. A time limit of 10 minutes is set for OCT-type algorithms. The values in this table are averaged over 3 seeds giving 3 different train/test datasets.

Datasets				Train Accuracy depth-5					Test	
Names	Samples	Features	Classes	DPDT	OCT	MFOCT	BinOCT	CART	DPDT	OCT
balance-scale	624	4	3	90.9%	71.8%	82.6%	67.5%	86.5%	77.1%	66.9%
breast-cancer	276	9	2	94.2%	88.6%	91.1%	75.4%	87.9%	66.4%	67.1%
car-evaluation	1728	6	4	92.2%	70.1%	80.4%	84.0%	87.1%	90.3%	69.5%
hayes-roth	160	9	3	93.3%	82.9%	95.4%	64.6%	76.7%	75.4%	77.5%
house-votes-84	232	16	2	100.0%	100.0%	100.0%	100.0%	99.4%	95.4%	93.7%
soybean-small	46	50	4	100.0%	100.0%	100.0%	76.8%	100.0%	93.1%	94.4%
spect	266	22	2	93.0%	92.5%	93.0%	92.2%	88.5%	73.1%	75.6%
tic-tac-toe	958	24	2	90.8%	68.5%	76.1%	85.7%	85.8%	82.1%	69.6%

TABLEAU 6.7 – Hyperparameter search spaces for tree-based models. More details about the hyperparamters meaning are given in [55].

Parameter	**CART**	**Boosted-CART**	**DPDT**	**Boosted-DPDT**	**STreeD**
Common Tree Parameters					
max_depth	{5 : 0.7, 2,3,4 : 0.1}	{2 : 0.4, 3 : 0.6}	{5 : 0.7, 2,3,4 : 0.1}	{2 : 0.4, 3 : 0.6}	5
min_samples_split	{2 : 0.95, 3 : 0.05}	{2 : 0.95, 3 : 0.05}	{2 : 0.95, 3 : 0.05}	{2 : 0.95, 3 : 0.05}	–
min_impurity_decrease	{0.0 : 0.85, 0.01,0.02,0.05 : 0.05}	{0.0 : 0.85, 0.01,0.02,0.05 : 0.05}	{0.0 : 0.85, 0.01,0.02,0.05 : 0.05}	{0.0 : 0.85, 0.01,0.02,0.05 : 0.05}	–
min_samples_leaf	$\mathcal{Q}(\log\text{-}\mathcal{U}[2,51])$	$\mathcal{Q}(\log\text{-}\mathcal{U}[2,51])$	$\mathcal{Q}(\log\text{-}\mathcal{U}[2,51])$	$\mathcal{Q}(\log\text{-}\mathcal{U}[2,51])$	$\mathcal{Q}(\log\text{-}\mathcal{U}[2,51])$
min_weight_fraction_leaf	{0.0 : 0.95, 0.01 : 0.05}	{0.0 : 0.95, 0.01 : 0.05}	{0.0 : 0.95, 0.01 : 0.05}	{0.0 : 0.95, 0.01 : 0.05}	–
max_features	{"sqrt" : 0.5, "log2" : 0.25, 10000 : 0.25}	{"sqrt" : 0.5, "log2" : 0.25, 10000 : 0.25}	{"sqrt" : 0.5, "log2" : 0.25, 10000 : 0.25}	{"sqrt" : 0.5, "log2" : 0.25, 10000 : 0.25}	–
Model-Specific Parameters					
max_leaf_nodes	{32 : 0.85, 5,10,15 : 0.05}	{8 : 0.85, 5 : 0.05, 7 : 0.1}	–	–	–
cart_nodes_list	–	–	8 configs (uniform)	5 configs (uniform)	–
learning_rate	–	$\log\mathcal{N}(\ln(0.01), \ln(10))$		$\log\mathcal{N}(\ln(0.01), \ln(10))$	
n_estimators	–	1000	–	1000	–
max_num_nodes	–	–	–	–	{3,5,7,11, 17,25,31} (uniform)
n_thresholds	–	–	–	–	{5,10,20,50} (uniform)
cost_complexity	–	–	–	–	0
time_limit	–	–	–	–	1800

Chapitre 7

Conclusion

7.1 Conclusion

In this paper, we introduced Dynamic Programming Decision Trees (DPDT), a novel framework that bridges the gap between greedy and optimal decision tree algorithms. By formulating tree induction as an MDP and employing adaptive split generation based on CART, DPDT achieves near-optimal training loss with significantly reduced computational complexity compared to existing optimal tree solvers. Furthermore, we prove that DPDT can learn strictly more accurate trees than CART.

Most importantly, extensive benchmarking on varied large and difficult enough datasets showed that DPDT trees and boosted DPDT trees generalize better than other baselines. To conclude, we showed that DPDT is a promising machine learning algorithm.

The key future work would be to make DPDT industry-ready by implementing it in `C` and or making it compatible with the most advanced implementation of e.g. XGBoost.

7.2 What about imitation ?

Troisième partie

Beyond Decision Trees : what can be done with other Interpretable Policies ?

Chapitre 8

Imitation and Evaluation

8.1 Intro

There exist applications of reinforcement learning like medicine where policies need to be "interpretable" by humans. User studies have shown that some policy classes might be more interpretable than others. However, it is costly to conduct human studies of policy interpretability. Furthermore, there is no clear definition of policy interpretabiliy, i.e., no clear metrics for interpretability and thus claims depend on the chosen definition. We tackle the problem of empirically evaluating policies interpretability without humans. Despite this lack of clear definition, researchers agree on the notions of "*simulatability*" : policy interpretability should relate to how humans understand policy actions given states. To advance research in interpretable reinforcement learning, we contribute a new methodology to evaluate policy interpretability. We distillate expert neural networks policies into small programs that we use as baselines. We then show that using our methodology to evaluate the baselines interpretability leads to similar conclusions as user studies. Most importantly, we show that there is no policy class that better trades off interpretability and performance across tasks.

There is increasing research in developing reinforcement learning algorithms that return "interpretable" policies such as trees, programs, first-order logic, or linear maps [**kohler2024interpretableeditableprogrammatictree**, 6, 103, 64, 25, 70, 42]. Indeed, interpretability has been useful for different applications : policy verification [6], misalignment detection [26, 67] and features importance analysis [108, 4, 1].

User studies have established the common beliefs that decision trees are more "interpretable" than linear maps, oblique trees (trees where nodes are tests of linear combina-

tions of features), and multi-layer perceptrons (MLPs) [35, 36, 66, 102]. Furthermore, for a fixed class of models, humans give different values of interpretability to models with different numbers of parameters [56]. However, survey works argue that every belief about interpretability needs to be verified with user studies and that interpretability evaluations are grounded to a specific set of users, to a specific application, and to a specific definition of interpretability [**rigorous**, **mythos**]. For example, [**mythos**] claims that depending on the notion of *simulatability* studied, MLPs can be more interpretable than trees, since deep trees can be harder for a human to read than compact MLPs. Hence, even with access to users it would be difficult to research interpretability. More realistically, since the cost of user studies is high (time, variety of subjects required, ethics, etc.), designing proxies for interpretability in machine learning has become an important open problem in both supervised [**rigorous**] and reinforcement learning [42].

In this work, we propose a methodology to evaluate the interpretability of reinforcement learning without human evaluators, by measuring inference times and memory consumptions of policies as programs. We show that those measures constitute adequate proxies for the notions of "*simulatability*" described in [**mythos**], which relates the interpretability of policy to humans ability to understand the inference of actions given states. In addition to the contributions summarized next, we open source some of the interpretable baselines to be be used for future interpretability research and teaching [1].

1. https://anonymous.4open.science/r/interpretable-rl-zoo-4DCC/README.md

Chapitre 9

Evaluation

Oblique decision trees. One can imitate oracles with programs that make tests of linear combinations of features. Many oracles learn oblique or more complex decision rules over an MDP state space. This is illustrated in Figure 9.3 where a PPO neural oracle creates oblique partitions of the state-space for the Pong environments. Programs that test only individual features would fail to fit this partition (see Figure 9.3). We thus modify CART breiman, an algorithm returning axes-parallel trees for regression and supervised classification problems, for it to return oblique decision trees.

In addition to single feature tests, our oblique trees consider linear combinations of two features with weights 1 and −1, e.g., for MDP states $s_i \in \mathbb{R}^p$, the oblique features values are $s_i^{oblique} = \{s_{i1}-s_{i0}, s_{i2}-s_{i0}, \ldots, s_{ip}-s_{i0}, \ldots, s_{ip-1}-s_{ip}\} \in \mathbb{R}^{p^2}$. For example, using an oracle dataset with n state-actions pairs : $(\bar{S}, \bar{A} = \pi^*(\bar{S})) \subsetneq \mathbb{R}^{n\cdot(p+\dim(A))}$, we obtain oblique decision trees by fitting $(\bar{S}, \bar{S}^{oblique}, \bar{A} = \pi^*(\bar{S})) \subsetneq \mathbb{R}^{n\cdot(p(p+1)+\dim(A))}$. Given $\bar{S}$, computing

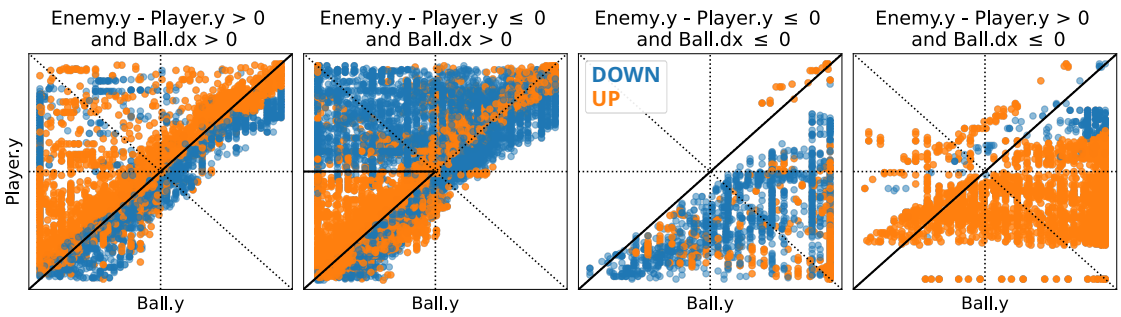


FIGURE 9.1 – **Oracle decision rules are oblique** illustrated on PPO for different state space partitions of the Pong environment. Decisions boundaries are both oblique and parallel.

$\bar{S}^{oblique}$ can be done efficiently by computing the values of the lower (or upper) triangles in the $\bar{S} \otimes \bar{S} - (\bar{S} \otimes \bar{S})^T$ tensor (excluding the diagonals) (see line **??** of Algorithm **??**). We further demonstrate the superiority of oblique trees in our experimental evaluation on a diverse set of RL tasks.

[7]r0.55

MDP	Ast.	Box.	Free.	Kang.	Pong	Sea.	SpaceI.	Ten.
Full	100	8	48	196	12	172	176	16
Simplified	90	8	22	28	8	54	164	16

The complexity of building the tree (line **??** of algorithm **??**) is $O(pn\log_2(n))$ when no maximum tree depth is given, and $O(pnD)$ with a maximum tree depth of D complexcart. In particular, at iteration i of our algorithm the complexity of building the tree is $O(p(p+1)itD)$, as rollouts of t MDP transitions are aggregated (line **??**) and oblique features are added to states (line **??**). This means that at each iteration i, the cost of computing an oblique tree is $p+1$ times the cost of computing an axes-parallel tree. In our algorithm we pass K the maximum number of leaf nodes as an argument. A tree with K leaf nodes has $2K-1$ total nodes and a depth of at most $D = K-1$.

9.0.1 Real life use case of tree programs for fertilization of soils (Q3)

[18]r0.53 Pythonannexes/programmes/play$_d$ssat$_o$blique$_6$6.2$_c$opy$_h$uman.py In this last experiment, we distill a human expert policy for soil fertilization on the gym_DSSAT gym environment [**gautron2023learning**]. Here, an RL agent has to learn to manage a crop, based on an accurate simulated mechanistic model of plant growth. We consider the task that consists in optimizing plant nitrogen absorption while penalizing the application of fertilizer to minimize the economical and the environmental costs. We extract an 's Python program, depicted in Figure 9.0.1. This program outputs the exact same actions as the human heuristic given the soil state and obtain the same cumulative reward in average (corresponding to an accuracy of 100%). It also provides an interpretation of the human expert heuristic that delivers a certain amount of fertilizer ($\{27, 35, 54\}$) after $\{39, 45, 80\}$ days after seeding, respectively). The feature importance coincides with agronomic principles and have been validated by an expert from the *Consultative Group on International Agricultural Research*. The nitrogen requirements of corn vary depending on the growth stage. They are important during the vegetative phase (plant growth) and the reproductive phase (from flowering to grain filling). This is why it is essential to consider the number of days after planting and the growth stage of the corn, as nitrogen requirements are highest during grain filling.

9.1 Methodology Overview

In this section, we explain our methodology for the evaluation of interpretability. Our approach consists of three main steps : (1) obtaining deep neural network policies trained with reinforcement learning that obtain high cumulative rewards, (2) distilling those policies into less complex ones to use as baselines (3) after parsing baselines from different classes into a common comparable language, we evaluate the interpretability of the policies using proxy metrics for *simulatability*.

Deep Neural Network Policies In reinforcement learning, an agent learns how to behave in an environment to maximize a cumulative reward signal [95]. The environment is defined by a Markov decision process (MDP) $M = (\mathcal{S}, \mathcal{A}, T, R)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state-transition function, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function. At each time step t, an agent observes the current state s_t and chooses an action a_t according to its policy π. The agent executes a_t, receives reward r_t, and observes the next state s'_{t+1}. The goal is to find an optimal policy π^* that maximizes the expected discounted future return : $\pi^* = \text{argmax}_\pi Q^\pi(s, a) = \text{argmax}\, \mathbb{E}[r + \gamma Q^\pi(s', a)]$, with γ a discount factor in $[0, 1)$. For large or continuous state spaces like the MDPs we consider in this work, MLPs are used to represent Q^π or π. While these MLPs can be trained efficiently to obtain high cumulative rewards [90, 71], they are too complex for interpretability considerations.

Distilling into Interpretable Policies To obtain interpretable policies, we distill the complex neural networks into simpler models using imitation learning, as described in Algorithm 15. This approach transforms the reinforcement learning task into a sequence of supervised learning problems.

Algorithm 15 inputs an environment, that simulates taking steps in an MDP, an expert policy to imitate, also called a teacher, and an (interpretable) policy class to fit, also called student. The hyperparameters of Algorithm 15 are : the number of times we fit a student policy, the total number of samples to be collected, and whether or not to use importance sampling. At each iteration of Algorithm 15 the student policy is fitted to a dataset of states collected with the expert at iteration 1 or with the previously fitted student (see Line 15). The actions are always given by the expert (see Line 15). When using importance sampling, the states are further re-weighted by the worst state-action value possible in the given state. When the number of iteration is 1, Algorithm 15 is behavior cloning [78]. When we use importance sampling, Algorithm 15 is Q-DAgger [6].

Algorithme 15 : Imitate Expert [78, 88, 6]

Input : Expert policy π^*, MDP M, policy class Π, number of iterations N, total samples S, importance sampling flag I
Output : Fitted student policy $\hat{\pi}_i$
Initialize dataset $\mathcal{D} \leftarrow \emptyset$;
Initialize $\hat{\pi}_1$ arbitrarily from Π;
for $i \leftarrow 1$ **to** N **do**
  if $i = 1$ **then** $\pi_i \leftarrow \pi^*$;
  else $\pi_i \leftarrow \hat{\pi}_i$;
  Sample S/N transitions from M using π_i;
  if I *is True* **then** $w(s) \leftarrow V^{\pi*}(s) - \min_a Q^{\pi*}(s,a)$;
  else $w(s) \leftarrow 1$;
  Collect dataset $\mathcal{D}_i \leftarrow \{(s, \pi^*(s), w(s))\}$ of states visited by π_i and expert actions;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$;
  Fit classifier/regressor $\hat{\pi}_{i+1}$ on $\mathcal{D}$;
end
return $\hat{\pi}_N$;

In other cases, Algorithm 15 is DAgger [88].

Measuring Policy Interpretability After obtaining interpretable policy baselines using Algorithm 15, we use two metrics to evaluate policy interpretability without requiring human intervention. Those metrics are proxies for the notion of *simulatability* from [**mythos**] that gives insights on how a human being would read a policy to understand how actions are inferred. In particular, *simulatability* admits two sub-definitions. The first one is a measure of how difficult it is for a human to reproduce the computations of the policy to infer actions given states. The second one measures how difficult it is for a human to read through the entire policy. [**mythos**] argues that this nuance is key when measuring interpretability because a tree is not read entirely to compute a single action and because there is no consensus on what is easier for a human to read between an MLP and a tree.

1. Policy Inference Time : to measure how a human would compute the action of a policy given a state at each environment step, we measure policy step inference time in seconds.

2. Policy Size : to measure how easily a human can read the entire policy, we measure its size in bytes. While this correlates with inference time for MLPs and linear models, tree-based policies may have large sizes but quick inference because they do not traverse all decision paths at each step.

As these measurements depend on many technical details (programming language, the compiler if any, the operating system, versions of libraries, the hardware it is executed on, etc), to ensure fair comparisons, we translate all student policies into a simple representation that mimics how a human being "reads" a policy. We call this process of standardizing policies language “unfolding”. In Figure 9.4, 9.5, and 9.6, we present some unfolded policy programs. Other works have distilled neural networks into programs [103] or even directly learn programmatic policies [81] from scratch. However, those works directly consider programs as a policy class and could compare a generic program (not unfolded, with, e.g., while loops or array operations) to, e.g, a decision tree [99]. We will discuss later on the limitations of unfolding policies in the overall methodology.

9.2 Computing Baseline Policies

9.2.1 Setup

All the experiments presented next run on a dedicated cluster of Intel Xeon Gold 6130 (Skylake-SP), 2.10GHz, 2 CPUs/node, 16 cores/CPU with a timeout of 4 hours per experiment. Codes to reproduce our results are given in the supplementary material. In the future, we will open source a python library with all the tools of our methodology. Using Algorithm 15, we distill deep neural network expert policies into less complex policy classes.

Policy Class	**Parameters**	**Training Algorithm**
Linear Policies	Determined by state-action dimensions	Linear/Logistic Regression
Decision Trees	[4, 8, 16, 64, 128] nodes	CART (2× nodes maximum leaves)
Oblique Decision Trees	[4, 8, 16, 64, 128] nodes	CART (2× nodes maximum leaves)
ReLU MLPs	[2×2, 4×4, 8×8, 16×16] weigths	Adam optimization (500 iterations)

Tableau 9.1 – Summary of baseline policy classes parameters and fitting algorithms (used in Line 15).

Policy classes We consider four policy classes for our baselines. We choose those policy classes because there exist efficient algorithms to fit them with supervised data which is a required step of imitation learning in Line 15. We consider linear policies that have been shown to be able to solve Mujoco tasks [64]. We fit linear policies to expert policies using simple linear (logistic) regressions with scikit-learn [77] default implementation. We also consider decision trees [**cart**] and oblique decision trees [**oblique**]. (Oblique) Decision trees are often considered the most interpretable model class in machine lear-

ning [**mythos**] and reinforcement learning [**IBMDP**, 6, 42, 70]. We train trees using the default CART [**cart**] implementation of scikit-learn with varying numbers of parameters (number of nodes in the tree). We also consider MLPs with ReLU activations [48] with varying number of parameters (total number of weights). This class of policy is often considered the least interpretable and is often used in deep reinforcement learning [47, 21, 49]. We train ReLU MLPs using the default scikit-learn implementation of Adam optimization [53] with 500 iterations. The 15 baseline policy classes that we consider are summarized in Appendix 9.1.

Neural network experts We do not train new deep reinforcement learning agents [71, 90, 47] but rather re-use ones available at the stables-baselines3 zoo [84]. Depending on the environments described next, we choose neural network policies from different deep reinforcement learning agents. Some may argue that during the imitation learning, ReLU MLPs baselines may obtain better performance because they are often from the same class as the expert they imitate unlike trees. But this is not of our concern as we do not benchmark the imitation learning algorithms. Furthermore, it is important to note that not all experts are compatible with all the variants of imitation learning Algorithm 15. Indeed, SAC experts [47] are not compatible with Q-DAgger [6] because it only works for continuous actions; and PPO experts, despite working with discrete actions do not compute a Q-function necessary for the re-weighting in Q-DAgger.

Environments We consider common environments in reinforcement learning research. We consider the classic control tasks from gymnasium [98], MuJoCo robots from [96], and Atari games from [7]. For Atari games, since the state space is frame pixels that can't be interpreted, we use the object-centric version of the games from [27]. In Appendix 9.3 we give the list of environments we consider in our experiments with their state-action spaces as well as a cumulative reward threshold past which an environment is consider "solved".

9.2.2 Ablation study of imitation learning

In this section, we present the results of the expert distillation into smaller policies. For each environment, we fit all the policy classes. To do so, we run different instantiations of Algorithm 15 multiple times with different total sample sizes. For each environment and each imitation learning variant, we summarize the number of times we fit all the baselines to an expert and which expert we use. The number of runs and imitation algorithm variants of Algorithm 15 are summarized in Appendix 9.4. After

running the imitation learnings, we obtain roughly 40000 baseline policies (35000 for classic control, 5000 thousands for MuJoCo and 400 for OCAtari). A dataset with all the baselines measurements is given in the supplementary material.

What is the best imitation algorithm? Even though the focus of our work is to evaluate trained policies, we still provide some insights on the best way to obtain interpretable policies from experts. Using the reinforcement learning evaluation library rliable [2], we plot on Figure 9.7 the interquartile means (IQM, an estimator of the mean robust to outliers) of the baseline policies cumulative rewards averaged over 100 episodes. For each imitation algorithm variant, we aggregate cumulative rewards over environments and policy classes. We normalize the baselines cumulative rewards between expert and random agent cumulative rewards.

The key observation is that for tested environments (Figures 9.7a,9.7b), Behavior Cloning is not an efficient way to train baseline policies compared to DAgger. This is probably because Behavior Cloning trains a student policy to match the expert's actions on states visited by the expert while DAgger trains a student to take the expert's actions on the states visited by the student [88]. An other observation is that the best performing imitation algorithms for MuJoCo (DAgger, Figure 9.7b) and OCAtari (Q-Dagger, Figure 9.7c) obtain baselines that in average cannot match well the performances of the experts. However baseline policies almost always match the expert on simple tasks like classic control (Figure 9.7a).

What is the best policy class in terms of reward? We also wonder if there is a policy class that matches expert performances more often than others across environments. For that we plot performance profiles of the different policy classes obtained with a fixed expert and fixed imitation learning algorithm. In particular, for each environments group we use the baseline policies obtained from the best performing imitation learning algorithm from Figure 9.7. From Figure 9.8 we see that on classic control environments, MLPs tend to perform better than other classes while on OCAtari games, trees tend to perform better than other classes. Now we move on to interpretability evaluation of our programmatic policies.

9.3 Measuring Policy Interpretability

9.3.1 From Policy to Program

In this section, we compute the step inference times, as well as the policy size for both the folded and unfolded variant of each policy obtained for classic control environments with DAgger-100K. To unfold policies, we convert them into Python programs formatted with PEP 8 (comparing other unfolding formats such as ONNX `https://github.com/onnx/onnx` is left to future work). We ensure that all policies operations are performed sequentially and compute the metrics for each policy on 100 episodes using the same CPUs.

Is it necessary to unfold policies to compute interpretability metrics? We see on Figure 9.9 that folded policies of the same class almost always give similar interpretability values (dotted lines) despite having very different number of parameters. Hence, measuring folded policies interpretability would contradict established results from user studies such as, e.g., trees of different sizes have different levels of interpretability [56].

Is there a best policy class in terms of interpretability? User studies from [36, 66, 102] show that decision trees are easier to understand than models involving mathematical equations like oblique trees, linear maps, and MLPs. However, [**mythos**] states that for a human wanting to have a global idea of the inference of a policy, a compact MLP can be more interpretable than a very deep decision tree. In Figure 9.9, we show that inference speed and memory size of programs help us capture those nuances : policy interpretability does not only depend on the policy class but also on the metric choice. Indeed, when we measure interpretability with inference times, we do observe that trees are more interpretable than MLPs. However, when measuring interpretability with policy size, we observe that MLPs can be more interpretable than trees for similar number of parameters. Because there seem to not be a more interpretable policy class across proxy metrics, we will keep studying both metrics at the same time.

9.3.2 Interpretability-performance trade-offs

Now that we trained baseline policies and validated the proposed methodology, we use the latter to tackle open problems in interpretable reinforcement learning. For each environment, we fix the imitation learning algorithm and save the best baseline policy of each class in terms of episodic rewards after unfolding them. Each single Python

Environment Attributes	Importance for Step inference	Importance for Policy size
States dimension	**80.87**	**35.52**
Expert episodes lengths	11.39	9.28
Episode reward of random	2.26	4.75
Expert episode reward	1.51	16.80
Episode reward to solve	1.41	14.26
Actions dimension	1.41	2.02
Expert reward - Solve reward	1.15	17.37

Tableau 9.2 – Environment attributes importance to predict interpretability using either of our metrics.

policy is then **run again on the same dedicated CPU** for 100 new environment episodes (similarly to choosing a classifier with validation score and reporting the test score in the context of supervised learning).

Is it possible to compute interpretable policies for high-dimensional environments? [42] claim that computing an interpretable policy for high dimensional MDPs is difficult since it is similar to program synthesis which is known to be NP-hard [46]. Using our measures of interpretability, we can corroborate this claim. On Figure 9.10, we can indeed observe that some relatively interpretable policies can solve Pong (20 state dimensions) or HalfCheetah (17 state dimensions) while for very high-dimensional environments like Seaquest (180 state dimensions), no baseline can solve the game.

For what environment are there good interpretable policies? We fitted a random forest regressor [14] to predict the interpretability values of our baseline policies using environment attributes. In Table 9.2 we report the importance of each environment attribute when it comes to accurately predicting interpretability scores. We show that as hinted previously, the states dimensions of the environment is determining to predict the interpretability of good policies. Unsurprisingly, expert attributes also influence interpretability : for the environments where there is a positive large gap between expert and threshold rewards, the task could be considered easy and vice-versa.

How does interpretability influence performance? [64, 65] show the existence of linear and tree policies respectively that solve MuJoCo and continuous maze environments respectively; essentially showing that there exist environments for which policies more interpretable than deep neural networks can still compete performance-wise. Our evaluation indeed shows the existence of such environments. On Figure 9.10 we

observe that on, e.g., LunarLander, increasing policy interpretability up to a certain point does not decrease reward. Actually, we can observe that for Pong a minimum level of interpretability is required to solve the game. Indeed, as stated in [35], optimizing interpretability can also be seen as regularizing the policy which can increase generalization capabilities. The key observation is that the policy class achieving the best interpretability-performance trade-off depends on the problem. Indeed, independent of the interpretability proxy metric, we see on Figure 9.10 that for LunarLander it is an MLP that achieves the best trade-off while for Pong it is a tree. Next, we compare our proxies for interpretability with another one ; the verification time of policies used in [6, 5].

9.3.3 Verifying interpretable policies

[5] states that the cost of formally verifying properties of MLPs scales exponentially with the number of the parameters. Hence, they propose to measure interpretability of a policy as the computations required to verify properties of actions given state subspaces, what they call local explainability queries [45]. Before [5], [6] also compared the time to formally verified properties of trees to the time to verify properties of MLPs to evaluate interpretability. In practice, this amounts to passing states and actions bounds and solving the SAT problem of finding a state in the state bounds for which the policy outputs an action in the action bounds. For example, for the LunarLander problem, a query could be to verify if when the y-position of the lander is below some threshold value, i.e, when the lander is close to the ground, there exists a state such that the tested policy would output the action of pushing towards the ground : if the solver outputs "SAT", then there is a risk that the lander crashes.

Designing interesting queries covering all risks is an open problem, hence to evaluate the verification times of our baseline policies, we generate 500 random queries per environment by sampling state and action subspaces uniformily. Out of those queries we only report the verification times of "UNSAT" queries since to verify that, e.g., the lander does not crash we want the queries mentioned above to be "UNSAT". We also only verify instances of ReLU MLPs using [110] for this experiment as verifying decision trees requires a different software [23] for which verification times would not be comparable.

On Figure 9.11, we can observe that verification time decreases exponentially with MLP interpretability, both memory and inference speed, as shown in [5]. This is another good validation of our proposed methodology as well as a motivation to learn interpretable policies.

9.4 Experimental details

In this section we give all the experimental details necessary to reproduce our results.

Classic	**MuJoCo**	**OCAtari**
CartPole (4, 2, **490**)	Swimmer (8, 2, **300**)	Breakout (452, 4, **30**)
LunarLander (8, 4, **200**)	Walker2d (17, 6, **2000**)	Pong (20, 6, **14**)
LunarLanderContinuous (8, 2, **200**)	HalfCheetah (17, 6, **3000**)	SpaceInvaders (188, 6, **680**)
BipedalWalker (24, 4, **250**)	Hopper (11, 3, **2000**)	Seaquest (180, 18, **2000**)
MountainCar (2, 3, **90**)		
MountainCarContinuous (2, 1, **-110**)		
Acrobot (6, 3, **-100**)		
Pendulum (3, 1, **-400**)		

Tableau 9.3 – Summary of considered environments (dimensions of states and number or dimensions of actions, **reward thresholds**). The rewards thresholds are obtained from gymnasium [98]. For OCAtari environments, we choose the thresholds as the minimum between the DQN expert from [84] and the human scores. We also adapt subjectively some thresholds that we find too restrictive especially for MuJoCo (for example, the PPO expert from [84] has 2200 reward on Hopper while the default threshold was 3800).

9.5 All interpretability-performance trade-offs

In this appendix we provide the interpretability-performance trade-offs of all the tested environments. All the measures come from the experiment from Section 9.3.2.

Envs	BC 50K	BC 100K	DAgger 50K	DAgger 100K	Q 50K	Q-DAgger 100K
Classic	50 (PPO, DQN)	50 (PPO, DQN)	50 (PPO, DQN)	50 (PPO, DQN)	50 (DQN)	50 (DQN)
OCAtari	0	0	0	5 (DQN)	0	5 (DQN)
Mujoco	10 (SAC)	10 (SAC)	10 (SAC)	10 (SAC)	0	0

Tableau 9.4 – Repetitions of each imitation learning algorithm on each environment. We specify which deep reinforcement learning agent from the zoo [84] uses as experts in parentheses.

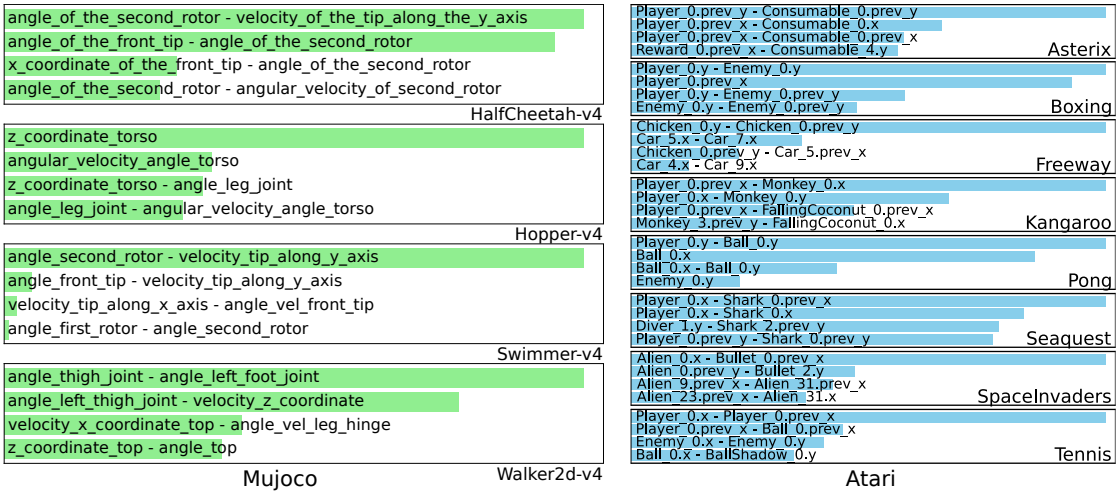


FIGURE 9.2 – **Oracle decision rules are oblique** illustrated on PPO for different state space partitions of the Pong environment. Decisions boundaries are both oblique and parallel.

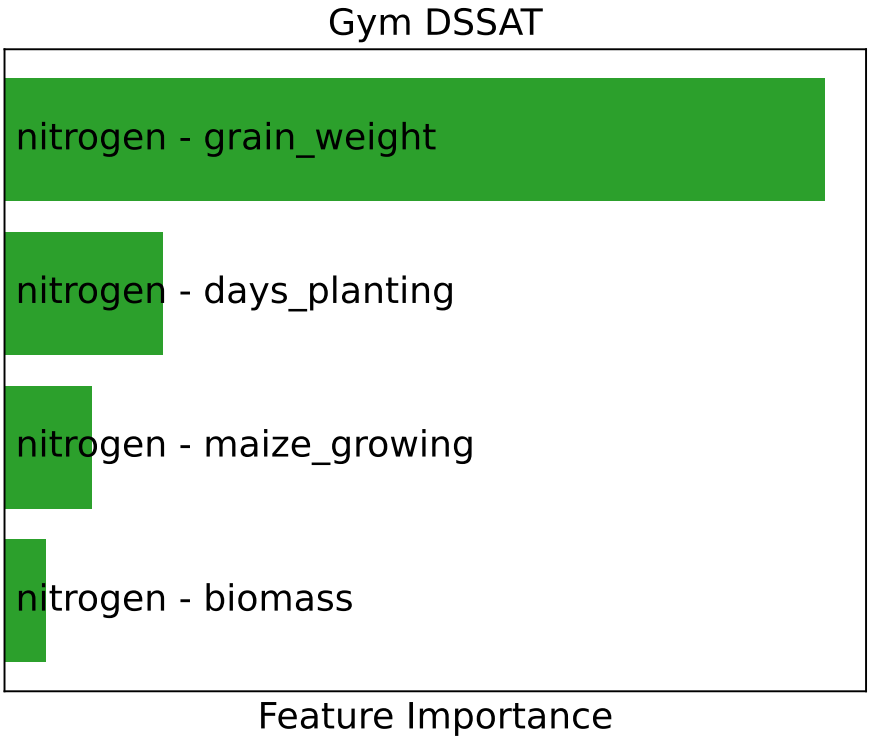


FIGURE 9.3 – **Oracle decision rules are oblique** illustrated on PPO for different state space partitions of the Pong environment. Decisions boundaries are both oblique and parallel.

```python
import gymnasium as gym

env = gym.make("MountainCar")
s, _ = env.reset()
done = False
while not done:
    y0 = 0.969*s[0]-30.830*s[
    y1 = -0.205*s[0]+22.592*s
    y2 = -0.763*s[0]+8.237*s[
    max_val = y0
    action = 0
    if y1 > max_val:
        max_val = y1
        action = 1
    if y2 > max_val:
        action = 2
    s, r, terminated, truncat
    infos = env.step(action)
    done = terminated or trun
```

FIGURE 9.4 – Unfolded linear policy interacting with an environment.

```python
def play(x):
    h_layer_0_0 = 1.238*x[0]+0.971*x[1]
                  +0.430*x[2]+0.933
    h_layer_0_0 = max(0, h_layer_0_0)
    h_layer_0_1 = -1.221*x[0]+1.001
                  *x[1]-0.423*x[2]
                  +0.475
    h_layer_0_1 = max(0, h_layer_0_1)
    h_layer_1_0 = -0.109*h_layer_0_0
                  -0.377*h_layer_0_1
                  +1.694
    h_layer_1_0 = max(0, h_layer_1_0)
    h_layer_1_1 = -3.024*h_layer_0_0
                  -1.421*h_layer_0_1
                  +1.530
    h_layer_1_1 = max(0, h_layer_1_1)

    h_layer_2_0 = -1.790*h_layer_1_0
                  +2.840*h_layer_1_1
                  +0.658
    y_0 = h_layer_2_0
    return [y_0]
```

FIGURE 9.5 – Tiny ReLU MLP for Pendulum.

```python
def play(x):
    if (x[3] - x[4]) <= -0.152:
        if (x[2] - x[3]) <= 0.049:
            return 3
        else:
            if x[5] <= 0.030:
                return 2
            else:
                if (x[2] - x[4]) <=
                    return 3
                else:
                    #Fire main eng
                    return 2
    else:
        if (x[0] - x[6]) <= -0.840
            return 0
        else:
            if (x[0] - x[4]) <= -0
                # Fire right engin
                return 3
            else:
                # Fire left engine
                return 1
```

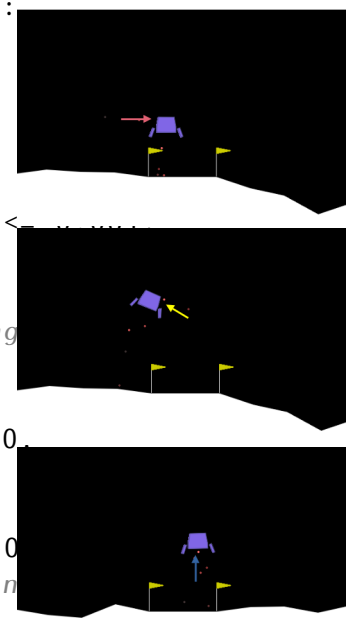
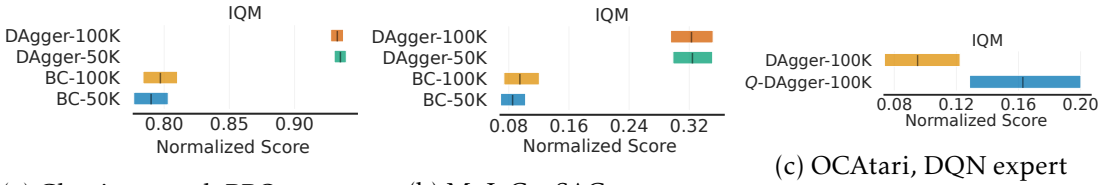


Figure 9.6 – An unfolded oblique tree policy's actions obtaining 250 rewards on Lunar Lander.



(a) Classic control, PPO expert (b) MuJoCo, SAC expert (c) OCAtari, DQN expert

Figure 9.7 – Performance of imitation learning variants of Algorithm 15 on different environments. We plot the 95% stratified bootstrapped confidence intervals around the IQMs.

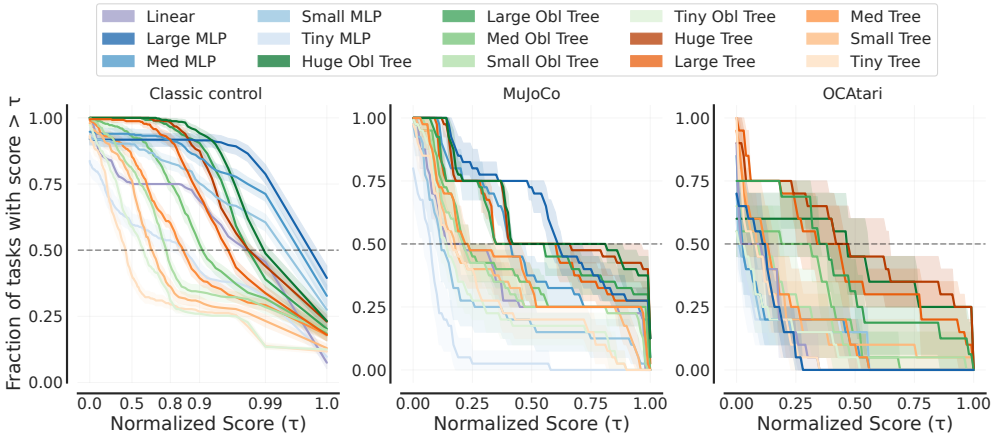


FIGURE 9.8 – Performance profiles of different policy classes on different environments.

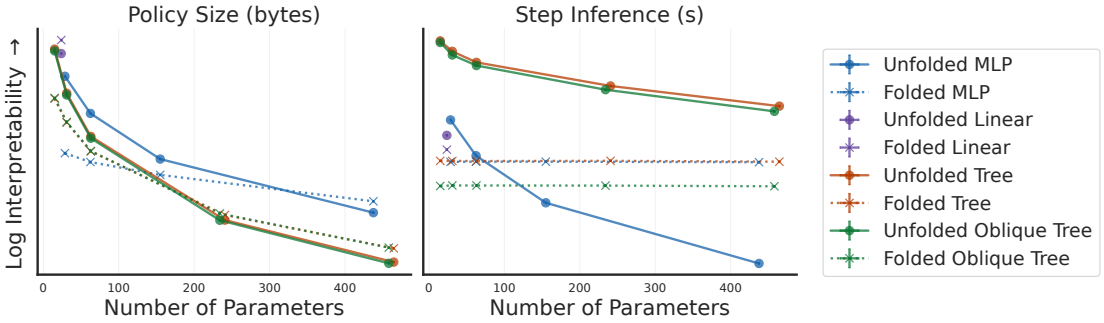


FIGURE 9.9 – Policies interpretability on classic control environments. We plot 95% stratified bootstrapped confidence intervals around means in both axes. In each sub-plot, interpreatbility is measured with either bytes or inference speed.

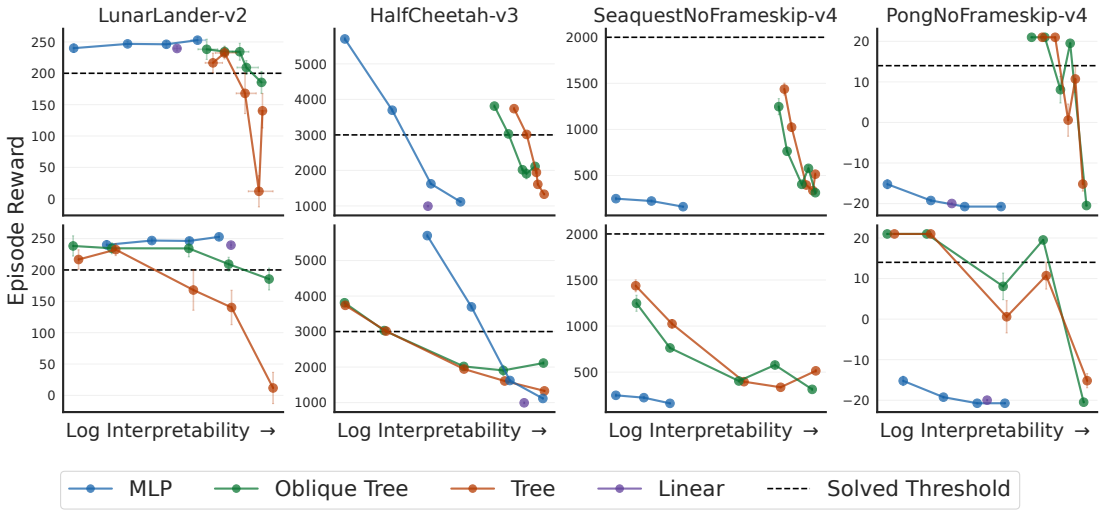


FIGURE 9.10 – Interpretability-Performance trade-offs. Top row, interpretability is measured with step inference times. Bottom row, the interpretability is measured with policy size. We plot 95% bootstrapped confidence intervals around means on both axes.

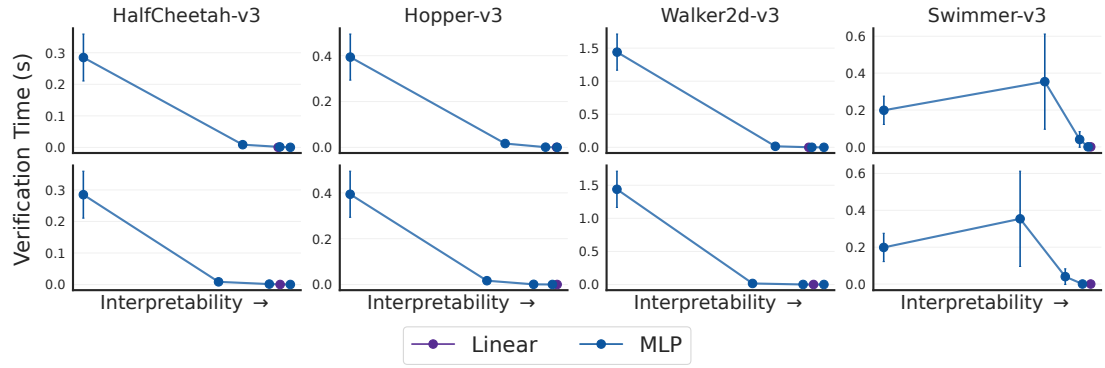


FIGURE 9.11 – Verification time as a function of policy interpretability. Top row, interpretability is measured with step inference times. Bottom row, the interpretability is measured with policy size. We plot 95% confidence intervals around means on both axes.

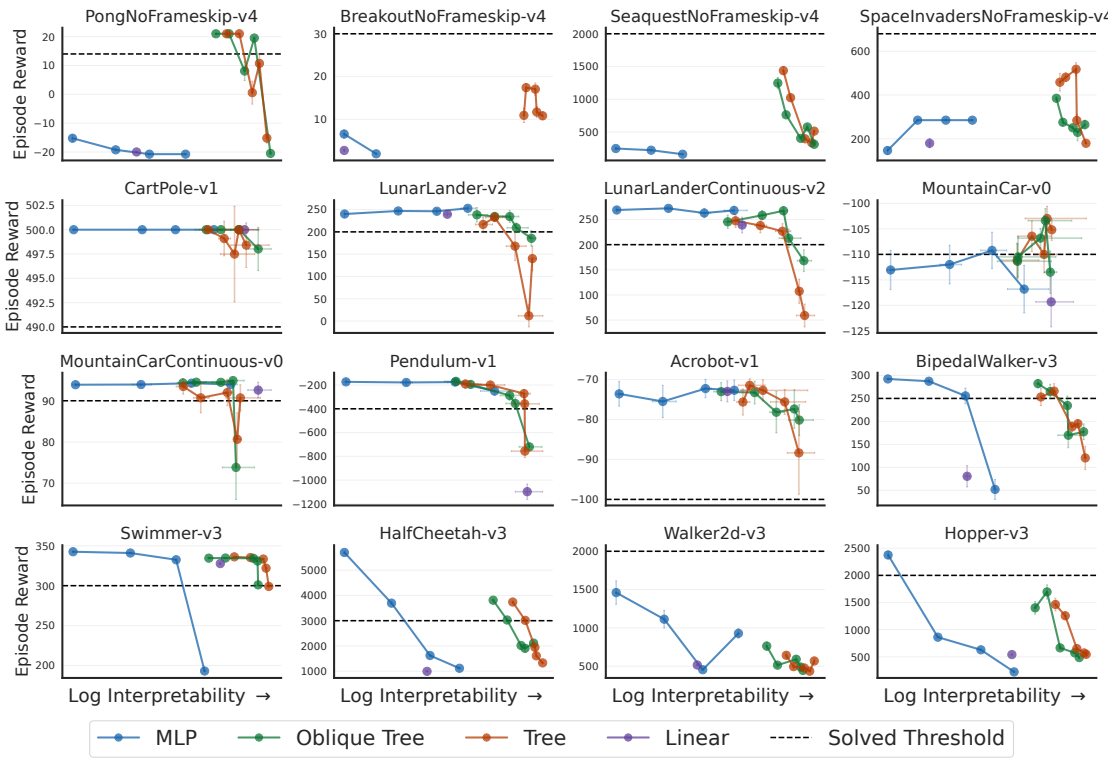


FIGURE 9.12 – Trade-off Cumulative Reward vs. Step Inference Time

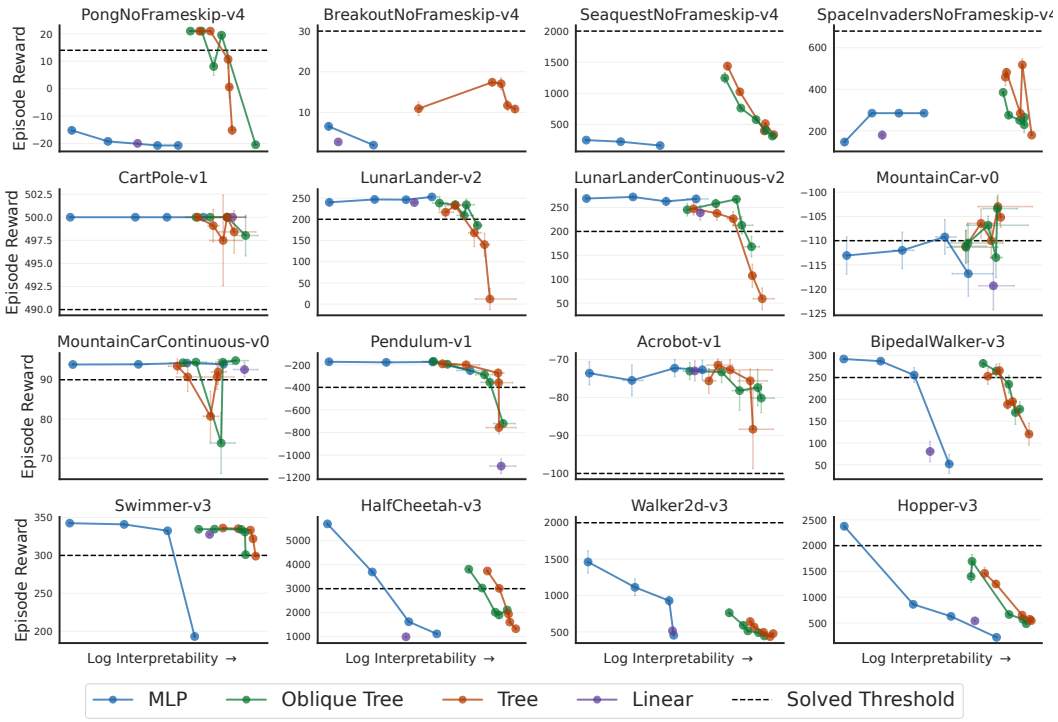


Figure 9.13 – Trade-off Cumulative Reward vs. Episode Inference Time

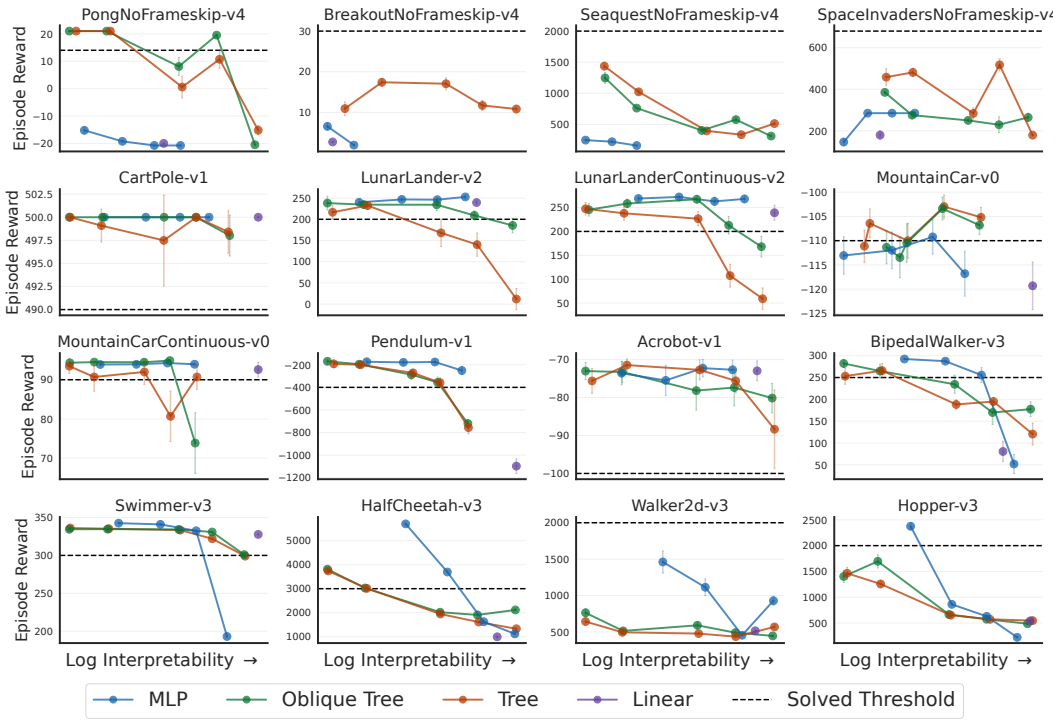


FIGURE 9.14 – Trade-off Cumulative Reward vs. Policy Size

Chapitre 10

Conclusion Imitation

10.1 Limitations and conclusions

We have shown that our proposed methodology provides researchers with a sound way of evaluating policy interpretability. In particular, we have shown that unfolding policies in a common language such as Python is a key component of our methodology to ensure that interpretability depends on the policy complexity (c.f. Figure 9.9). Furthermore, we were able to show that the proxies we use for interpretability leads to similar conclusions from user studies of interpretability or from other empirical evaluations of interpretability (c.f. Figures 9.9, 9.10, and 9.11). Using the proposed methodology, we were able to illustrate the trade-offs between episodic reward and interpretability of policies from different classes (c.f. Figure 9.10) and showed the crucial need of our methodology as there is no better off policy class across tasks and metrics (c.f. Figures 9.8, 9.9, and 9.10).

A nice property of our methodology is that it is independent of the learning algorithm of the interpretable policy. We chose imitation learning but it could have been a random search in the policies parameters space [64]. Furhtermore, there sould be no limitation to use our methodology to evaluate the interpretability of arbitrary compositions of linear policies, trees and oblique trees, and MLPs, such as the hybrid policies from [92]. However, the unfolded version of policies with loops which lengths depend on the state would change between step, hence, the policy size metric value will change during episodes. This is not necessarily a strong limitation but would require more work on the unfolding procedures as well as on defining episodic metrics.

In the future, it would be interesting to compare episodic to averaged measures of

interpretability. Indeed, we additionally show in Appendix 9.13 the interpretability-performance trade-offs using the inference time summed over entire episodes as the measure of interpretability. Even though using episodic inference does not change the trade-offs compared to step inference time, it is important to discuss this nuance in future work since a key difference between supervised learning and reinforcement learning interpretability could be that human operators would read policies multiple times until the end of a decision process. Using episodic metrics for interpretability is not as straightforward as someone would think as for some MDPs, e.g. Acrobot, the episodes lengths depend on the policy. We also did not evaluate the role of sparsity in the interpretability of linear and MLP policies even thought this could greatly influence the inference time. In the future it would be interesting to apply our methodologies to policies obtained with e.g. [93]. Moving away from evaluation, we also believe that our interpretable baselines can be used to train hierarchical agents [111] using our baselines as options. We hope that our methodology as well as the provided baselines will pave the way to a more rigorous science of interpretable reinforcemeent learning.

Conclusion générale

Bibliographie

[1] Fernando Acero et Zhibin Li. « Distilling Reinforcement Learning Policies for Interpretable Robot Locomotion : Gradient Boosting Machines and Symbolic Regression ». In : (2024). URL : https://openreview.net/forum?id=fa3fjH3dEW.

[2] Rishabh Agarwal et al. « Deep Reinforcement Learning at the Edge of the Statistical Precipice ». In : *Advances in Neural Information Processing Systems* (2021).

[3] Sina Aghaei, Andres Gomez et Phebe Vayanos. « Learning Optimal Classification Trees : Strong Max-Flow Formulations ». In : (2020). arXiv : 2002.09142 [stat.ML].

[4] Safa Alver et Doina Precup. « An Attentive Approach for Building Partial Reasoning Agents from Pixels ». In : *Transactions on Machine Learning Research* (2024). ISSN : 2835-8856. URL : https://openreview.net/forum?id=S3FUKFMRw8.

[5] Pablo Barceló et al. « Model interpretability through the lens of computational complexity ». In : *Advances in neural information processing systems* (2020).

[6] Osbert Bastani, Yewen Pu et Armando Solar-Lezama. « Verifiable Reinforcement Learning via Policy Extraction ». In : (2018).

[7] Marc G. Bellemare et al. « The arcade learning environment : an evaluation platform for general agents ». In : *J. Artif. Int. Res.* 47.1 (mai 2013), p. 253-279. ISSN : 1076-9757.

[8] James Bergstra, Daniel Yamins et David Cox. « Making a Science of Model Search : Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures ». In : *Proceedings of the 30th International Conference on Machine Learning*. Proceedings of Machine Learning Research 28.1 (17–19 Jun 2013). Sous la dir. de Sanjoy Dasgupta et David McAllester, p. 115-123. URL : https://proceedings.mlr.press/v28/bergstra13.html.

[9] Dimitris Bertsimas et Jack Dunn. « Optimal classification trees ». In : *Machine Learning* 106 (2017), p. 1039-1082.

[10] Jock Blackard. « Covertype ». In : (1998). DOI : https://doi.org/10.24432/C50K5N.

[11] Guy Blanc et al. « Harnessing the power of choices in decision tree learning ». In : *Advances in Neural Information Processing Systems* 36 (2023), p. 80220-80232.

[12] George Boole. *The Laws of Thought*. Walton, Maberly Macmillan et Co., 1854.

[13] L Breiman et al. *Classification and Regression Trees*. Wadsworth, 1984.

[14] Leo Breiman. « Random forests ». In : *Machine learning* 45 (2001), p. 5-32.

[15] Marco Bressan et al. « A Theory of Interpretable Approximations ». In : *Proceedings of Thirty Seventh Conference on Learning Theory*. Proceedings of Machine Learning Research 247 (2024), p. 648-668.

[16] Lars Buitinck et al. « API design for machine learning software : experiences from the scikit-learn project ». In : *ECML PKDD Workshop : Languages for Data Mining and Machine Learning* (2013), p. 108-122.

[17] Miguel A. Carreira-Perpinan et Pooya Tavallali. « Alternating optimization of decision trees, with application to learning sparse oblique trees ». In : *Advances in Neural Information Processing Systems* 31 (2018). Sous la dir. de S. Bengio et al. url : https://proceedings.neurips.cc/paper_files/paper/2018/file/185c29dc24325934ee377cfda20e414c-Paper.pdf.

[18] Miguel Á Carreira-Perpiñán et Arman Zharmagambetov. « Ensembles of Bagged TAO Trees Consistently Improve over Random Forests, AdaBoost and Gradient Boosting ». In : *Proceedings of the 2020 ACM-IMS on Foundations of Data Science Conference*. FODS '20 (2020), p. 35-46. doi : 10.1145/3412815.3416882. url : https://doi.org/10.1145/3412815.3416882.

[19] Ayman Chaouki, Jesse Read et Albert Bifet. « Branches : A Fast Dynamic Programming and Branch & Bound Algorithm for Optimal Decision Trees ». In : (2024). arXiv : 2406.02175 [cs.LG]. url : https://arxiv.org/abs/2406.02175.

[20] Tianqi Chen et Carlos Guestrin. « XGBoost : A Scalable Tree Boosting System ». In : *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), p. 785-794.

[21] Xinyue Chen et al. « Randomized Ensembled Double Q-Learning : Learning Fast Without a Model ». In : (2021). url : https://openreview.net/forum?id=AY8zfZm0tDd.

[22] Vinícius G Costa et Carlos E Pedreira. « Recent advances in decision trees : An updated survey ». In : *Artificial Intelligence Review* 56 (2023), p. 4765-4800.

[23] Leonardo De Moura et Nikolaj Bjørner. « Z3 : an efficient SMT solver ». In : TACAS'08/ETAPS'08 (2008), p. 337-340.

[24] Jonas Degrave et al. « Magnetic control of tokamak plasmas through deep reinforcement learning ». In : *Nature* 602.7897 (2022), p. 414-419.

[25] Quentin Delfosse et al. « Interpretable and Explainable Logical Policies via Neurally Guided Symbolic Abstraction ». In : *Advances in Neural Information Processing (NeurIPS)* (2023).

[26] Quentin Delfosse et al. « Interpretable Concept Bottlenecks to Align Reinforcement Learning Agents ». In : (2024). URL : https://openreview.net/forum?id=ZC0PSk6Mc6.

[27] Quentin Delfosse et al. « OCAtari : Object-Centric Atari 2600 Reinforcement Learning Environments ». In : *Reinforcement Learning Journal* 1 (2024), p. 400-449.

[28] Emir Demirovic et al. « MurTree : Optimal Decision Trees via Dynamic Programming and Search ». In : *Journal of Machine Learning Research* 23.26 (2022), p. 1-47. URL : http://jmlr.org/papers/v23/20-520.html.

[29] Emir Demirović, Emmanuel Hebrard et Louis Jean. « Blossom : an Anytime Algorithm for Computing Optimal Decision Trees ». In : *Proceedings of the 40th International Conference on Machine Learning*. Proceedings of Machine Learning Research 202 (23–29 Jul 2023). Sous la dir. d'Andreas Krause et al., p. 7533-7562. URL : https://proceedings.mlr.press/v202/demirovic23a.html.

[30] Jacob Devlin et al. « Bert : Pre-training of deep bidirectional transformers for language understanding ». In : *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics : human language technologies, volume 1 (long and short papers)*. 2019, p. 4171-4186.

[31] Finale Doshi-Velez et Been Kim. « Towards A Rigorous Science of Interpretable Machine Learning ». In : (2017). arXiv : 1702.08608 [stat.ML]. URL : https://arxiv.org/abs/1702.08608.

[32] Gabriel Dulac-Arnold et al. « Datum-Wise Classification : A Sequential Approach to Sparsity ». In : *Machine Learning and Knowledge Discovery in Databases* (2011), p. 375-390. ISSN : 1611-3349. DOI : 10.1007/978-3-642-23780-5_34. URL : http://dx.doi.org/10.1007/978-3-642-23780-5_34.

[33] Jan-Niklas Eckardt et al. « Reinforcement learning for precision oncology ». In : *Cancers* 13.18 (2021), p. 4624.

[34] Floriana Esposito et al. « A comparative analysis of methods for pruning decision trees ». In : *IEEE transactions on pattern analysis and machine intelligence* 19.5 (1997), p. 476-491.

[35] Alex A. Freitas. « Comprehensible classification models : a position paper ». In : *SIGKDD Explor. Newsl.* 15.1 (mars 2014), p. 1-10. ISSN : 1931-0145. DOI : 10.1145/2594473.2594475. URL : https://doi.org/10.1145/2594473.2594475.

[36] Alex A. Freitas, Daniela C. Wieser et Rolf Apweiler. « On the Importance of Comprehensible Classification Models for Protein Function Prediction ». In : *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 7.1 (jan. 2010), p. 172-182. ISSN : 1545-5963. DOI : 10.1109/TCBB.2008.47. URL : https://doi.org/10.1109/TCBB.2008.47.

[37] Yoav Freund et Robert E Schapire. « A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting ». In : *Journal of Computer and System Sciences* 55.1 (1997), p. 119-139. issn : 0022-0000. doi : https://doi.org/10.1006/jcss.1997.1504. url : https://www.sciencedirect.com/science/article/pii/S002200009791504X.

[38] Jerome H. Friedman. « Greedy Function Approximation : A Gradient Boosting Machine ». In : *The Annals of Statistics* 29.5 (2001), p. 1189-1232.

[39] Jerome H. Friedman. « Stochastic gradient boosting ». In : *Comput. Stat. Data Anal.* 38.4 (2002), p. 367-378.

[40] Abhinav Garlapati et al. « A Reinforcement Learning Approach to Online Learning of Decision Trees ». In : (2015). arXiv : 1507.06923 [cs.LG]. url : https://arxiv.org/abs/1507.06923.

[41] Romain Gautron. « FApprentissage par renforcement pour l'aide à la conduite des cultures des petits agriculteurs des pays du Sud : vers la maîtrise des risques. » Thèse de doct. Montpellier SupAgro, 2022.

[42] Claire Glanois et al. « A survey on interpretable reinforcement learning ». In : *Machine Learning* (2024), p. 1-44.

[43] Yury Gorishniy et al. « Revisiting deep learning models for tabular data ». In : *Proceedings of the 35th International Conference on Neural Information Processing Systems* (2024).

[44] Léo Grinsztajn, Edouard Oyallon et Gaël Varoquaux. « Why do tree-based models still outperform deep learning on typical tabular data ? » In : *Advances in neural information processing systems* 35 (2022), p. 507-520.

[45] Riccardo Guidotti et al. « A Survey of Methods for Explaining Black Box Models ». In : *ACM Comput. Surv.* 51.5 (août 2018). issn : 0360-0300. doi : 10.1145/3236009. url : https://doi.org/10.1145/3236009.

[46] Sumit Gulwani, Oleksandr Polozov, Rishabh Singh et al. « Program synthesis ». In : *Foundations and Trends® in Programming Languages* 4.1-2 (2017), p. 1-119.

[47] Tuomas Haarnoja et al. « Soft Actor-Critic : Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor ». In : *Proceedings of the 35th International Conference on Machine Learning*. Sous la dir. de Jennifer Dy et Andreas Krause. T. 80. Proceedings of Machine Learning Research. PMLR, oct. 2018, p. 1861-1870. url : https://proceedings.mlr.press/v80/haarnoja18b.html.

[48] Kaiming He et al. « Delving deep into rectifiers : Surpassing human-level performance on imagenet classification ». In : (2015), p. 1026-1034.

[49] Takuya Hiraoka et al. « Dropout Q-Functions for Doubly Efficient Reinforcement Learning ». In : (2022). url : https://openreview.net/forum?id=xCVJMsPv3RT.

[50] Laurent Hyafil et Ronald L. Rivest. « Constructing optimal binary decision trees is NP-complete ». In : *Information Processing Letters* 5.1 (1976), p. 15-17. issn : 0020-0190. doi : https://doi.org/10.1016/0020-0190(76)90095-8. url : https://www.sciencedirect.com/science/article/pii/0020019076900958.

[51] Rasul Kairgeldin et Miguel Á. Carreira-Perpiñán. « Bivariate Decision Trees : Smaller, Interpretable, More Accurate ». In : *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '24 (2024), p. 1336-1347. doi : 10.1145/3637528.3671903. url : https://doi.org/10.1145/3637528.3671903.

[52] Guolin Ke et al. « Lightgbm : A highly efficient gradient boosting decision tree ». In : *Advances in neural information processing systems* 30 (2017), p. 3146-3154.

[53] Diederik P. Kingma et Jimmy Ba. « Adam : A Method for Stochastic Optimization ». In : (2015).

[54] Donald Ervin Knuth. « Finite semifields and projective planes ». Thèse de doct. California Institute of Technology, 1963.

[55] Brent Komer, James Bergstra et Chris Eliasmith. « Hyperopt-Sklearn : Automatic Hyperparameter Configuration for Scikit-Learn ». In : *Proceedings of the 13th Python in Science Conference* (2014). Sous la dir. de Stéfan van der Walt et James Bergstra, p. 32-37. doi : 10.25080/Majora-14bd3278-006.

[56] Nada Lavrač. « Selected techniques for data mining in medicine ». In : *Artificial Intelligence in Medicine* 16.1 (1999). Data Mining Techniques and Applications in Medicine, p. 3-23. issn : 0933-3657. doi : https://doi.org/10.1016/S0933-3657(98)00062-1. url : https://www.sciencedirect.com/science/article/pii/S0933365798000621.

[57] Yann LeCun et al. « Backpropagation applied to handwritten zip code recognition ». In : *Neural computation* 1.4 (1989), p. 541-551.

[58] Edouard Leurent. « Safe and Efficient Reinforcement Learning for Behavioural Planning in Autonomous Driving ». Thèse de doct. Université de Lille, 2020.

[59] Jimmy Lin et al. « Generalized and scalable optimal sparse decision trees ». In : *International Conference on Machine Learning* (2020), p. 6150-6160.

[60] Jacobus van der Linden, Mathijs de Weerdt et Emir Demirović. « Necessary and Sufficient Conditions for Optimal Decision Trees using Dynamic Programming ». In : *Advances in Neural Information Processing Systems* 36 (2023). Sous la dir. d'A. Oh et al., p. 9173-9212.

[61] Jacobus G. M. van der Linden et al. « Optimal or Greedy Decision Trees ? Revisiting their Objectives, Tuning, and Performance ». In : (2024). arXiv : 2409.12788 [cs.LG]. url : https://arxiv.org/abs/2409.12788.

[62] Zachary C. Lipton. « The Mythos of Model Interpretability : In machine learning, the concept of interpretability is both important and slippery. » In : *Queue* 16.3 (2018), p. 31-57.

[63] Wei-Yin Loh. « Fifty years of classification and regression trees ». In : *International Statistical Review* 82.3 (2014), p. 329-348.

[64] Horia Mania, Aurelia Guy et Benjamin Recht. « Simple random search of static linear policies is competitive for reinforcement learning ». In : 31 (2018). Sous la dir. de S. Bengio et al. url : https://proceedings.neurips.cc/paper_files/paper/2018/file/7634ea65a4e6d9041cfd3f7de18e334a-Paper.pdf.

[65] Yishay Mansour, Michal Moshkovitz et Cynthia Rudin. *There is no Accuracy-Interpretability Tradeoff in Reinforcement Learning for Mazes*. 2022. arXiv : 2206.04266 [cs.LG]. url : https://arxiv.org/abs/2206.04266.

[66] David Martens et al. « Performance of classification models from a user perspective ». In : *Decision Support Systems* 51.4 (2011). Recent Advances in Data, Text, and Media Mining & Information Issues in Supply Chain and in Service System Design, p. 782-793. issn : 0167-9236. doi : https://doi.org/10.1016/j.dss.2011.01.013. url : https://www.sciencedirect.com/science/article/pii/S016792361100042X.

[67] Sascha Marton et al. « Mitigating Information Loss in Tree-Based Reinforcement Learning via Direct Optimization ». In : (2025). url : https://openreview.net/forum?id=qpXctF2aLZ.

[68] Rahul Mazumder, Xiang Meng et Haoyue Wang. « Quant-BnB : A Scalable Branch-and-Bound Method for Optimal Decision Trees with Continuous Features ». In : *Proceedings of the 39th International Conference on Machine Learning*. Proceedings of Machine Learning Research 162 (17–23 Jul 2022). Sous la dir. de Kamalika Chaudhuri et al., p. 15255-15277. url : https://proceedings.mlr.press/v162/mazumder22a.html.

[69] Ameet Talwalkar Mehryar Mohri Afshin Rostamizadeh. *Foundations of Machine Learning*. MIT Press, 2012.

[70] Stephanie Milani et al. « Explainable Reinforcement Learning : A Survey and Comparative Review ». In : *ACM Comput. Surv.* 56.7 (avr. 2024). issn : 0360-0300. doi : 10.1145/3616864. url : https://doi.org/10.1145/3616864.

[71] Volodymyr Mnih et al. « Human-level control through deep reinforcement learning ». In : *nature* 518.7540 (2015), p. 529-533.

[72] W Nor Haizan W Mohamed, Mohd Najib Mohd Salleh et Abdul Halim Omar. « A comparative study of reduced error pruning method in decision tree algorithms ». In : *2012 IEEE International conference on control system, computing and engineering* (2012), p. 392-397.

[73] Sreerama Murthy et Steven Salzberg. « Decision tree induction : how effective is the greedy heuristic ? » In : *Proceedings of the First International Conference on Knowledge Discovery and Data Mining* (1995), p. 222-227.

[74] Sreerama Murthy et Steven Salzberg. « Lookahead and Pathology in Decision Tree Induction ». In : *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'95 (1995), p. 1025-1031.

[75] Sreerama K Murthy, Simon Kasif et Steven Salzberg. « A system for induction of oblique decision trees ». In : *Journal of artificial intelligence research* 2 (1994), p. 1-32.

[76] Mohammad Norouzi et al. « Efficient Non-greedy Optimization of Decision Trees ». In : *Advances in Neural Information Processing Systems* 28 (2015). Sous la dir. de C. Cortes et al. url : https://proceedings.neurips.cc/paper_files/paper/2015/file/1579779b98ce9edb98dd85606f2c119d-Paper.pdf.

[77] F. Pedregosa et al. « Scikit-learn : Machine Learning in Python ». In : *Journal of Machine Learning Research* 12 (2011), p. 2825-2830.

[78] Dean A Pomerleau. « Alvinn : An autonomous land vehicle in a neural network ». In : *Advances in neural information processing systems* 1 (1988).

[79] Liudmila Prokhorenkova et al. « CatBoost : unbiased boosting with categorical features ». In : *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18 (2018), p. 6639-6649.

[80] Martin L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[81] Wenjie Qiu et He Zhu. « Programmatic Reinforcement Learning without Oracles ». In : (2022). url : https://openreview.net/forum?id=6Tk2noBdvxt.

[82] J Ross Quinlan. « C4. 5 : Programs for machine learning ». In : *Morgan Kaufmann google schola* 2 (1993), p. 203-228.

[83] J. R. Quinlan. « Induction of Decision Trees ». In : *Mach. Learn.* 1.1 (1986), p. 81-106.

[84] Antonin Raffin. *RL Baselines3 Zoo*. GitHub, 2020.

[85] Antonin Raffin et al. « Stable-Baselines3 : Reliable Reinforcement Learning Implementations ». In : *Journal of Machine Learning Research* 22.268 (2021), p. 1-8.

[86] Marco Tulio Ribeiro, Sameer Singh et Carlos Guestrin. « "Why Should I Trust You ?" : Explaining the Predictions of Any Classifier ». In : KDD '16 (2016), p. 1135-1144. doi : 10.1145/2939672.2939778. url : https://doi.org/10.1145/2939672.2939778.

[87] Frank Rosenblatt. « The perceptron : a probabilistic model for information storage and organization in the brain. » In : *Psychological review* 65.6 (1958), p. 386.

[88] Stéphane Ross, Geoffrey J. Gordon et J. Andrew Bagnell. « A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning ». In : (2010).

[89] Patrick Saux et al. « Development and validation of an interpretable machine learning-based calculator for predicting 5-year weight trajectories after bariatric surgery : a multinational retrospective cohort SOPHIA study ». In : *The Lancet Digital Health* (août 2023). doi : 10.1016/S2589-7500(23)00135-8. url : https://hal.science/hal-04192198.

[90] John Schulman et al. « Proximal policy optimization algorithms ». In : *arXiv preprint arXiv :1707.06347* (2017).

[91] Yijun Shao et al. « Shedding Light on the Black Box : Explaining Deep Neural Network Prediction of Clinical Outcomes ». In : *J. Med. Syst.* 45.1 (jan. 2021). issn : 0148-5598. doi : 10.1007/s10916-020-01701-8. url : https://doi.org/10.1007/s10916-020-01701-8.

[92] Hikaru Shindo et al. « BlendRL : A Framework for Merging Symbolic and Neural Policy Learning ». In : *arXiv* (2025).

[93] Anna Soligo, Pietro Ferraro et David Boyle. *Induced Modularity and Community Detection for Functionally Interpretable Reinforcement Learning*. 2025. arXiv : 2501.17077 [cs.LG]. url : https://arxiv.org/abs/2501.17077.

[94] Gowthami Somepalli et al. « SAINT : Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training ». In : (2021). arXiv : 2106.01342 [cs.LG]. url : https://arxiv.org/abs/2106.01342.

[95] Richard S. Sutton et Andrew G. Barto. *Reinforcement Learning : An Introduction*. Cambridge, MA : The MIT Press, 1998.

[96] Emanuel Todorov, Tom Erez et Yuval Tassa. « MuJoCo : A physics engine for model-based control. » In : (2012), p. 5026-5033.

[97] Nicholay Topin et al. « Iterative bounding mdps : Learning interpretable policies via non-interpretable methods ». In : *Proceedings of the AAAI Conference on Artificial Intelligence* 35 (2021), p. 9923-9931.

[98] Mark Towers et al. « Gymnasium : A Standard Interface for Reinforcement Learning Environments ». In : *arXiv preprint arXiv :2407.17032* (2024).

[99] Dweep Trivedi et al. « Learning to Synthesize Programs as Interpretable and Generalizable Policies ». In : (2021). Sous la dir. d'A. Beygelzimer et al. url : https://openreview.net/forum?id=wP9twkexC3V.

[100] Alan Turing. « Computing Machinery and Intelligence ». In : *Mind* (1950).

[101] Joaquin Vanschoren et al. « OpenML : networked science in machine learning ». In : *SIGKDD Explor. Newsl.* 15.2 (juin 2014), p. 49-60. issn : 1931-0145. doi : 10.1145/2641190.2641198. url : https://doi.org/10.1145/2641190.2641198.

[102] Wouter Verbeke et al. « Building comprehensible customer churn prediction models with advanced rule induction techniques ». In : *Expert Systems with Applications* 38.3 (2011), p. 2354-2364. ISSN : 0957-4174. DOI : https://doi.org/10.1016/j.eswa.2010.08.023. URL : https://www.sciencedirect.com/science/article/pii/S0957417410008067.

[103] Abhinav Verma et al. « Programmatically interpretable reinforcement learning ». In : (2018), p. 5045-5054.

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

[105] Sicco Verwer et Yingqian Zhang. « Learning optimal classification trees using a binary linear program formulation ». In : *Proceedings of the AAAI conference on artificial intelligence* 33 (2019), p. 1625-1632.

[106] Daniël Vos et Sicco Verwer. « Optimal decision tree policies for Markov decision processes ». In : *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. IJCAI '23 (2023). DOI : 10.24963/ijcai.2023/606. URL : https://doi.org/10.24963/ijcai.2023/606.

[107] Daniël Vos et Sicco Verwer. « Optimizing Interpretable Decision Tree Policies for Reinforcement Learning ». In : (2024). arXiv : 2408.11632 [cs.LG]. URL : https://arxiv.org/abs/2408.11632.

[108] Maxime Wabartha et Joelle Pineau. « Piecewise Linear Parametrization of Policies : Towards Interpretable Deep Reinforcement Learning ». In : (2024). URL : https://openreview.net/forum?id=hOMVq57Ce0.

[109] Daniel Whiteson. « HIGGS ». In : (2014). DOI : https://doi.org/10.24432/C5V312.

[110] Haoze Wu et al. *Marabou 2.0 : A Versatile Formal Analyzer of Neural Networks*. 2024. arXiv : 2401.14461 [cs.AI]. URL : https://arxiv.org/abs/2401.14461.

[111] Jesse Zhang, Haonan Yu et Wei Xu. « Hierarchical Reinforcement Learning by Discovering Intrinsic Options ». In : (2021). URL : https://openreview.net/forum?id=r-gPPHEjpmw.

[112] Arman Zharmagambetov, Magzhan Gabidolla et Miguel Ê. Carreira-Perpiñán. « Improved Boosted Regression Forests Through Non-Greedy Tree Optimization ». In : *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), p. 1-8. DOI : 10.1109/IJCNN52387.2021.9534446.

[113] Arman Zharmagambetov et al. « Non-Greedy Algorithms for Decision Tree Optimization : An Experimental Comparison ». In : *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), p. 1-8. DOI : 10.1109/IJCNN52387.2021.9533597.

Annexe A

Programmes informatiques

Les listings suivants sont au cœur de notre travail.

Listing A.1 – Il est l'heure

```
#include <stdio.h>
int heures, minutes, secondes;

/*****************************************************/
/*                                                   */
/*            print_heure                            */
/*                                                   */
/*   But:                                            */
/*       Imprime l'heure                             */
/*                                                   */
/*   Interface:                                      */
/*       Utilise les variables globales              */
/*           heures, minutes, secondes               */
/*                                                   */
/*****************************************************/

void print_heure(void)
{
  printf("Il est %d heure",heures);
  if (heures > 1) printf("s");
  printf(" %d minute",minutes);
  if (minutes > 1) printf("s");
  printf(" %d seconde",secondes);
  if (secondes > 1) printf("s");
  printf("\n");
}
```

Listing A.2 – Factorielle

```
1 int factorielle(int n)
2 {
3   if (n > 2) return n * factorielle(n - 1);
4   return n;
5 }
```

Annexe B

Appendix I

B.1 Tree value computations

Depth-0 decision tree : has only one leaf node that takes a single base 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 :

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

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 base 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 base 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 :

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

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

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

B.2 Hyperparameters

Tableau B.1 – 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

Tableau B.2 – 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

Tableau B.3 – 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

Tableau B.4 – 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

Tableau B.5 – 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

Tableau B.6 – 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

Tableau B.7 – 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

Tableau B.8 – 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 des matières