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 14 juillet 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?

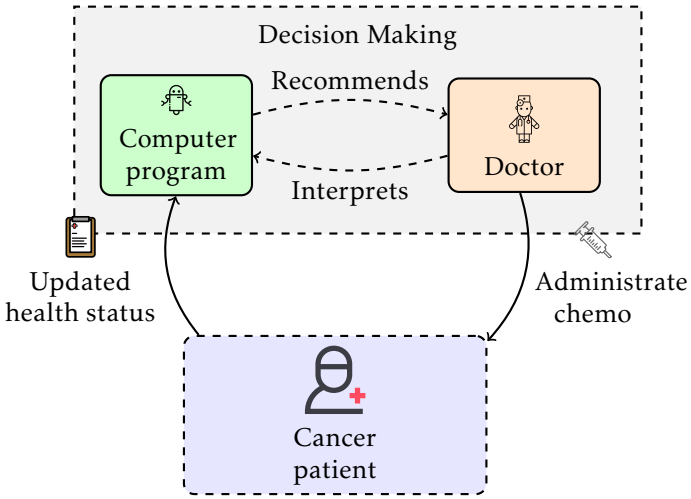


FIGURE 1 – Sequential decision making in cancer treatment. The AI system reacts to the patient's current state (tumor size, blood counts, etc.) and makes a recommendation to the doctor, who administers the chemotherapy to the patient. The patient's state is then updated, and this cycle repeats over time.

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 administrate based on the patient's current health in order to cure [33]. In agriculture, agronomists have to decide when to fertilize next based on the current soil and weather conditions in order to maximize plant growth [41]. In automotive, the auto-pilot system has to decide how to steer the wheel next based on lidar sensors in order to maintain a safe trajectory [58]. Those sequential decision

making processes exhibits key similarities : an agent takes actions based on some current information to achieve some goal.

As computer scientists, we ought to design computer programs [54] that can help humans during those 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 its patient through time and adapts its treatment to the changing health of the patient. In that case, machine learning, and in particular, *reinforcement* learning [95], can be used to teach the program how take decisions that lead to the patient recovery in the future based on how the patient's health changes from one chemo dose to another. Such machine learning algorithms train more and more performing program that make their way into our society to, e.g. identify digits on images [57], control tokamak fusion [24], or write the abstract of a scientific article [30].

However, the key problematic behind this manuscript is that those programs computations 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]

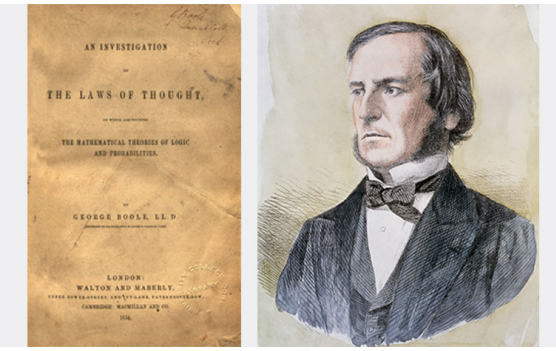
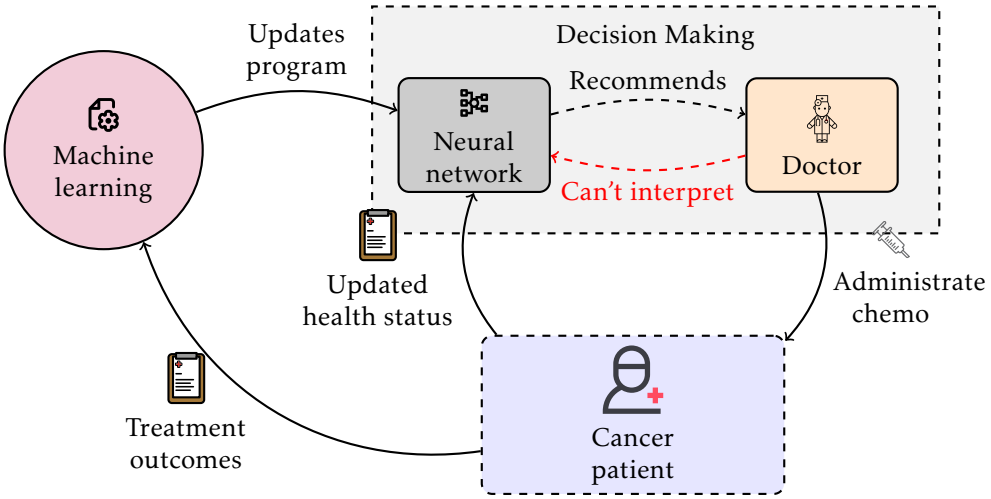


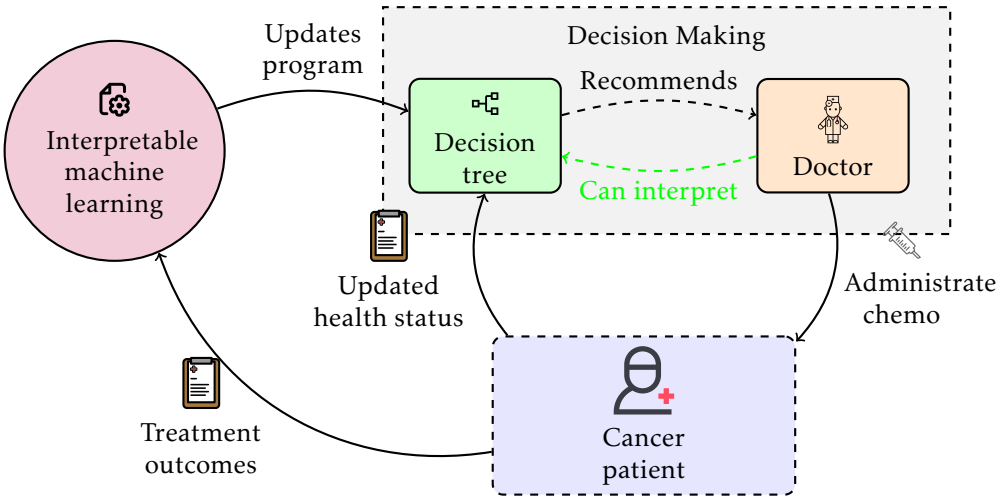
Figure 2 – British logician and philosopher George Boole (1815-1864) next to its book *The Laws of Thoughts* (1854) that is the oldest known record of the word "interpretability".

What is remarkable is that the supposedly first recorded occurrence of "interpretability" was in the context of computer science. 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 netowrks [87], that learn complex unintelligible combinations of inputs (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



(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.

in principle to help with cancer treatment. In truth this should be prohibited without some kind of transparency in the program's recommendation : why did the program recommended such 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 which recommendations are interpretable.

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

What are existing approaches for learning interpretable programs ?

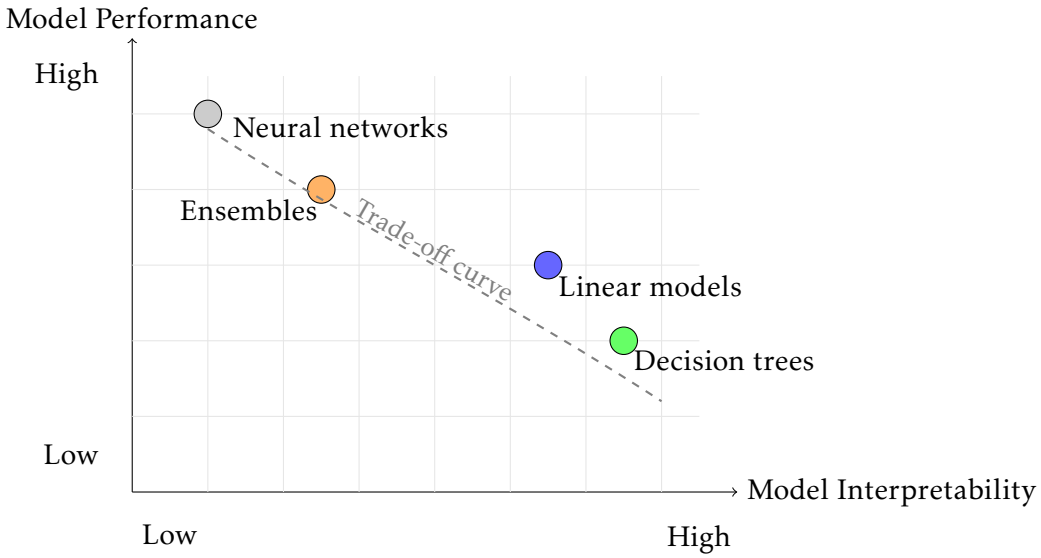


Figure 4 – The interpretability-performance trade-off in machine learning. Different program classes are positioned according to their typical interpretability and performance characteristics. The dashed line illustrates the general trade-off between these two properties.

Interpretable machine learning provides either local or global explanations [42]. Global methods output a program which all recommendations can be interpreted without additional computations, e.g. a decision tree [13]. On the other hand, local

methods require additional computations but are agnostic to the program class : they can give an *approximate* interpretation of e.g. neural networks recommendations. In Figure 4 we present the popular trade-off between interpretability and performance of different program classes.

The most famous local explanation algorithm is LIME (Local Interpretable Model-agnostic Explanations) [86]. Given a program class, 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 whole program per recommendation that needs to be interpreted ; this is a lot of computations. (Figure) HEATMAPS-FEATURE IMPORTANCE : key message is that Global => Local and Local =$\tilde{>}$ Global.

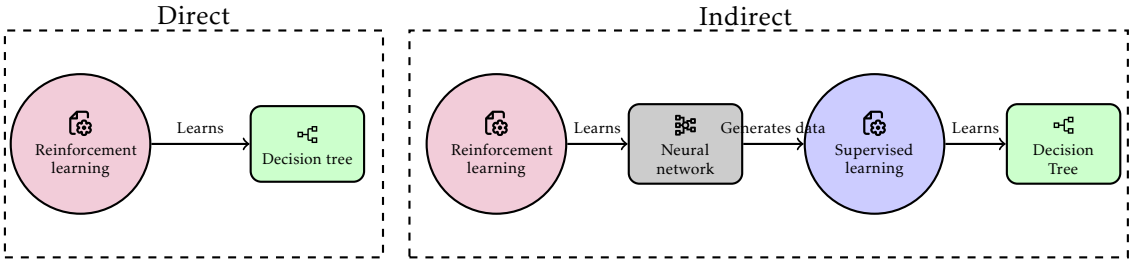


Figure 5 – Comparison of direct and indirect approaches for learning interpretable policies in sequential decision making

Global approaches are either direct or indirect [70]. Direct algorithms, such as decision tree induction [13], are algorithms that directly search a space of interpretable programs (see Figure 4). One of the key challenges that motivates this thesis is that decision tree induction is only defined for supervised learning but not for reinforcement learning. It means that to directly learn computer programs for interpretable sequential decision making, one has to design completely new algorithms. What most existing research have focused on so far, is to work around this confinement of decision tree induction to supervised learning and develop indirect methods. Indirect methods for interpretable sequential decision making–sometimes called post-hoc–start with the reinforcement learning of a non-interpretable computer program, e.g., deep reinforcement learning of a neural network, and then use supervised learning of an interpretable model with the objective to emulate the non-interpretable program. This approach is called behavior cloning or imitation learning [78, 88] and many, if not all, work on interpretable sequential decision making use this indirect approach[6, 103]. Figure 5

illustrates the key difference between these two approaches.

Researchers just recently started to study the advantage of direct over indirect learning of interpretable programs [97, 99]. In short, the motivation behind developing direct methods is to have the interpretable program optimized to solve your actual goal, e.g. patient treatment, while indirect methods learn an interpretable program that is optimized to match the behaviour of a non-interpretable model that was itself optimized to solve your goal. There is no guarantee that optimizing this indirect objective yields the "best" interpretability-performance trade-offs. Hence, the ideal solution to interpretable sequential decision making would be to have global direct algorithms.

Other related works

Explainability

Misalignement

Programmatic RL

Causality

Mechanistic Interpretability in the era of large language models

Next, we present the outline of this thesis.

Outline of the Thesis

In this thesis we take a stab at the difficult task of designing global direct algorithms for interpretable sequential decision making. In the first part of the manuscript we will present a mathematical formalism for the reinforcement learning of decision trees for sequential decision making. In particular we will show that indeed the direct approach can yield better trees than the indirect one. However the unfortunate key result from this opening part is that a good direct method cannot find decision trees for sequential decision making that are not *very* easy. Fortunately for us, in the second part of this manuscript, we show that some of these easy instances of interpretable sequential decision making tasks can be made *non*-sequential giving rise to a whole new decision tree induction algorithm for supervised learning. In particular, we thoroughly benchmark our new decision tree induction algorithm and claim the state-of-the-art for decision tree induction. Finally, after heavily studying decision trees and direct methods, we

will leverage the diversity and simplicity of *indirect* methods to compare other model classes of programs 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

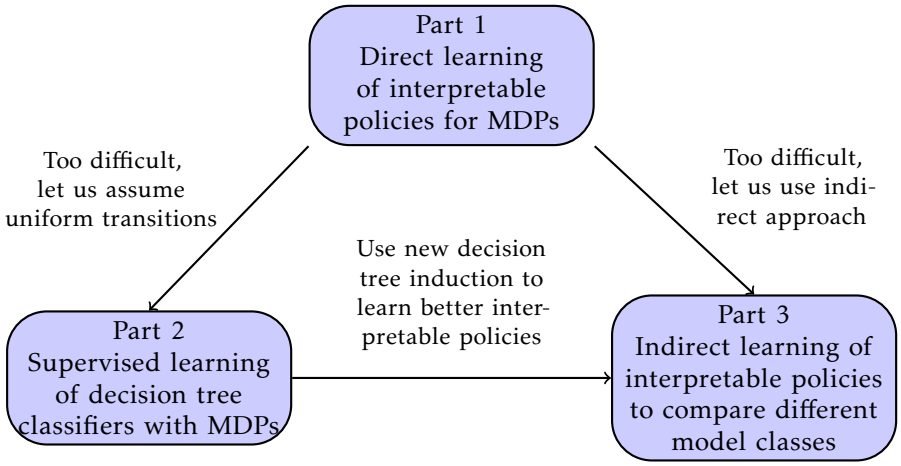


FIGURE 6 – Thesis structure showing the progression from direct reinforcement learning of decision tree policies (Chapter 1) to simplified approaches : supervised learning with uniform transitions (Chapter 2) and indirect learning methods (Chapter 3).

Technical Preliminaries

What are decision trees?

As mentioned earlier, as opposed to neural networks, decision trees are supposedly very interpretable because they only apply boolean operations on the program input without relying on internal complex representations.

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.*

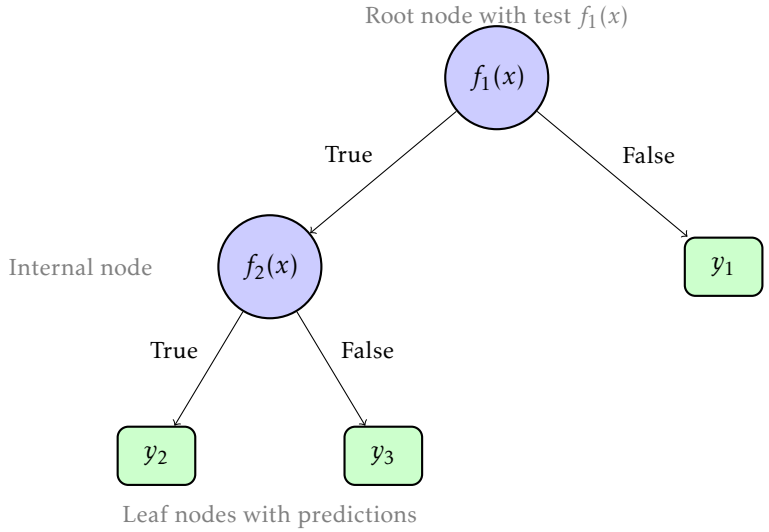


FIGURE 7 – A generic decision tree structure. Internal nodes contain test functions $f_v(x) : \mathcal{X} \to \{0, 1\}$ that map input features to boolean values. Edges represent the outcomes of these tests (True/False), and leaf nodes contain predictions $y_\ell \in \mathcal{Y}$. For any input x, the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

— *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?

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 $f_v(x) = \mathbb{I}[x[feature] \leq threshold]$ 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.

Algorithme 1 : CART Algorithm for Decision Tree Learning

Data : Training data (X, y) where $X \in \mathbb{R}^{n \times d}$ 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* $f \in \{1, 2, \ldots, d\}$ **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)

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

Markov decision processes/problems

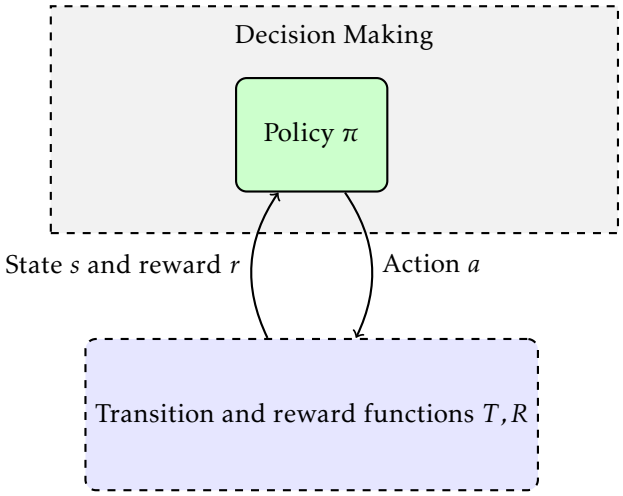


FIGURE 9 – Markov decision process

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 its goal. At every timestep, the agent observes its current state, e.g. a patient weight and tumor size, and takes an action, e.g. injects a certain amount of chemotherapy. When doing a certain action in a certain state, the agent gets a reward that helps it evaluate the quality of its action with respect to its goal, e.g., the tumor size decrease when the agent has to cure cancer. Finally, the agent is provided with 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 as follows.

Definition 2 (Markov decision process)**.** *An MDP is a tuple $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$ where :*

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

Now we can also model the "goal" of the agent. Informally, the goal of an agent is to behave such that it gets as much reward as it can over time. For example, in the cancer treatment case, the best reward the agent can get is to completely get rid of the patient's tumor after some time. Furthermore, we want our agent to prefer behaviour that gets rid of the patient's tumor as fast as possible. We can formally model the agent's goal as an optimization problem as follows.

Definition 3 (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 : $\pi^\star = \underset{\pi}{\text{argmax}} J(\pi)$ For the rest of this text, we will use an abuse of notation and denote both a Markov decision process and the associated Markov decision problem by MDP.

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 4 (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 :

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

where $T^{s'}(s, \pi(s))$ is the probability of transitioning to state s' when taking action $\pi(s)$ in state s.

Definition 5 (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 6 (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 of state s when taking action a and then following the optimal policy : $V^{\pi^{\star}}(s)$.*

$$Q^{\star}(s, a) = Q^{\pi^{\star}}(s) = R(s, a) + \gamma \sum_{s' \in S} V^{\star}(s')$$

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).

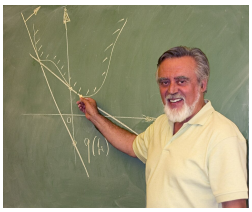
More realistically, neither the transition kernel T nor the reward function R of the MDP are known, e.g., the doctor can't **know** how the tumor and the patient health will change after a dose of chemotherapy, it can only **observe** the change. This distinction between the information available to the agent is paralleled with the distinction between dynamic programming and reinforcement learning (RL) that we describe next.

Reinforcement learning of approximate solutions to MDPs

Reinforcement learning algorithms popularized by Richard Sutton (Figure 10) (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 :

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



(a) D. Bertsekas

(b) M.L. Puterman

(c) A. Barto

(d) R. Sutton

Figure 10 – The godfathers of sequential decision making. Andrew Barto and Richard Sutton are the ACM Turing Prize 2024 laureate and share an advisor advisee relationship.

value-based (cite) and policy gradient (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 8.

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 timestep 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

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.

Imitation learning, the baseline (indirect) global interpretable reinforcement learning method

Algorithme 5 : 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 $\hat{\pi}_N$;

Première partie

A Difficult Problem : Direct Interpretable Reinforcement Learning

Chapitre 1

A Formal Framework for the Reinforcement Learning of Decision Tree Policies

1.1 Learning Decision Tree policies

Deicision tree policies offer transparency over neural network policies (cite). One can attribute an importance measure to each feature of the state for the deicison of a tree policy which is harder to do for neural networks (cite).

Recently, algorithms have been developed to return decision tree policies for an MDP. Those algorithms, like any interpretable machine learning method, are either direct or indrect (cite). We propose an additional distinction amongst the direct methods : algorithms learning parametric trees and algorithms learning 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. In the next section we present, to the best of our knowledge, the only direct approach to learn non-parametric decision tree policies for MDPs ; Iterative Bounding Markove Decision Processes (cite).

Other more specialized approaches deal with tree policies either for specific MDPs like maze (cite) or for very small depth (cite) or when the MDP model is known (cite).

1.2 Iterative Bounding Markov Decision Processes

In 2021, Topin et. al. introduced Iterative Bouding Markov Decision Processes (IBMDPs) with the promise of shifting the challenges of non-parametric decision tree policy learning in the problem formulation rather than in the design of specialized reinforcement learning algorithms. Given a base 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, additinal reward signal, and additional transition kernel. Authors showed that certain IBMDP policies, that can be learned with RL, are equivalent to decision tree policies for the base MDP.

1.2.1 Formalism

The key thing to know about IBMDPs is that they are, as their name suggests, MDPs. Hence they inherit all their properties such as existence of a deterministic optimal Markovian policy. The states in an IBMDP are concatenations of 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 such *information gathering* actions; the reward is an arbitrary value supposed to trade-off between performance and interpretability.

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

Definition 7 (Iterative Bounding Markov decision process)**.** *Given a factored (cite) MDP $\mathcal{M}$ (cite), that is, a tuple $\langle S, A, R, T, T_0 \rangle$ with $S \subsetneq \mathbb{R}^n$, an Iterative Bouding MDP $\mathcal{M}_{\rfloor}$ is a tuple :*

$$\langle \overbrace{S, O}^{\text{State space}} , \underbrace{A, A_{info}}_{\text{Action space}} , \overbrace{R, \zeta}^{\text{Reward function}} , \underbrace{T_{info}, T, T_0}_{\text{Transition kernels}} \rangle$$

- *S the base MDP state space should be of the form $S = [L_1, U_1] \times \cdots \times [L_n, U_n]$ with $\infty < L_i \leq U_i < \infty \forall 1 \leq i \leq n$.*
- *O are the observations in an IBMDP. They are partial information about the values of base MDP states : $O \subsetneq S^2 = [L_1, U_1] \times \cdots \times [L_n, U_n] \times [L_1, U_1] \times \cdots \times [L_n, U_n]$. So the complete IBMDP state space is $(S, O) = S \times O$ the concatenations of states and observations.*
- *A are the actions of the base MDP.*
- *A_{info} are added information gathering actions (AIGs) 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 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 a information gathering action.*
- *$T_{info} : S \times O \times (A_{info} \cup A) \to \Delta(S \times O)$ is the transition kernel of IBMDPs. Given the current observation $o_t = (L'_1, U'_1, \ldots, L'_n, U'_n) \in O$ and the current state is $s_t = (s_1, s_2, \ldots, s_n)$ if an AIG $\langle i, v \rangle$ is taken, only the bounds in the observation change :*

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

If a base action $a \in A$ is taken, o_{t+1} is reset to the default state bounds $(L_1, U_1, \ldots, L_n, U_n)$

Algorithme 6 : Extract a Decision Tree Policy from an IBMDP policy π, beginning traversal from *obs*.

Data : IBMDP policy π and observation $o = (L'_1, U'_1, \dots, L'_n, U'_n)$
Result : Decision tree policy extracted from π
Function `Subtree_From_Policy(`o, π`)` **:**
  $a \leftarrow \pi(o)$
  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`
    $o_L \leftarrow o$; $o_R \leftarrow o$
    $o_L \leftarrow (L'_1, U'_1, \dots, L'_i, v, \dots, L'_n, U'_n)$; $o_R \leftarrow (L'_1, U'_1, \dots, v, U'_i, \dots, L'_n, U'_n)$
    $child_L \leftarrow$ Subtree_From_Policy(o_L, π)
    $child_R \leftarrow$ Subtree_From_Policy(o_R, π)
    return Internal_Node(feature : i, value : v, children : $(child_L, child_R)$)
  end
return Subtree_From_Policy(obs, π)

and the base state changes according to the base MDP transitition kernel : $s_{t+1} \sim T(s, a)$. At initialization, the base part of the IBMDP states is drawn from T_0 and the observation is set always set to $(L_1, U_1, \dots, L_n, U_n)$. The overall IBMDP transitions are given by either T_{info}, which i fully deterministic, if an AIG is played, and by the base MDP's transition kernel otherwise.

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

1.2.2 From Policies to Trees

One can notice that *information gathering* actions resemble the Boolean functions that make up internal decision tree nodes (cite). Indeed, an agent evolving in an IBMDP essentially builds a tree by taking sequences of AIGs (internal nodes) and then a base action (leaf node) and repeats this process over time. However not all IBMDP policies are decision tree policies. In particular, only deterministic policies depending solely on the observation part of the IBMDP states $\pi : O \to A$ are decision tree policies $\pi_{\mathcal{T}} S \to A$ for the base MDP (cite). Algorithm (cite) from (cite) extracts a decision tree policy for the base MDP from a determenistic partailly observable IBMDP policy.

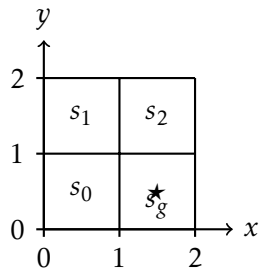


Figure 1.1 – Grid MDP

1.2.3 Didactic example

We now present a didactic example of how some policies in IBMDPs correspond to a decision tree policy. Suppose a factored MDP representing a grid world with 4 cells (cite). The state space is $S = \{(0.5, 0.5), (0.5, 1.5), (1.5, 1.5), (1.5, 0.5)\} \subsetneq [0, 2] \times [0, 2]$. The actions space 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 $s_g = (1.5, 0.5)$ which is an absorbing state : once in this state, you stay there forever.

In Figure (cite), we present how trajectories in IBMDP correspond to a decision tree policy.

1.2.4 Partially Observable IBMDPs

Now we know that to find a decision tree policy for a given MPD $\mathcal{M}$ satisfying the definition (cite); one has to find a partially observable–sometimes called *memoryless* or *reactive*–deterministic policy for an IBMDP $\mathcal{M}_{IB}$. Such problems are classical instances of Partially Observable Markov Decision Processes (POMDPs) (cite). This connection with POMDP was not done by the authors of IBMDPs.

Definition 8 (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 base definition of MDPs).*
- *A is a finite set of actions (like in the base definition of MDPs).*
- *O is a set of observations.*
- *$T : X \times A \rightarrow \Delta X$ is the transition kernal, where $T(s, a, x') = P(x'|x, a)$ is the probability of transitioning to state x' when taking action a in state x*
- *T_0 : is the intial distribution over states.*

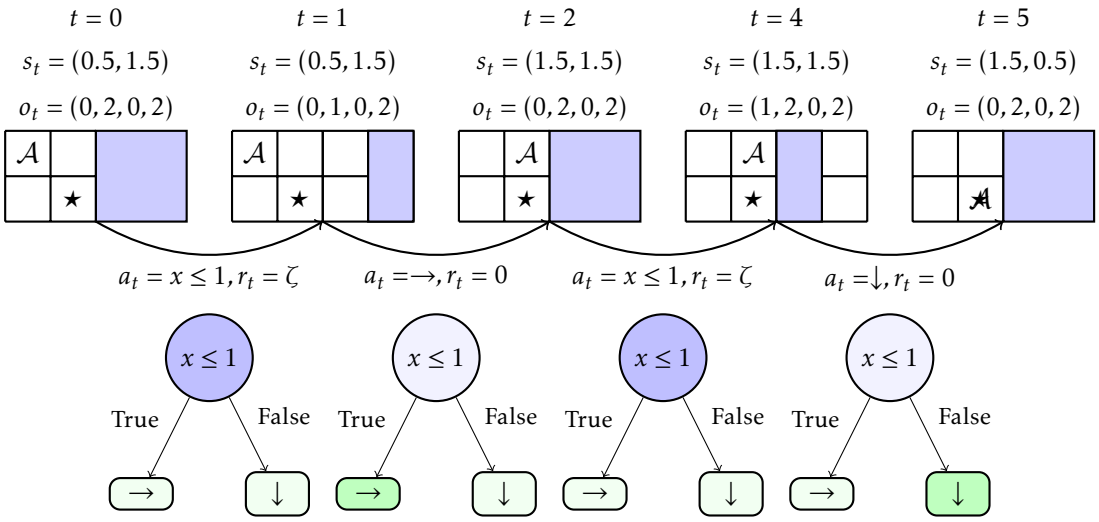


FIGURE 1.2 – An IBMDP trajectory when the base MDP is 2×2 grid world, and equivalent decision tree policy traversal. At $t = 0$, the agent is in $s_1 = (0.5, 1.5)$. The initial observation is always the base MDP state bounds, here $(0, 2, 0, 2)$ because the base states are in $[0, 2] \times [0, 2]$. The agent takes an information gathering action in A_{info} that tests the feature x of the states against the value 1. For that, the agent receives 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 $o_t = (0, 1, 0, 2)$ and the base state 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 $s_t = (1.5, 1.5)$. And the trajectory continues like this until the agent reaches the absorbing state $s_g = (1.5, 0.5)$

— $\Omega : X \to \Delta O$ *is the observation kernel, where* $\Omega(x', a, o) = P(o|x', a)$ *is the probability of observing* o *in state* x
— $R : X \times A \to \mathbb{R}$ *is the reward function, where* $R(x, a)$ *is the immediate reward for taking action* a *in state* 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 9 (Partially Observable Iterative Bounding Markov Decision Processes)**.** *a Partially Observable Iterative Bounding Markov Decision Process (POIBMDP) is a an IBMDP*

extended with an observation kernel

$$\langle \overbrace{S, \underbrace{O}_{Observations}}^{fully\ observable\ states\ X}, \underbrace{A, A_{info}}_{A}, \overbrace{R, \zeta}^{R}, \underbrace{T_{info}, T, T_0}_{T,T_0}, \Omega \rangle$$

The sole specifity of POIBMDPs compared to the general definition of POMDPs, is that $\Omega(o|(s,o'))$, the probability of observing o in (s,o'), is $1_{o=o'}$. This particular instance of POMDPs with observations being some indices of the fully-observable states has other names in the litterature : Mixed Observability MDPs (cite), Block 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 10 (Partial observable value function)**.** *The expected cumulative discounted reward of a deterministic partially observable policy $\pi : O \to A$ 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 :

$$\pi^\star = \underset{\pi}{\text{argmax}} J(\pi) = \underset{\pi}{\text{argmax}} V^\pi(o_0) \tag{1.1}$$

With $\pi : O \to A$. There is no expectation over possible initial observation in the above objective function as there is only one initial observation in a POIBMDP : $o_0 = (L_1, U_1, \ldots, L_n, U_n)$.

This particular problem of learning deterministic partially observable policies for POMDPs has been studied in the works of Littman, Singh and Jordan : (cite). In (cite)

atuhors give som intution behind why the above optimization problem is hard. For example, the optimal partially observable policy can be stochastic (cite precise section), hence policy gradient algoriothms (cite) are to avoid. Furthermore, the optimal deterministic patially observable policy might not maximize all the value of all observations simulataneously (cite precise section) which makes difficult the use 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 the next chapter, we use reinforcement learning to train decision tree policies for MDPs by seeking deterministic partially observable policies for POIBMDPs (cite).

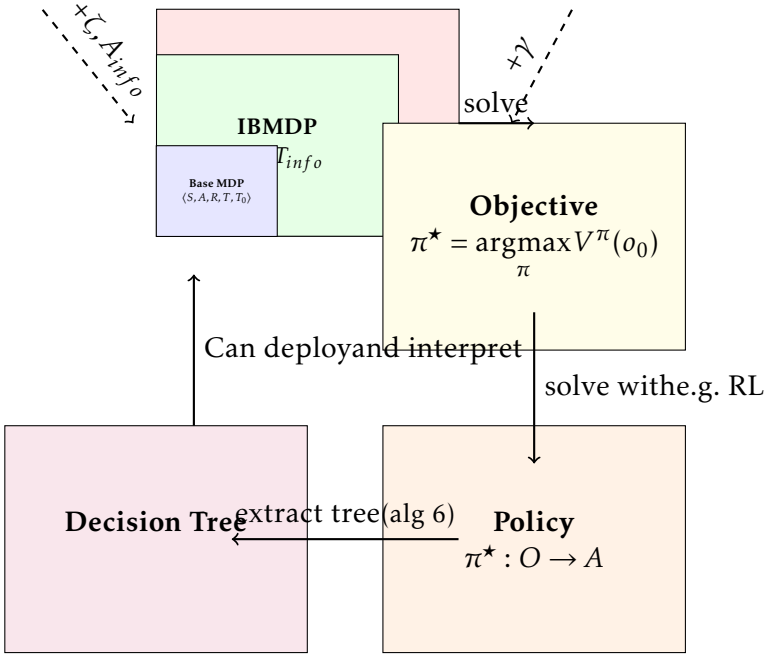


FIGURE 1.3 – Nested structure of decision processes : Base MDP (innermost), IBMDP (middle), and POIBMDP (outermost).

Chapitre 2

The Limits of Direct Reinforcement Learning of Decision Tree Policies

2.1 Grid World tabular solutions

In this section, for the sake of example, we assume that a dictionary with 4 key-value pais is less interpretable than a depth-1 decision tree. In Figure 2.1, we illustrate *a*–there are others–tabular optimal policy for the grid world MDP described in the previous section, i.e. that maximizes the objective (cite). In any starting state, the agent will move towards the abosrbing state with positive reward.

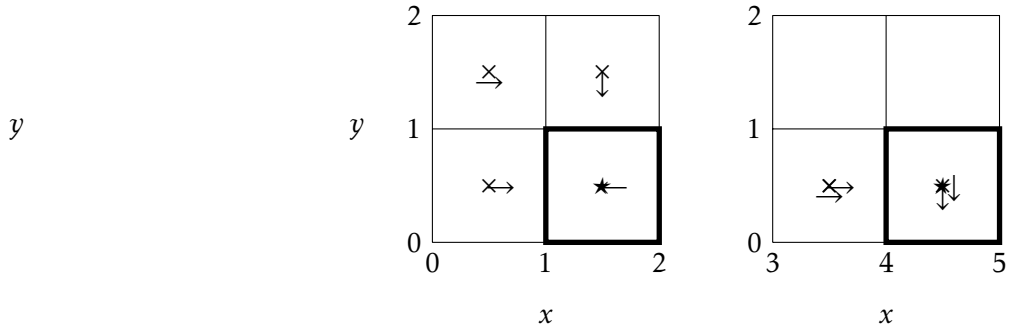


FIGURE 2.1 – Optimal tabular policies for the grid world.

For this grid world, there exist *optimal* decision tree policies. In particular Tree 4 and 6 are have very good interpretability-performance trade-offs (smallest tree that can get the maximum MDP reward). In the rest of this section we shall attempt to *learn* those trees from data.

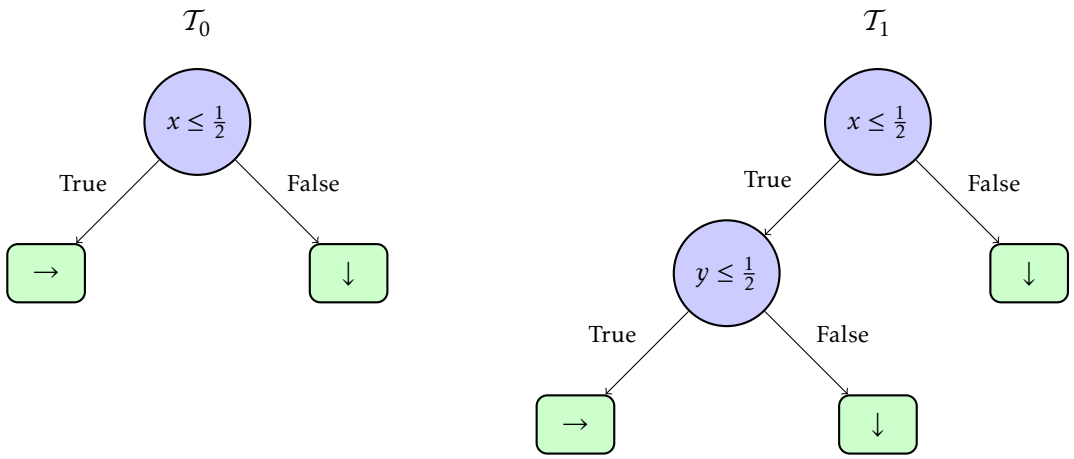


FIGURE 2.2 – Some optimal decision tree policies.

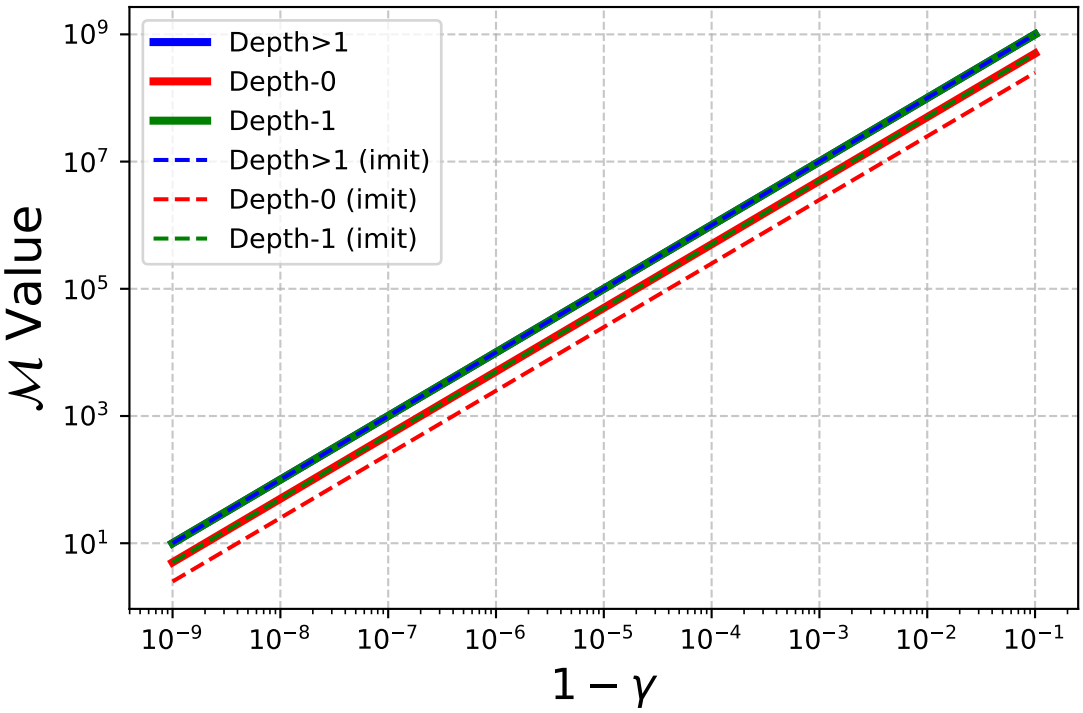


FIGURE 2.3 – MDP objective values.

2.2 Method

2.2.1 Getting the values of Trees

Let us compute the objective value (cite) for the trees presented above. We will identify the range of ζ values–the interpretabiliy penalty–for which the depth-1 tree is optimal. Let us compute the objective values of the most rewarded tree for each tree structure.

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 :

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

Depth-1 decision tree : has one root node that tests $x \leq 0.5$ (respectively $y \leq 0.5$) and two leaf nodes $\rightarrow$ and $\downarrow$. In the goal state $G = (1, 0)$, the depth-1 decision tree policy

cycles between taking an information gathering action $x \leq 0.5$ and moving down to get a positive reward for which it gets the returns :

$$\begin{aligned} V_G^{\mathcal{T}_1} &= \zeta + \gamma + \gamma^2\zeta + \gamma^3 \dots \\ &= \sum_{t=0}^{\infty} \gamma^{2t}\zeta + \sum_{t=0}^{\infty} \gamma^{2t+1} \\ &= \frac{\zeta + \gamma}{1-\gamma^2} \end{aligned}$$

In states $S_0 = (0,0)$ and $S_2 = (1,1)$, the value of the depth-1 decision tree policy is the return when taking one information gathering action $x \leq 0.5$, then moving right or down, then following the policy from the goal state G :

$$\begin{aligned} V_{S_0}^{\mathcal{T}_1} &= \zeta + \gamma 0 + \gamma^2 V_G^{\mathcal{T}_1} \\ &= \zeta + \gamma^2 V_G^{\mathcal{T}_1} \\ &= V_{S_2}^{\mathcal{T}_1} \end{aligned}$$

Similarly the value of the depth-1 decision tree policy in state $S_1 = (0,1)$ is the value of taking one information gathering action then moving right to S_2 then following the policy in S_2 :

$$\begin{aligned} V_{S_1}^{\mathcal{T}_1} &= \zeta + \gamma 0 + \gamma^2 V_{S_2}^{\mathcal{T}_1} \\ &= \zeta + \gamma^2 V_{S_2}^{\mathcal{T}_1} \\ &= \zeta + \gamma^2(\zeta + \gamma^2 V_G^{\mathcal{T}_1}) \\ &= \zeta + \gamma^2\zeta + \gamma^4 V_G^{\mathcal{T}_1} \end{aligned}$$

The objective value of the depth-1 decision tree policy is :

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

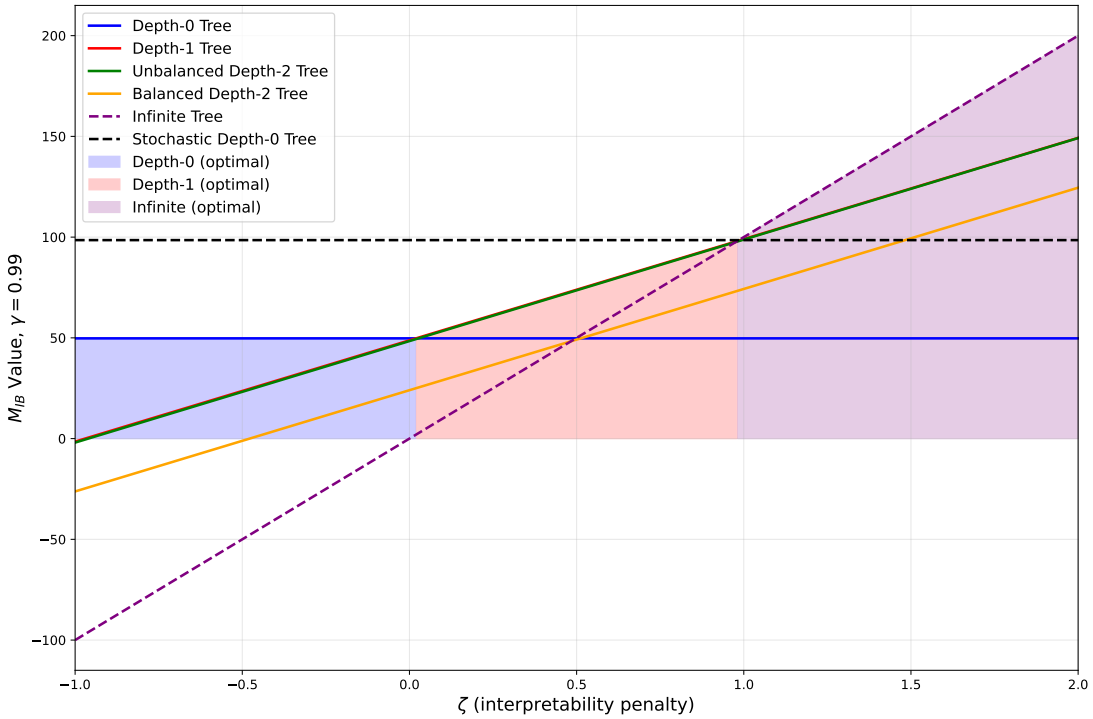


FIGURE 2.4 – IBMDP objective values.

2.2.2 Identifying the optimality range of the depth-1 tree

2.3 Results

Now that we have shown theoretically that there exists values of *zeta* such that the depth-1 tree policy from Figure (cite) is optimal in terms of IBMDP value (cite) and is also optimal in terms of MDP value (cite), we show that existing algorithms fail to learn the depth-1 tree even though it is the optimal IBMDP policy. We conjecture that this is because learning decision tree policies in IBMDP is equivalent to learning a deterministic reactive POMDP policy.

2.3.1 Experimental Setup

IBMDP : we will benchmark RL algorithms for the IBMDP of Section (cite) when the base part of the state is hidden to guarantee that the learned policy is a tree. There are two information gathering actions $x \leq 0.5$ and $y \leq 0.5$. We have to be extra careful when implementing this grid world because we work in the setting of discounted infinite

horizon MDP ; so in theory trajectories are infinite which is prohibited in practice. To ensure that trajectories are diverse, we reset the learning agent every 100 steps in the MDP. Furthermore we try to solve 200 different IBMDPs. Each IBMDP has γ fixed to 0.99 and ζ value linearly spaced in $[-1, 2]$.

Baselines : we consider two groups of RL algorithms. The first group is standard tabular RL ; Q-learning (cite)(cite), Sarsa (cite)(cite), and Policy Gradient (cite)(cite). (cite) and (cite) already studied the theoritcal and practical implications of applying standard RL directly to POMDP by considering partial observations as full 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. We draw similar conclusions from our experiments. We also use a vanilla tabular Policy Gradient baseline (cite), which to the best of our knowledge, nobody studied in our setting. In theory PG should not be a good candidate algorithm to solve our problem since it can learn stochastic policies that we showed can be better than our target policy (cite). More recently, new algorithms were developped specially for our setting of learning policies for POMDPs. Those algorithms are called asymmetric RL which authors of (cite) use without knowing. Asymmetric algorithms, also called (rememebr name deployment) ; 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. Asymmetric Q-learning (cite) make use of state-dependent Q function $Q : S \to \mathbb{R}$ to use a TD target to update a $Q : O \times A \to \mathbb{R}$ (cite)(algo). Asymmetric Policy Gradient (cite) use a state-dependent critic $V : S \to \mathbb{R}$ to plug into the gradient estimate of a policy $\pi : O \times A \to \mathbb{R}$. We benchmark Asymmetric Q-learning and Asymmetric Sarsa and the tabular algorithm from (cite) that we call JSJ in our experiments (algo) which is equivalent to a tabular Asymmetric Policy Gradient. In previous work (cite) Deep RL version of asymmetric algorithms have been benchmarked and showed to only work well for history-dependent policies or Q-function which we can't use for our particular problem. In particular in Figures (cite)(cite) of (cite) and cite() ; the authors show that for the particular case of asymmetric critic or target depending on full states and policy or Q-function depending on partial observation, the performance of asymmetric RL can be worse than that of non-specialized naive RL. We use at least one million timesteps to train each agent. Each agent is trained on 100 seeds.

Hyperparameters : For all RL baselines in the first part of this section we use, when applicable, exploration rates $\epsilon = 0.3$ and/or learning rates $\alpha = 0.1$.

Algorithme 7 : Asymmetric Q-Learning (cite)

Data : POMDP $\mathcal{M}_{po} = \langle X, O, A, R, T, T_0, \Omega \rangle$, learning rates α_1, α_2, exploration rate ϵ
Result : Partially observable deterministic policy π
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
    $U(x_t, a_t) \leftarrow (1 - \alpha_1) U(x_t, a_t) + \alpha_1 y$
    $Q(o_t, a_t) \leftarrow (1 - \alpha_1) Q(o_t, a_t) + \alpha_1 y$
    $x_t \leftarrow x_{t+1}$
    $o_t \leftarrow o_{t+1}$
  end
end
$\pi(o) = \arg\max_a Q(o,a)$ // Extract greedy policy

Metrics : we will consider two main metrics ; the sub-optimality gap of the learned policy with respect to the IBMDP objective (cite) and the proportion of the type of learned decision tree for the original grid world. Because we know the whole (IB)MDP model that we can represent exactly as tables ; and because we know for each ζ value the IBMDP objective value of the optimal partially observable IBMDP policy ; we can report the **exact** sub-optimality gaps :

$$\text{Gap}(\pi) = |V(\pi^\star) - V(\pi)|$$

Where we can compute $V(\pi)$ exactly by first mapping $\pi : O \times A \to \mathbb{R}$ to $\pi_S : S \times A \to \mathbb{R}$ by setting $\pi_S(s,a) = \pi(o,a)$ if $o \in s \; \forall s \in S, \forall o \in O \forall a \in A$. And then we apply repetitevly the Bellman operator (show). Similarly, once the RL training finished, we can use the tree extraction algorithm (cite) to count the learned tree distributions across the 100 seeds of each experiment.

Algorithme 8 : 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 : Policy π_θ and Q-function estimates
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

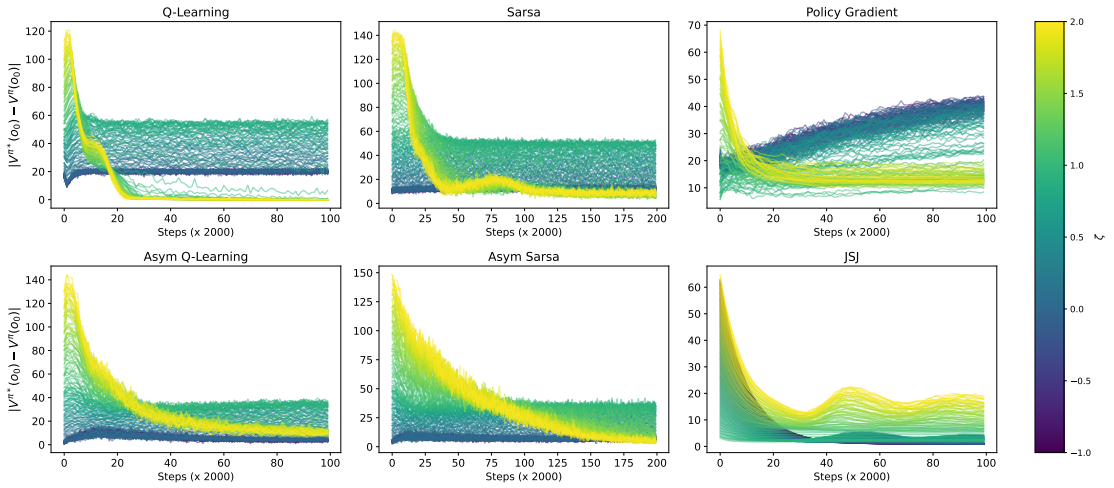


FIGURE 2.5 – Baseline algorithms for learning decision tree policies by learning partially observable policies in IBMDPs. The top row represents the sub-optimality gaps during training. In each subplot, each single learning curve is colored by the value of ζ of the IBMDP they solve. And each single learning curve represent the sub-optimality gap averaged over 100 seeds. So for each algortihm we ran a total of 200×100 single training seed. The bottom row represents the distributions of final tree policies learned across the 100 seeds. For each ζ value, there are three color points each represeting the share of Depth-0 tree (red), Depth-1 tree (gree) and Depth-2 tree (blue).

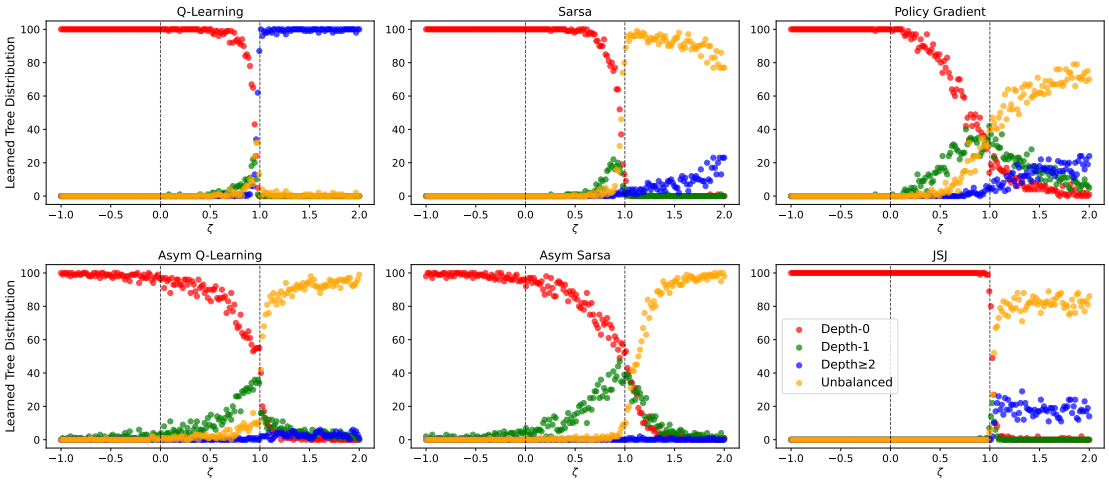


FIGURE 2.6 – Baseline algorithms for learning decision tree policies by learning partially observable policies in IBMDPs. The top row represents the sub-optimality gaps during training. In each subplot, each single learning curve is colored by the value of ζ of the IBMDP they solve. And each single learning curve represent the sub-optimality gap averaged over 100 seeds. So for each algortihm we ran a total of 200×100 single training seed. The bottom row represents the distributions of final tree policies learned across the 100 seeds. For each ζ value, there are three color points each represeting the share of Depth-0 tree (red), Depth-1 tree (gree) and Depth-2 tree (blue).

2.3.2 How well do exsisting baselines learn in POIBMDPs?

In Figure (cite) we observe that despite RL algorithms converging independently of the ζ values, not all runs fully minimize the sub-optimality gap. In particular all RL algorithms properly minimze the gap, i.e. learn the optimal policy or Q-function, for $\zeta \in [-1, 0]$, where the optimal policy is to repeat going right or down, Similarly all RL algorithms properly learn the optimal solutions for $\zeta \in [1, 2]$, where the optimal policy is to repeat taking informathion gathering actions. While in $\zeta \in]0, 1[$ where the depth-1 tree is optimal, no baseline can learn the optimal solution. In this range, baselines also converge to trees deeper than 1. One interpretation of this phenomenon is that the learning in POIBMDPs is very hard and so agents tend to converge to easy policies : repeating the same actions (whether it is a base or an information gathering action).

2.3.3 Why is it so hard to learn in POIBMPDs?

As discussed in the previous chapter (cite), POIBMDPs inherit many difficulties from POMDPs (list). To prove that solving POIBMDPs even though there are only a handful of states and observations and clear deterministic reactive optimal policies; we compare the success rate of baselines RL algorithms when applied to an IBMDP with $\gamma = 0.99$, $\zeta = 0.5$, and to the partially observable version of the same IBMDP. because we know that for those values of ζ and γ, the optimal partially observable policy is the depth-1 tree and the optimal-fully observable policy is the (cite); we can compute the success rate of an RL algorithm as the proportion of learn policies that match either optimal policy. In particular, for each of the three baselines Q-Learning, Sarsa, and Policy gradient, and their asymmetric counterparts when available; we train policies with each combination of those hyperparameters (cite) 10 times and compute the success rate over all individual runs. To fully quantify the difficulty of RL on those problems; we also consider advanced tricks for tabular RL.

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

The key observations from Figure (cite) is that POIBMDPs are way harder than their IBMDPs counterparts. Even though asymmetry seems to increase performance; learning a decision tree policy for a simple grid world directly with RL seem way to difficult and costly. We suggest researcher and practitioners interested in decision tree policies to

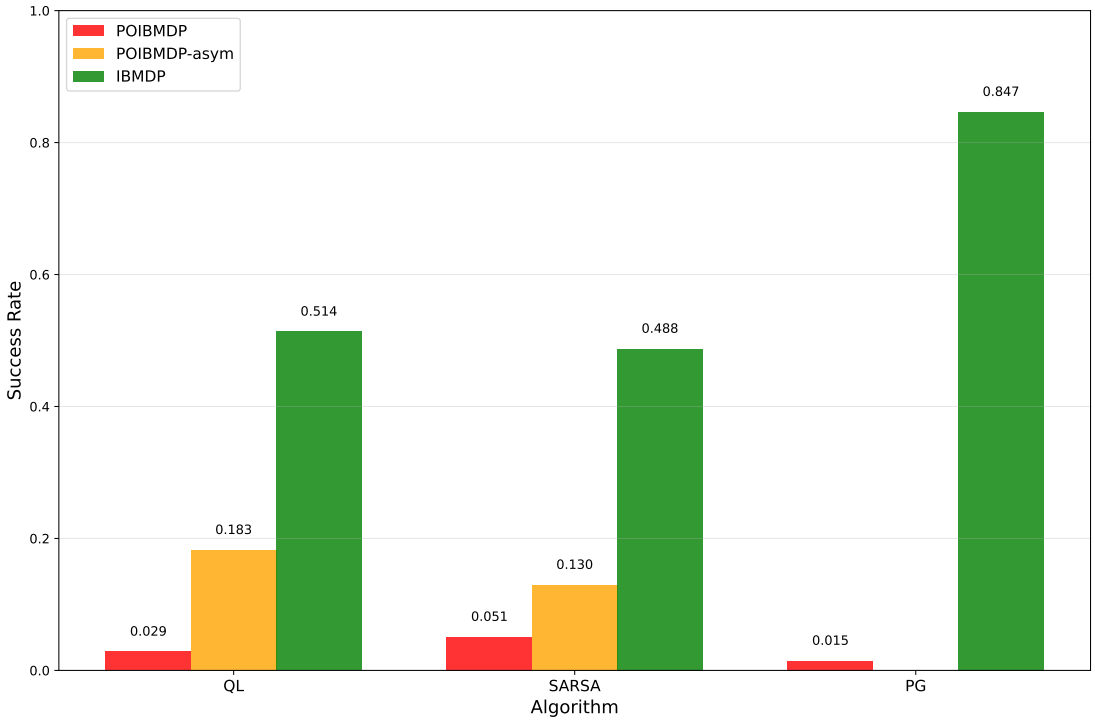


FIGURE 2.7 – Success rates of different RL algorithms over thousands of runs on two variants of the same IBMDP ; the default and Partially observable one.

forget the direct RL approach and focus on the indirect imitation-based approach (cite). Or, if for some reason one needs to still use reinforcement learning training; fitting parametric trees, trees whose depth and nodes are predifined and where only thresholds are optimized, then one can use SYMPOL (cite) or Soft decision trees (cite).

In the next chapter, we show that when the base MDP's transitions are independent of the action, then POIBMDPs are actually "fully"-observable.

Chapitre 3

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 classification problem 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.

After showing the equivalence between decision tree induction and solutions to POIBMDPs when the underlying base MDP encodes a classification task, we call them Classification-POIBMDPs, we highlight key limitations of the RL approach for decision tree induction. There are few other RL approaches for decision tree induction (cite). Here we focus on RL for Classification-POIBMDPs We defer the formal definition of classification tasks in the supervised learning setting in the next chapter and rather directly define an MDP that encodes such tasks.

Definition 11 (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 dsicounted cumulative reward objective (cite) are classifiers that maximize the prediction accuracy $\sum_{i=1}^{N} 1_{\pi(x_i)=y_i} = \sum_{i=1}^{N} R(x_i, \pi(x_i))$. Learning a classifier with a reward signal can also be done with contextual bandits (cite).

To learn a decision tree classifier, we use the POIBMDP formalism and show in the particular case of the base MDP being a Classification MDP, the POIBMDP is in fact an MDP with stochastic transitions.

Definition 12 (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 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.* $\frac{|\{x_i : L'_j \leq x_{ij} \leq U'_j \forall j \wedge x_{ik} > v\}|}{|\{x_i : L'_j \leq x_{ij} \leq U'_j \forall j\}|}$*)*

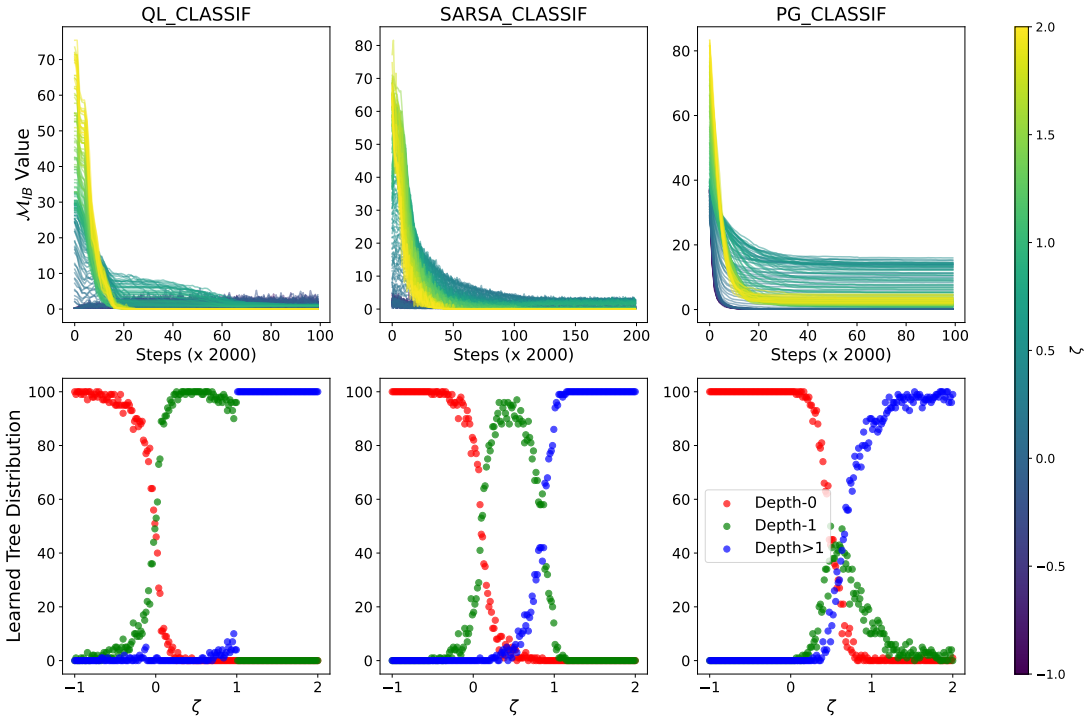


FIGURE 3.1 – RL algorithms for classification OIBMDPs

Chapitre 4

Conclusion

4.1 What happens when the MDP's transitions are independent of the current state ?

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-

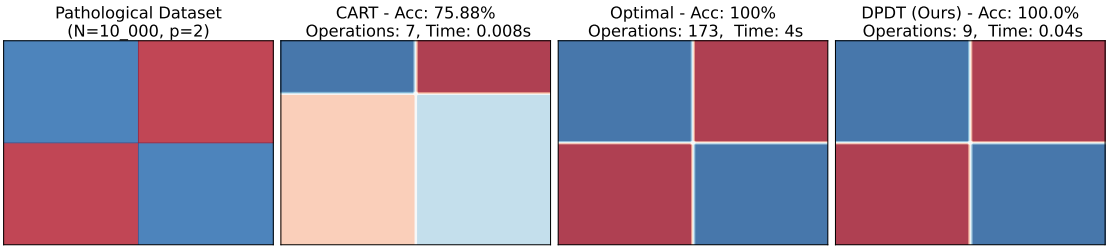


Figure 5.1 – Pathological dataset and learned depth-2 trees with their scores, complexities, runtimes, and decision boundaries.

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.

[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 6.1, 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 subset of trees of depth smaller or equal to D, $\mathcal{T}_D$. We now define some interesting split

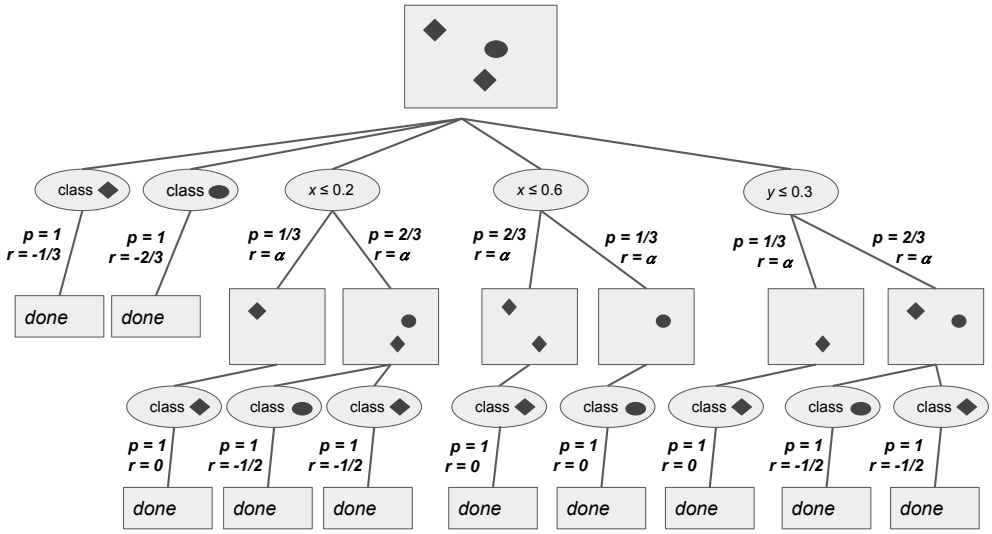


FIGURE 6.1 – Schematics of the MDP to learn a decision tree of depth 2 to classify a toy dataset with three samples, two features (x,y), and two classes (oval, diamond) and using an arbitrary splits generating function.

This figure represent a schematic of the MDP described in previous sections. It looks like an acyclic graph where nodes are either MDP actions or subsets of training examples.

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]

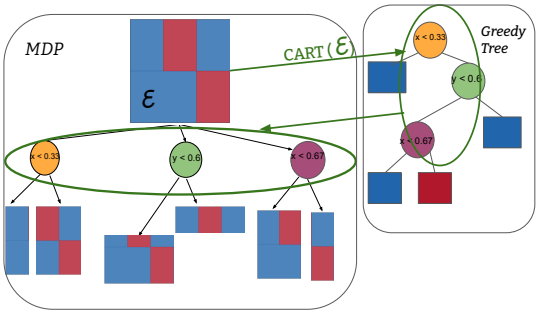
Top B most informative splits. [11] proposed to generate splits with a function that returns, for any state $s = (X, d)$, the B most informative splits over X with respect to some information gain measure such as the entropy or the Gini impurity. The number of states in the MDP would be $O((2B)^D)$. *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.

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

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 9 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.



This is a schematic of split generation with calls to CART.

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

Algorithme 9 : 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
$T \leftarrow E(\pi^*, s_0^{\mathcal{M}})$

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 : [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. [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, $\mathcal{L}_{\alpha=0}(T_D^{greedy}) > \mathcal{L}_{\alpha=0}(T_D^{DPDT})$.

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

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.

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 13 (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* $f(x,y) = (\lfloor 2x \rfloor + \lfloor 2y \rfloor) \bmod 2$.

The first greedy split is chosen at random on the XOR dataset from definition 13.

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 13. 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. □

When the first split is greedy on the XOR dataset from definition 13, 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 14 (Forced- Tree)**.** *Let us define the forced-tree as a greedy tree that is forced to make its first split at* $y = \frac{1}{2}$.

The forced-tree of depth 2 has a 0 loss on the XOR dataset from definition 13 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 6.3.4.

Démonstration. By definition of DPDT, all instances of DPDT with the CART nodes parameter $B \geq 2$ include the forced-tree from definition 14 in their solution set when applied to the XOR dataset (definition 13). We know from lemma 6.3.5 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 6.2 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 9 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 9 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 6.1) 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 6.3.4 and 6.3.4 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 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-

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

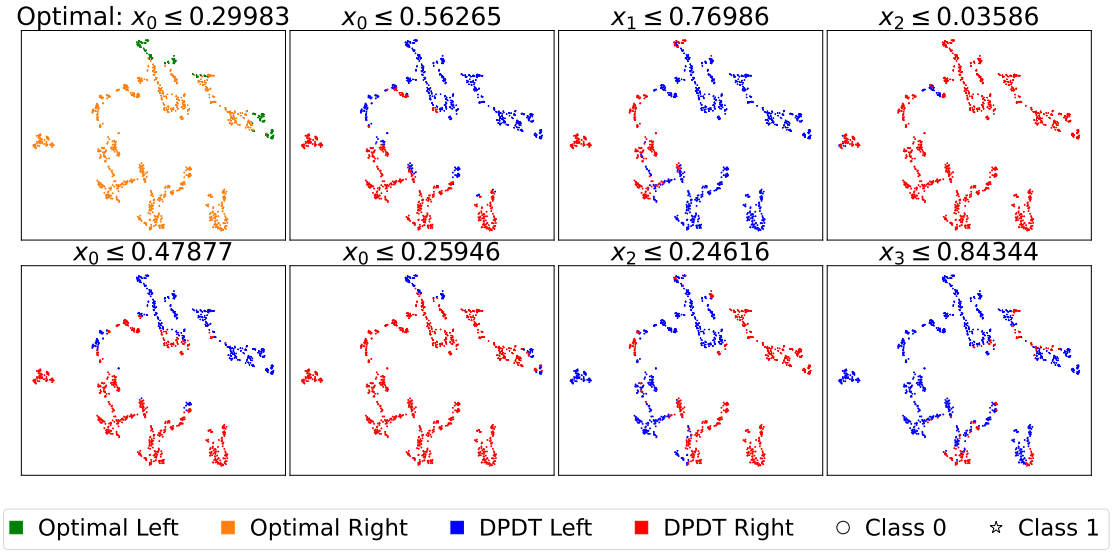


FIGURE 6.3 – Root splits candidate obtained with DPDT compared to the optimal root split on the Bank dataset. Each split creates a partition of p-dimensional data that we projected in the 2-dimensional space using t-SNE.
This figures shows a point cloud. The point cloud is similar to a clustered one with data from one class in a region and data from an other class in an other region. They are well separated.

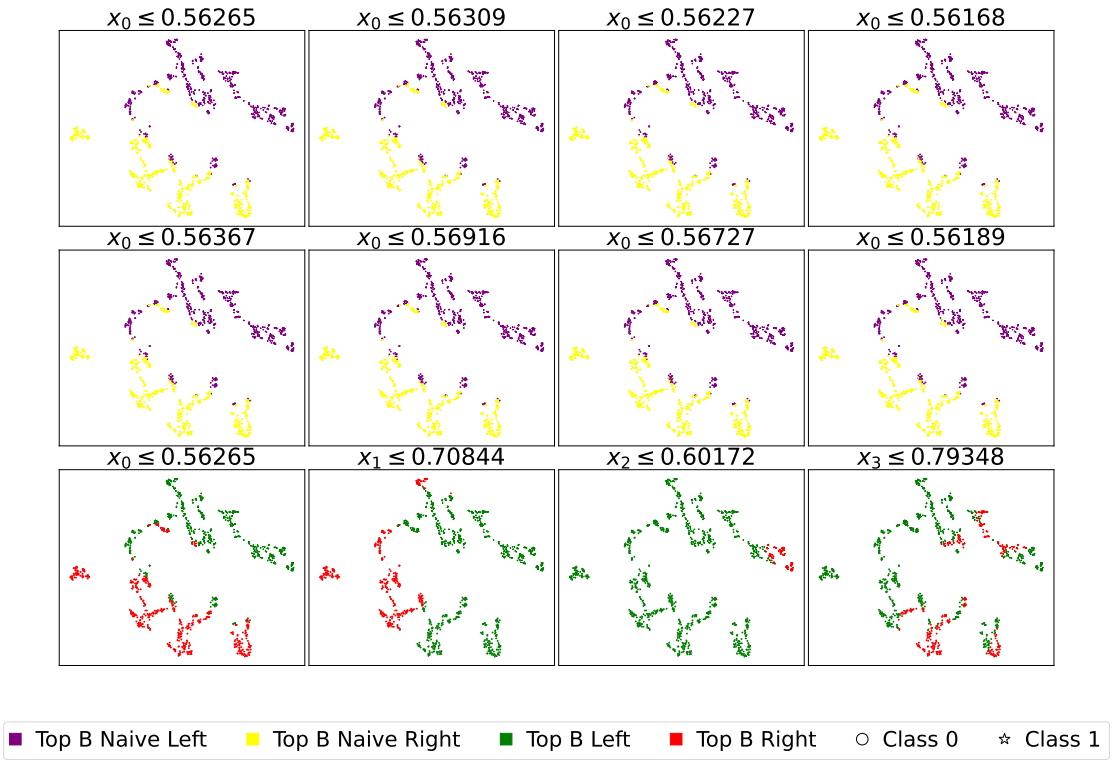


FIGURE 6.4 – Root splits candidate obtained with Top-B[11] on the Bank dataset. Each split creates a partition of p-dimensional data that we projected using t-SNE.
This figures shows a point cloud. The point cloud is similar to a clustered one with data from one class in a region and data from an other class in an other region. They are well separated.

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

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 continuous features. We use a time limit of 24 hours per dataset. DPDT and Top-B trees are obtained with Algorithm 9 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.

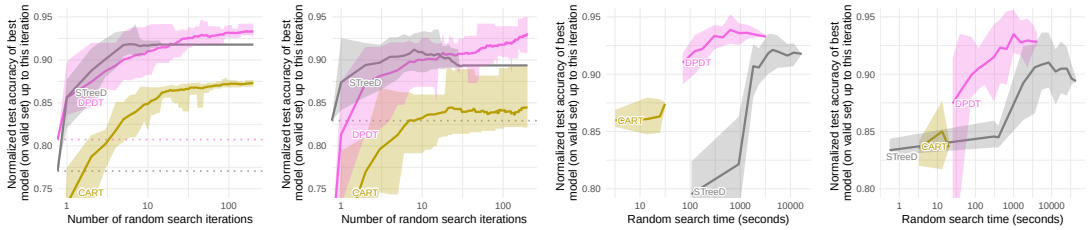
Datasets : we us the same datasets as the Quant-BnB paper [68].

Observations

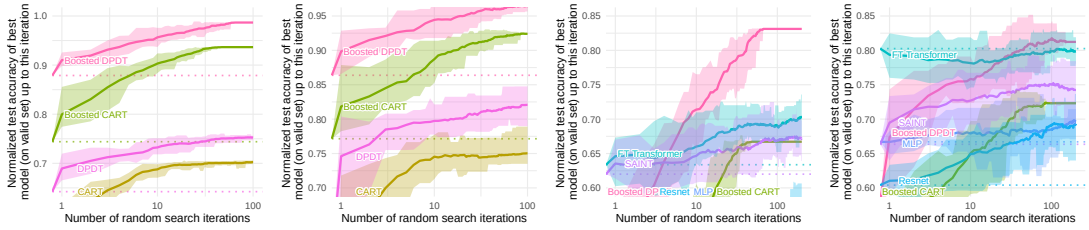
Near-optimality. Our experimental results demonstrate that unlike Deep RL, DPDT and Top-B approaches consistently improve upon greedy solutions while requiring significantly fewer operations than exact solvers. Looking at Table 6.1, we observe several key patterns : first, light DPDT with 16 candidate 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 6.3.4 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 (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



(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.
This figure contains many subplots. Each subplot contains curves that increase then flatten. Those curves show the evolution of the best solution found by decision tree induction algorithms after n iterations of random search.

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 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

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

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 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

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

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 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.

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

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 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

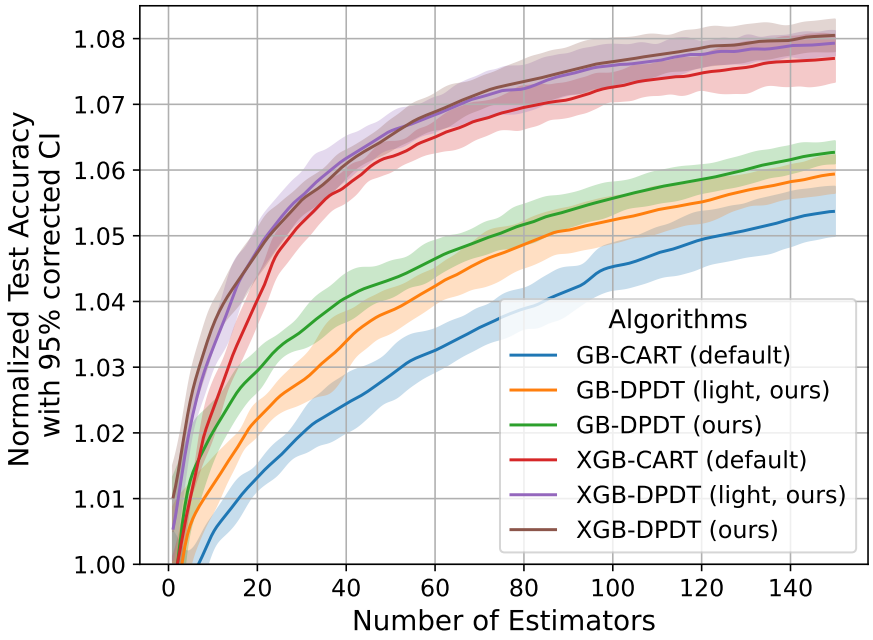


FIGURE 6.6 – Aggregated mean test accuracies of Gradient Boosting models as a function of the number of single trees.
This figures contains increasing then flattening curves representing the learning process of gradient boosting algorithms.

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 6.2

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	T
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±
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 ±

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 Accurac		
Names	Samples	Features	Classes	DPDT	OCT	MFOCT	BinOCT	CART	DPDT	OCT	MFOC
balance-scale	624	4	3	90.9%	71.8%	82.6%	67.5%	86.5%	77.1%	66.9%	71.3%
breast-cancer	276	9	2	94.2%	88.6%	91.1%	75.4%	87.9%	66.4%	67.1%	73.8%
car-evaluation	1728	6	4	92.2%	70.1%	80.4%	84.0%	87.1%	90.3%	69.5%	79.8%
hayes-roth	160	9	3	93.3%	82.9%	95.4%	64.6%	76.7%	75.4%	77.5%	77.5%
house-votes-84	232	16	2	100.0%	100.0%	100.0%	100.0%	99.4%	95.4%	93.7%	94.3%
soybean-small	46	50	4	100.0%	100.0%	100.0%	76.8%	100.0%	93.1%	94.4%	91.7%
spect	266	22	2	93.0%	92.5%	93.0%	92.2%	88.5%	73.1%	75.6%	74.6%
tic-tac-toe	958	24	2	90.8%	68.5%	76.1%	85.7%	85.8%	82.1%	69.6%	73.6%

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

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

Algorithme 10 : 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 $\hat{\pi}_N$;

Algorithm 10. This approach transforms the reinforcement learning task into a sequence of supervised learning problems.

Algorithm 10 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 10 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 10 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 10). The actions are always given by the expert (see Line 10). 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 10 is behavior cloning [78]. When we use importance sampling, Algorithm 10 is Q-DAgger [6]. In other cases, Algorithm 10 is DAgger [88].

Measuring Policy Interpretability After obtaining interpretable policy baselines using Algorithm 10, 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 computa-

tions 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 9.1, 9.2, and 9.3, 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 10, we distill deep neural network expert policies into less complex policy classes.

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

```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 9.1 – 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 9.2 – Tiny ReLU MLP for Pendulum.

```
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
                    (*@\colorbox{c
    else:
        if (x[0] - x[6]) <= -0.840
            return 0
        else:
            if (x[0] - x[4]) <= -0
                # Fire right engin
                (*@\colorbox{yello
            else:
                # Fire left engine
                (*@\colorbox{pink}{return 1}@*)
```

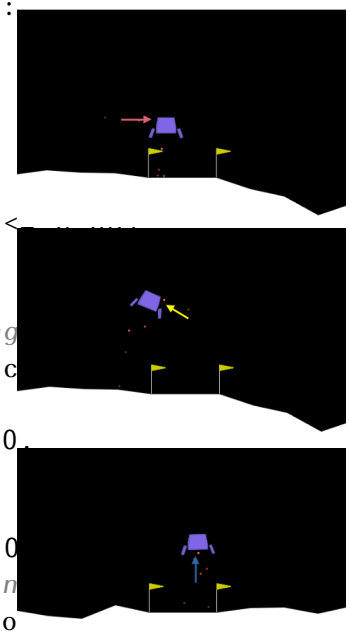


FIGURE 9.3 – An unfolded oblique tree policy's actions obtaining 250 rewards on Lunar Lander.

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 10).

a required step of imitation learning in Line 10. 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 learning [**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.

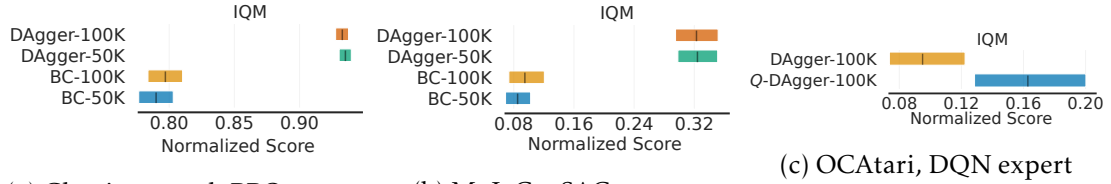
Neural network experts We do not train new deep reinforcement learning agents [71, 90, 47] but rather re-use ones available at the stables-baselines3 zoo [84]. Depending on the environments described next, we choose neural network policies from different deep reinforcement learning agents. Some may argue 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 10. 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 10 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 10 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.



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

FIGURE 9.4 – Performance of imitation learning variants of Algorithm 10 on different environments. We plot the 95% stratified bootstrapped confidence intervals around the IQMs.

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 9.4 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 9.4a,9.4b), 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 9.4b) and OCAtari (Q-Dagger, Figure 9.4c) 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 9.4a).

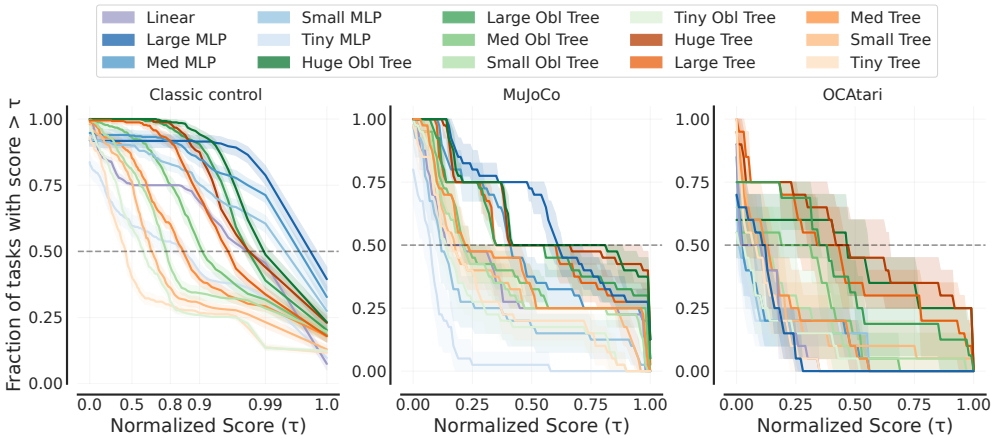


FIGURE 9.5 – Performance profiles of different policy classes on different environments.

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 9.4. From Figure 9.5 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

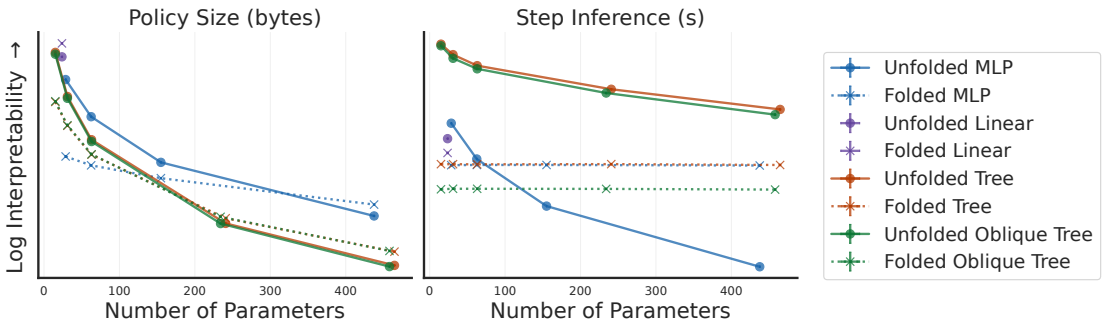


FIGURE 9.6 – Policies interpretability on classic control environments. We plot 95% stratified bootstrapped confidence intervals around means in both axes. In each sub-plot, interpreatbility is measured with either bytes or inference speed.

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 9.6 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 9.6, 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.

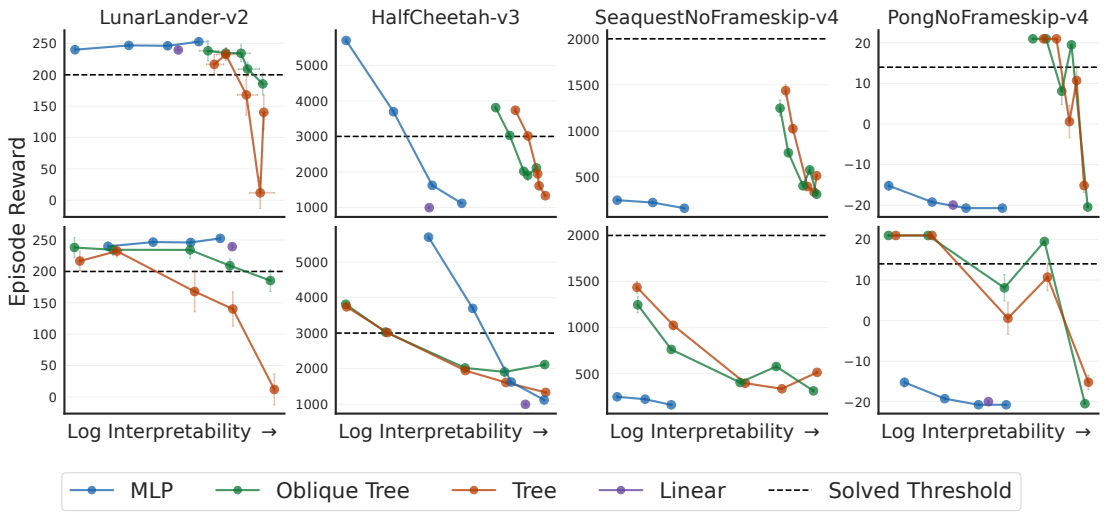


FIGURE 9.7 – Interpretability-Performance trade-offs. Top row, interpretability is measured with step inference times. Bottom row, the interpretability is measured with policy size. We plot 95% bootstrapped confidence intervals around means on both axes.

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 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 9.7, 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	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.

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 9.7 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 9.7 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.

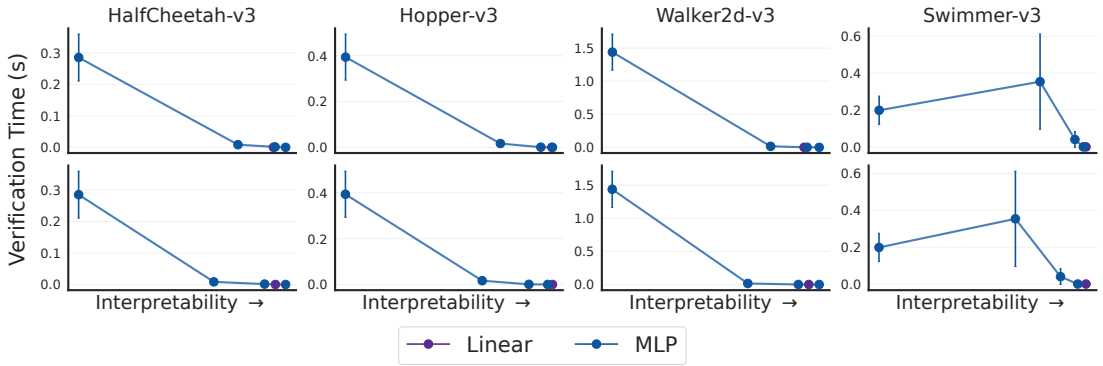


FIGURE 9.8 – Verification time as a function of policy interpretability. Top row, interpretability is measured with step inference times. Bottom row, the interpretability is measured with policy size. We plot 95% confidence intervals around means on both axes.

On Figure 9.8, 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.

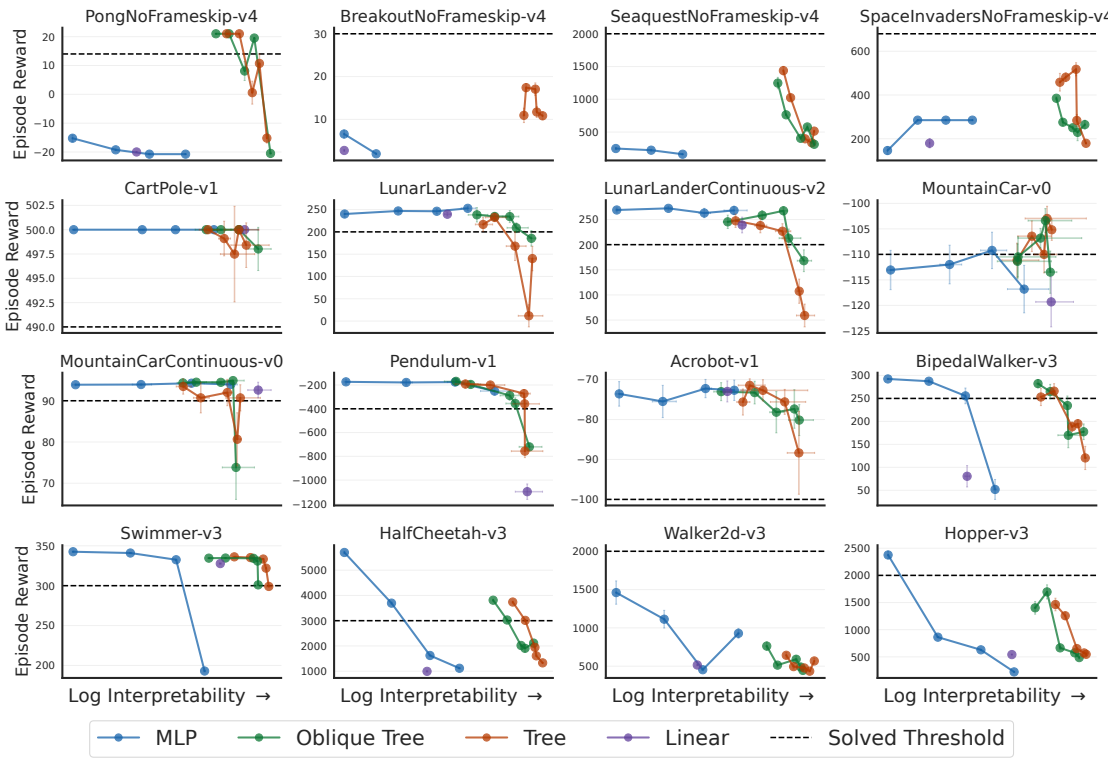


FIGURE 9.9 – Trade-off Cumulative Reward vs. Step Inference Time

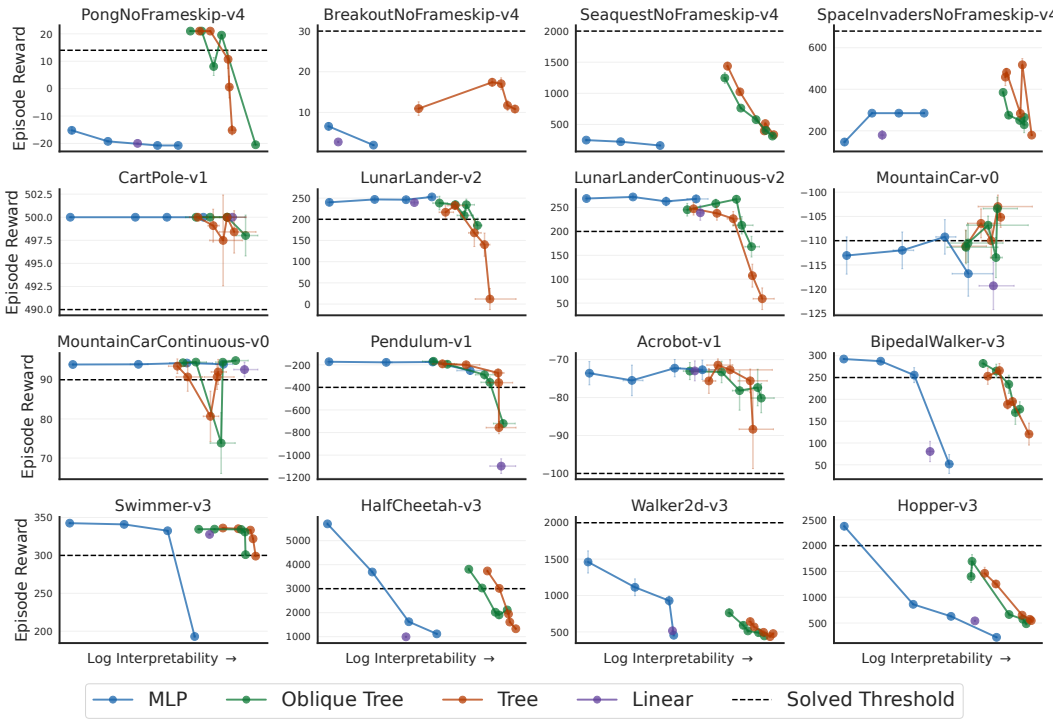


Figure 9.10 – Trade-off Cumulative Reward vs. Episode Inference Time

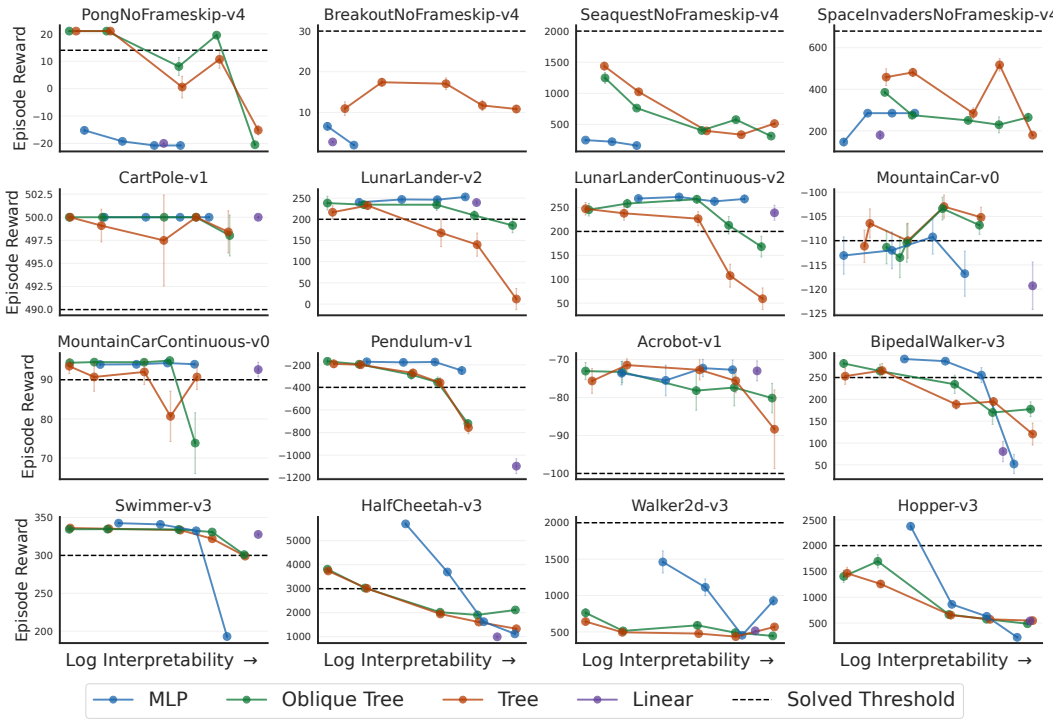


FIGURE 9.11 – Trade-off Cumulative Reward vs. Policy Size

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 9.6). 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 9.6, 9.7, and 9.8). 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 9.7) and showed the crucial need of our methodology as there is no better off policy class across tasks and metrics (c.f. Figures 9.5, 9.6, and 9.7).

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 9.10 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

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 :

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

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

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

Table des matières