Université de Lille

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

Discipline **Informatique**

Spécialité **Informatique et Applications**

Interprétabilité, Arbres de Décision, et Prise de Décisions Séquentielle

Thèse dirigée par Philippe Preux directeur
Riad Akrour co-directeur

Composition du jury

Rapporteurs	Olivier Buffet	chargé de recherche HDR à l'Université de Lorraine	
	Aurélie Beynier	professeure à Sorbonne Université	
Examinateurs	Lydia Boudjeloud-Assala	professeure à l'Université de Lorraine	présidente du jury
	Osbert Bastani	MCF à l'University of Pennsylvania	
Invité	Sonali Parbhoo	professeure assistante à l'Imperial College London	
Directeurs de thèse	Philippe Preux	professeur à l'Université de Lille	
	Riad Akrour	Inria starting faculty position à l'Inria	

Colophon

Mémoire de thèse intitulé « Interprétabilité, Arbres de Décision, et Prise de Décisions Séquentielle », écrit par Hector Kohler, achevé le 22 septembre 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

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

Discipline **Informatique**

Spécialité **Informatique et Applications**

Interprétabilité, Arbres de Décision, et Prise de Décisions Séquentielle

Thèse dirigée par Philippe Preux directeur
Riad Akrour co-directeur

Composition du jury

Rapporteurs	Olivier Buffet	chargé de recherche HDR à l'Université de Lorraine	
	Aurélie Beynier	professeure à Sorbonne Université	
Examinateurs	Lydia Boudjeloud-Assala	professeure à l'Université de Lorraine	présidente du jury
	Osbert Bastani	MCF à l'University of Pennsylvania	
Invité	Sonali Parbhoo	professeure assistante à l'Imperial College London	
Directeurs de thèse	Philippe Preux	professeur à l'Université de Lille	
	Riad Akrour	Inria starting faculty position à l'Inria	

Université de Lille

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

Academic Field **Computer Science**

Speciality **Computer Science and Applications**

Interpretability, Decision Trees, and Sequential Decision Making

Thesis supervised by

Philippe Preux	Supervisor
Riad Akrour	Co-Supervisor

Committee members

Referees	Olivier Buffet	HDR Research fellow at Université de Lorraine	
	Aurélie Beynier	Professor at Sorbonne Université	
Examiners	Lydia Boudjeloud-Assala	Professor at Université de Lorraine	Committee President
	Osbert Bastani	Associate Professor at University of Pennsylvania	
Guest	Sonali Parbhoo	Assistant Professor at Imperial College London	
Supervisors	Philippe Preux	Professor at Université de Lille	
	Riad Akrour	Inria Starting Faculty Position at Inria	

Interprétabilité, Arbres de Décision, et Prise de Décisions Séquentielle

Résumé

Dans cette thèse de doctorat, nous étudions des algorithmes d'apprentissage d'arbres de décision pour la classification suervisée et la prise de décision séquentielle. Les arbres de décision sont interprétables car les humains peuvent lire les opérations de l'arbre de décision depuis la racine jusqu'aux feuilles. Cela fait des arbres de décision le modèle de référence lorsque la vérification humaine est requise, comme dans les applications médicales. Cependant, les arbres de décision ne sont pas différentiables, ce qui les rend difficiles à optimiser, contrairement aux réseaux neuronaux qui peuvent être entraînés efficacement avec la descente de gradient. Les approches existantes d'apprentissage par renforcement interprétables apprennent généralement des arbres souples (non interprétables en l'état) ou sont ad hoc (entraînent un réseau neuronal puis entraînent un arbre à imiter le reseau). Cette apprentissage d'arbre indirect ne garantit pas de trouver des bonnes solutions pour le problème initial.

Dans la première partie de ce manuscrit, nous visons à apprendre directement des arbres de décision pour un processus de décision Markovien avec de l'apprentissage par renforcement. En pratique, nous montrons que cela revient à résoudre un problème de décision Markovien partiellement observable (PDMPO). La plupart des algorithmes d'apprentissage par renforcement existants ne sont pas adaptés aux PDMPOs. Ce parallèle entre l'apprentissage des arbres de décision et la résolution des PDMPOs nous aide à comprendre pourquoi, dans la pratique, il est souvent plus facile d'obtenir une politique experte non interprétable (un réseau neuronal) puis de la distiller en un arbre plutôt que d'apprendre l'arbre de décision à partir de zéro.

La deuxième contribution de ce travail découle de l'observation selon laquelle la recherche d'un arbre de décision est une instance complètement observable du problème précédent. Nous formulons donc l'induction d'arbres de décision comme la résolution d'un problème de décision Markovien et proposons un nouvel algorithme de pointe qui peut être entraîné à partir de données d'exemple supervisés et qui généralise bien à des données nouvelles.

Les travaux des parties précédentes reposent sur l'hypothèse que les arbres de décision sont un modèle interprétable que les humains peuvent utiliser dans des applications sensibles. Mais est-ce vraiment le cas ? Dans la dernière partie de cette thèse, nous tentons de répondre à des questions plus générales sur l'interprétabilité : pouvons-nous mesurer l'interprétabilité sans intervention humaine ? Et les arbres de décision sont-ils vraiment plus interprétables que les réseaux neuronaux ?

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

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

Interpretability, Decision Trees, and Sequential Decision Making

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 (RL) 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 problem (POMDP). 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 decision tree classifier (or regressor) is a fully observable instance of the above problem. 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

Table des matières

Preliminary Concepts

What is sequential decision making?

In this manuscript, we study algorithms for sequential decision making. Humans engage in sequential decision making in all aspects of life. In medicine, doctors have to decide how much chemotherapy to administer based on the patient's current health [40]. In agriculture, agronomists have to decide when to fertilize based on the current soil and weather conditions to maximize plant growth [48]. In automotive settings, the autopilot system has to decide how to steer based on lidar and other sensors to maintain a safe trajectory [69]. These sequential decision making processes exhibit key similarities : an agent takes actions based on current information to achieve a goal.

As computer scientists, we ought to design computer programs [61] that can help humans during these sequential decision making processes. For example, as depicted in figure 1, a doctor could benefit from a program that would recommend the "best" treatment given the patient's state. Machine learning algorithms [127] 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 [87]. In the cancer treatment example, the doctor follows the patient over time and adapts treatment to the patient's changing health. In that case, machine learning—and in particular *reinforcement* learning (RL) [121]—can be used to teach the program how to take decisions that lead to recovery based on how the patient's health changes from one dose to another. Such machine learning algorithms train increasingly performant programs that are deployed to, e.g., identify digits in images [67], control tokamak fusion [31], or write the abstract of a scientific article [3].

However, the computations performed by these programs cannot be understood and

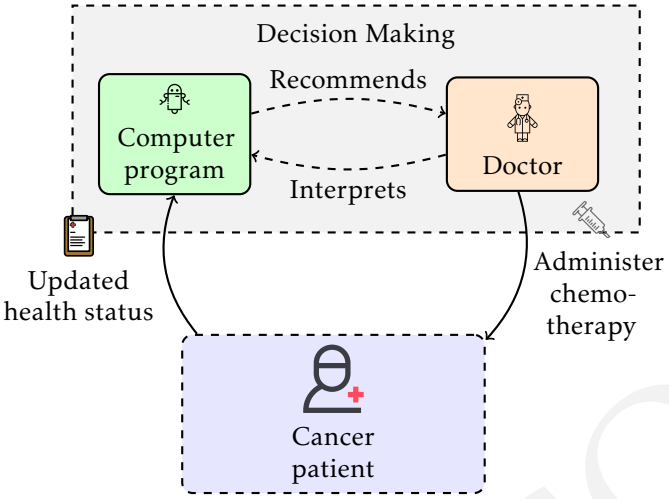


FIGURE 1 – Sequential decision making in cancer treatment. The AI system reacts to the patient's current state (tumor size, blood cells counts, etc.) and makes a recommendation to the doctor, who administers chemotherapy to the patient. The patient's state is then updated, and this cycle repeats over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

verified by humans : the programs are black-box. This can be problematic in applications where decisions have important consequences. Consider the following extreme scenario in the context of the cancer treatment example : a program recommends to the doctor to amputate a patient's limb. While amputating could be the only way to save the patient's life, it is not hard to imagine that such an extreme recommendation would not be followed by the doctor unless it comes with convincing explanations or unless the factors that influenced the program's recommendation can be presented to the doctor. In an other setting where the program directly act in the real-world, i.e. it does not make recommendations to a human but rather acts autonomously, black-box programs are also not desirable. Consider the slightly extreme scenario of a combat drone in which a program decides on who to shoot. In this context, it is hard to imagine leaders deploying such drones and programs without having strict guarantees that it will, e.g. not shoot civilians. Such guarantees can be obtained through formal verifications of programs which in turn can only be obtain for certain programs.

Next, we describe the notion of interpretability that is key to ensure that programs can be verified and their recommendations can be understood.

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 ? [19, p. 48]

What is remarkable is that the first recorded occurrence of "interpretability" was in the context of logic and computation. In Boole's system, the expression $x + y$ is interpretable only when x and y are disjoint sets, because then "+" clearly represents their union. By contrast, in e.g. a neural network program [110], an operation like $x + y$ typically means adding two hidden vectors in a high-dimensional space. The result is mathematically well-defined, but its semantic meaning—what concept or feature this new vector corresponds to—is black-box. Just as Boole worried that logic would lose its generality and practical use if restricted only to easily interpretable operations, there exists similar tension in machine learning today. In machine learning, should we limit ourselves to inherently interpretable programs (like decision trees [20], where

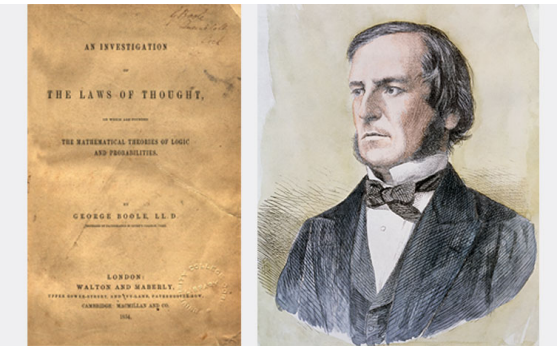


FIGURE 2 – British logician and philosopher George Boole (1815–1864) next to his book *The Laws of Thought* (1854), the oldest known record of the word "interpretability".

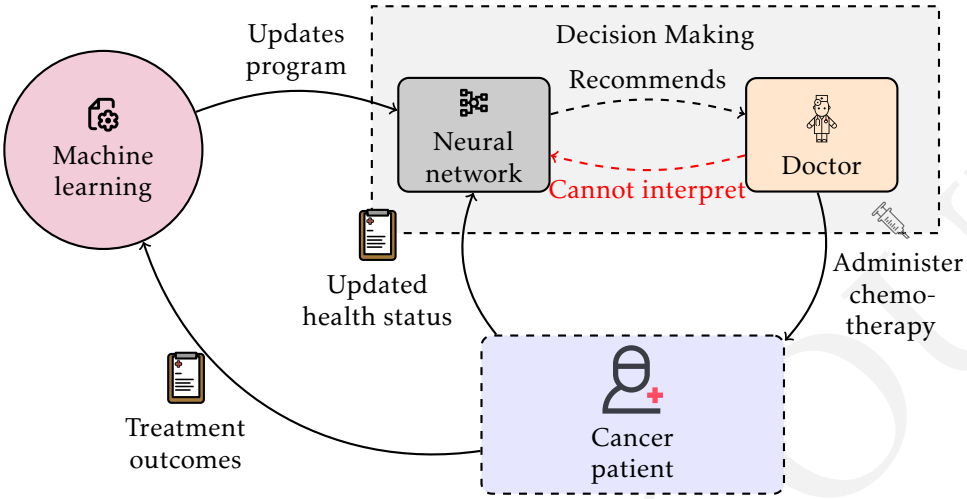
$x + y$ might literally mean combining two feature contributions, like blood cells count and tumor size) or accept uninterpretable computations in black-box programs (like in neural networks) in order to have broadly applicable and highly performant programs ? The global trend in machine learning research answers the latter.

In figure 3a, we illustrate how existing machine learning algorithms *could* be used in principle to help with cancer treatment. In truth, this should be prohibited without some kind of transparency in the program's recommendation : why did the program recommend such a dosage ? In figure 3b, we illustrate how machine learning *should* be used in practice. Ideally, we want doctors to have access to computer programs that can recommend "good" treatments and whose recommendations are interpretable. In the next section, present related works that fit this desiderata.

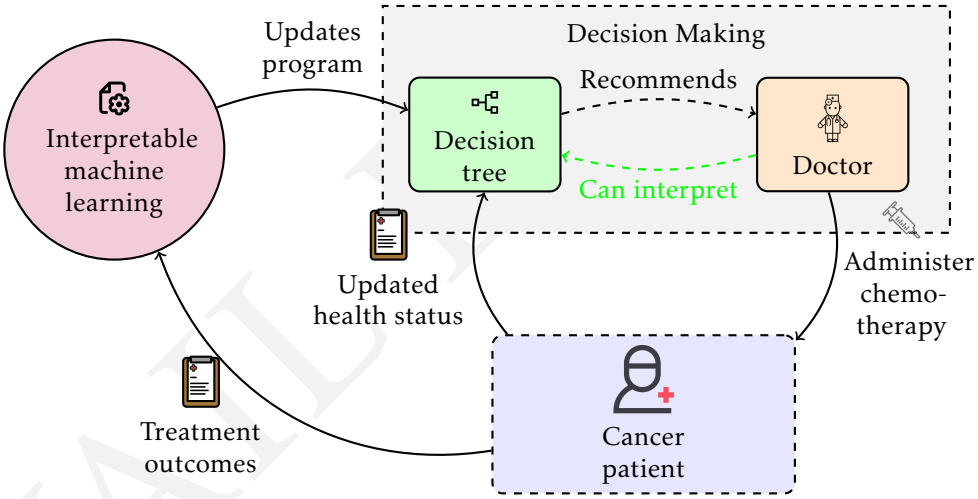
What are existing approaches for learning interpretable programs ?

In this section we follow sections 6 and 7 of [49] and section 5 of [88]. Furthermore we now employ the term "model" to refer to "programs" to be consistent with the machine learning research conventions. Models are essentially mappings from inputs to outputs that can be trained with machine learning algorithms while programs might designate other types of computations like `print("Hello World");`.

Interpretable machine learning provides either local or global interpretations [49]. Global methods, like decision tree induction [20], return a model whose outputs can be interpreted without having to run an additional algorithm. By contrast, local methods require to run an additional algorithm, e.g. linear regression [138], but are agnostic



(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 because the computations are operations over uninterpretable concepts (cf. end of section 4 and Boole's quote 4). Bottom : an interpretable decision tree approach where the doctor can understand and verify the AI's recommendations because the operations are performed over meaningful concepts (cf. end of section 4 and Boole's quote 4). Both systems learn from treatment outcomes to improve their recommendations over time.

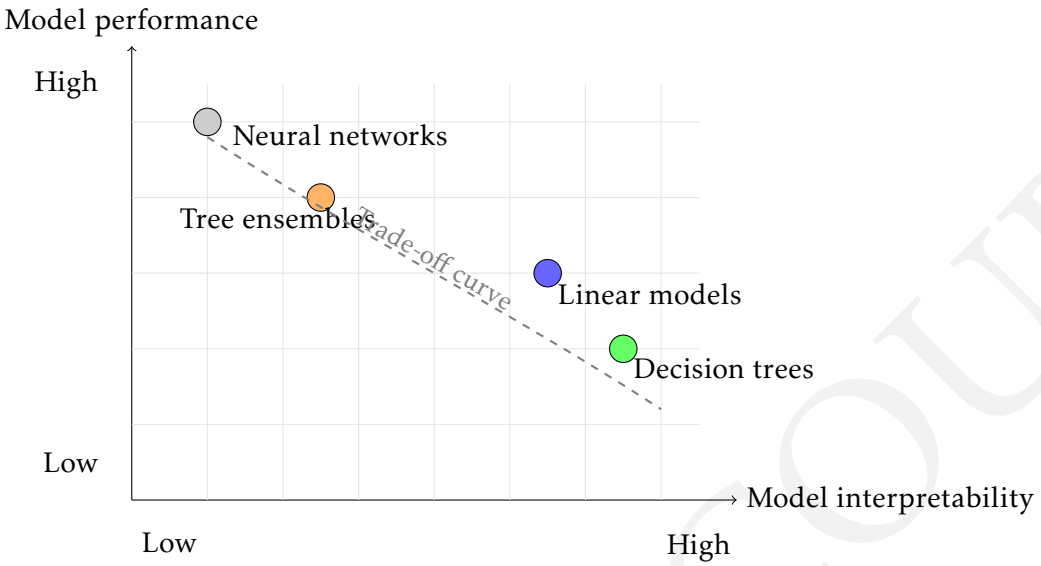


FIGURE 4 – The interpretability–performance trade-off in machine learning. Different model classes are positioned according to their typical interpretability and performance characteristics. The dashed line illustrates the general trade-off between these two properties.

to the model class. In figure 4 we present a popular trade-off between interpretability and performance of different model classes based on various user studies and popular beliefs [42, 43, 83, 130, 66, 63, 117, 56].

Local interpretable model-agnostic explanations (LIME) [109] is a popular local interpretability method. Given a model, LIME works by perturbing the input and learning a simple interpretable model locally to explain that particular prediction (see figure 5). For each individual prediction, LIME provides interpretations by identifying which features were most important for that specific decision.

Local interpretability can also be called explainability. Global interpretability methods constrain the model class so that the computations are transparent or verifiable by construction. On the other hand, explainability methods–or local interpretability methods–keep black-box models and generates post hoc explanations of their decisions. In additions to linear models, explanations can take various forms : visual explanations with saliency maps [101], attribution such as SHAP[77], attention-based highlighting [113].

While useful for insight, these explanations are often subjective and might not be faithful to the underlying computations [5]. For safety-critical settings, this motivates our focus on models that are interpretable by design. Those interpretable by design

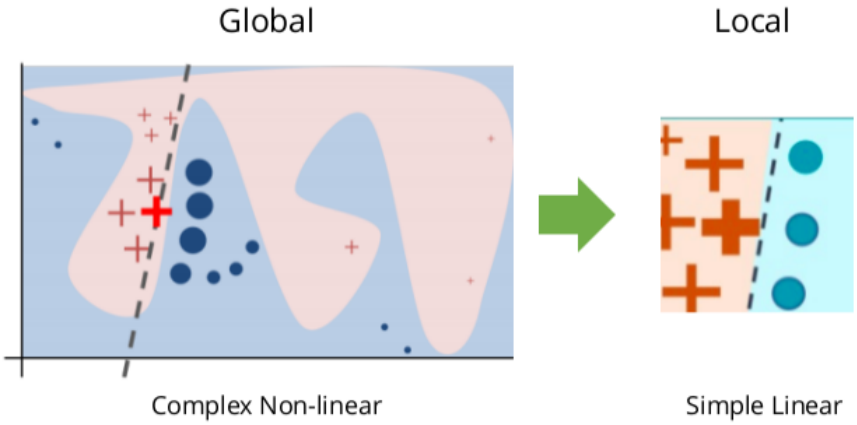


FIGURE 5 – Local Interpretable Model-agnostic Explanations [109] fit an interpretable linear model to data around the red cross prediction to be interpreted.

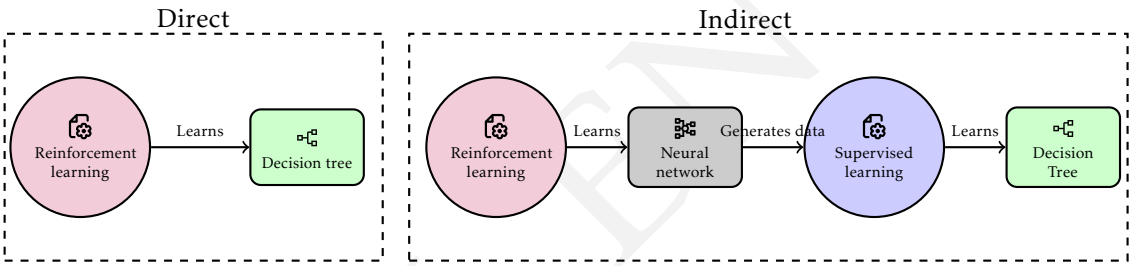


FIGURE 6 – Comparison of direct and indirect approaches for learning interpretable models in sequential decision making.

models can be obtained by global interpretability methods that we present next.

Global approaches are either direct or indirect [88]. Direct algorithms, such as decision tree induction [20], *directly* learn an interpretable model optimizing some objective (see figure 4). One key challenge motivating this thesis is that decision tree induction is well-developed for supervised learning but not for reinforcement learning. To directly learn interpretable models for sequential decision making, one must design new algorithms which will be the core of the first part of this thesis.

Most existing research has focused on developing indirect methods. Indirect methods for interpretable sequential decision making—sometimes called *post hoc* methods—begin by learning a non-interpretable model (e.g., reinforcement learning of a neural network model), and then use supervised learning to fit an interpretable model

that emulates the black-box. Indirect methods rely on behavior cloning or imitation learning [99, 111] to emulate the black-box models. Most work on interpretable sequential decision focuses on the indirect approach[12, 131].

Verifiable Reinforcement learning via Policy Extraction, or VIPER [12], is a strong indirect method to learn decision tree models for sequential decision making. VIPER first trains a neural network model with reinforcement learning and then fit a decision tree to minimize the disagreement between the neural network and the tree outputs given the same inputs. They show that decision tree models, in addition to being transparent, are also fast to verify in the formal sense of the term [137]. Programmatic models are an interpretable class that contains decision trees. Programmatically Interpretable Reinforcement Learning (PIRL) [131] synthesizes programs in a domain-specific language, also by imitating a neural network model.

However, unlike direct methods that return interpretable models optimizing the desired objective, indirect methods learn an interpretable model to match the behavior of a black-box that itself optimizes the objective of interest. Due to the restricted policy class, the best decision tree model might employ a completely different strategy than the best black-box neural network model to solve the task. Furthermore, the decision tree model that best fits the best neural network model might be sub-optimal compared to the decision tree model that best solves the task of interest. Hence, there is no guarantee that optimizing this surrogate objective of best emulating a black-box yields the best interpretability–performance trade-offs. Figure 6 illustrates the key difference between these two approaches.

Beyond direct and indirect learning, a complementary strategy is to train experts that are inherently easier to imitate and understand. This is achieved by adding interpretability-oriented regularization during training. In the context of supervised learning tasks, authors of [108] regularize the neural network model during training such that indirect approaches will be biased towards more interpretable trees.

In addition to finding models which computations can be read by humans or that can be formally verified, interpretable machine learning has also been used to detect reward misalignment in sequential decision making : by exposing the decision process of a model, one can identify goal mis-specification or unintended shortcuts. Such shortcuts can be, for example, following the shadow of someone instead of actually following someone because for the model they lead to the same reactions. The learning of interpretable models for misalignment detection has been heavily studied by Quentin Delfosse contemporarily to this manuscript [33, 114, 32, 34]. In the next chapter, we describe technical preliminaries useful to understand the content of this manuscript.

Technical preliminaries

What are decision trees ?

As the reader might have already guessed, we will put great emphasis on decision tree models as a mean to study interpretability. While other interpretable models might have other properties than the ones we will highlight through this thesis, one conjecture from [49] is that interpretable models are all hard to optimize or learn because they are non-differentiable in nature. This is something that will be key in our study of decision tree models that we introduce next and that we illustrate in figure 7.

Definition 1 (Decision tree)**.** *A decision tree is a rooted tree $T = (\mathcal{N}, E)$ where :*

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

The depth of a tree is the maximum path length from root to any leaf.

How to learn decision trees ?

Training decision trees to optimize the following supervised learning objective is well studied [20].

Definition 2 (Supervised learning objective for classification or regression)**.** *Assume that we have access to a set of N examples denoted $\mathcal{E} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$. Each datum $\boldsymbol{x}_i \in \mathcal{X}$ is described by a set of p features x_{ij} with $1 \leq j \leq p$. $y_i \in \mathcal{Y}$ is the label associated with $\boldsymbol{x}_i$. For*

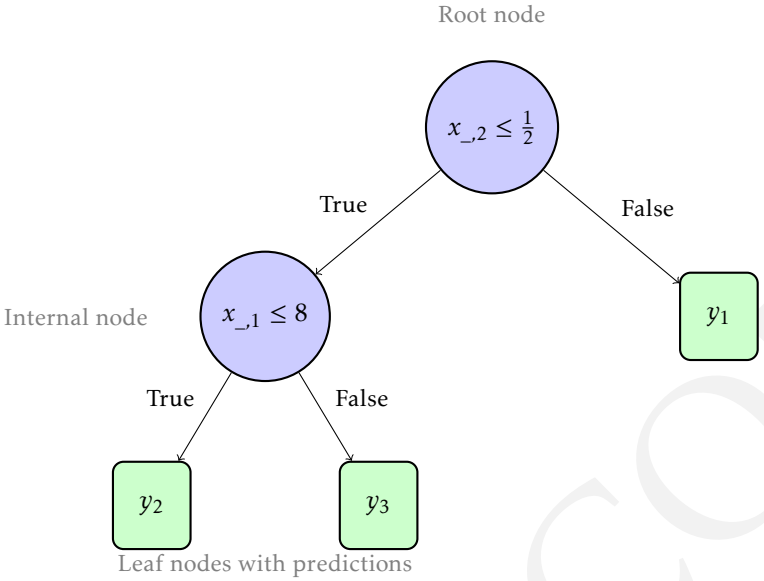


FIGURE 7 – A generic depth 2 decision tree with 2 nodes and 3 leaves. The root node applies the test $1_{\{x_{\_,1} \leq \frac{1}{2}\}}$ to check if the first features of data is below $\frac{1}{2}$. Edges represent the outcomes of the tests in each internal nodes(True/False), and leaf nodes contain predictions $y_l \in \mathcal{Y}$. For any input $\boldsymbol{x}_i$, the tree defines a unique path from root to leaf.

a classification task $\mathcal{Y} = \{1, \ldots, K\}$ and for a regression task $\mathcal{Y} \subseteq \mathbb{R}$. The goal of supervised learning for a classification (or regression) task is to find a classifier (or a regressor) $f : \mathcal{X} \to \mathcal{Y}$ that minimizes the following objective :

$$\mathcal{L}_\alpha(f) = \frac{1}{N} \sum_{i=1}^{N} l(y_i, f(\boldsymbol{x}_i)) + \alpha C(f), \tag{1}$$

where f is a model in a particular model class, e.g. the set of neural networks with relu activations or decision trees with depth at most 4, and where $C : \mathcal{F} \to \mathbb{R}$ is a regularization penalty.

The classification and regression trees (CART) algorithm [20] (cf. algorithm 1), developed by Leo Breiman and colleagues in 1984, is one of the most widely used method for learning decision trees from supervised data. CART builds binary decision trees through a greedy, top-down approach that recursively partitions the feature space. Throughout this manuscript we will focus on numerical features ($\mathcal{X} \subsetneq \mathbb{R}^p$) and assume that non-numerical features, e.g. nominal features, can always be converted to numerical ones without loss of information. At each internal node, the algorithm selects the feature

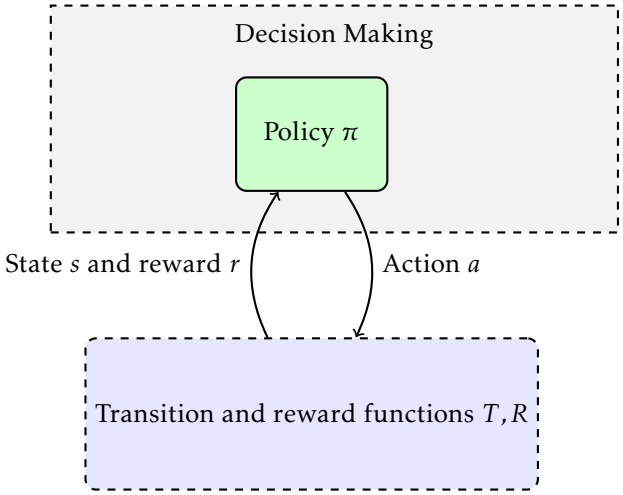


Figure 8 – Markov decision process

and its threshold that best split the data according to a purity criterion such as the Gini impurity for classification or mean squared error for regression. CART uses threshold-based tests of the form $1_{\{x_{\_j} \leq v\}}$, where $1_{\{\}}$ is the indicator function. The key idea is to find splits that maximize the homogeneity of the resulting subsets. We use CART as well as other decision tree algorithms in this manuscript.

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

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

Markov decision processes and problems

Markov decision processes (MDPs) were first introduced in the 1950s by Richard Bellman [14]. Informally, an MDP models how an agent acts over time to achieve a goal. At every time step, the agent observes its current state (e.g., patient weight and tumor size) and takes an action (e.g., administers a certain amount of chemotherapy). The agent receives a reward that helps evaluate the quality of the action with respect to the goal (e.g., tumor size decreases when the objective is to cure cancer). Finally, the agent transitions to a new state (e.g., the updated patient state) and repeats this process over time. Following Martin L. Puterman's book on MDPs[102], we formally define :

Algorithme 1 : CART for decision tree induction to optimize the supervised learning objective 2

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

Definition 3 (Markov decision process)**.** *An MDP is a tuple $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$ where :*

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

Informally, we would like to act in an MDP so that we obtain as much reward as possible over time. For example, in cancer treatment, the best outcome is to eliminate the patient's tumor as quickly as possible. We can formally define this objective, that we call the reinforcement learning objective, as follows :

Definition 4 (Reinforcement learning objective)**.** *Given an MDP $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$ (cf. definition 3), the goal of reinforcement learning for sequential decision making is to find a model, also known as a policy, $\pi : S \to A$ that maximizes the expected discounted sum of rewards :*

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

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

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

Example : a grid world MDP

In figure 9, we present a very simple MDP (cf. definition 3). This MDP is essentially a grid where the starting state is chosen at random and the goal is to reach the bottom-right cell as fast as possible in order to maximize the RL objective (cf. definition 4). The state space is discrete with state labels representing 2D-coordinates. The actions are to move up, left, right, or down. The bottom-right cell gives reward 1 and is an absorbing state, i.e., once in the state, the MDP stays in this state forever. Other states give reward 0 and are not absorbing. The optimal actions that get to the goal as fast as possible in every state (cell) are presented in green in figure 9.

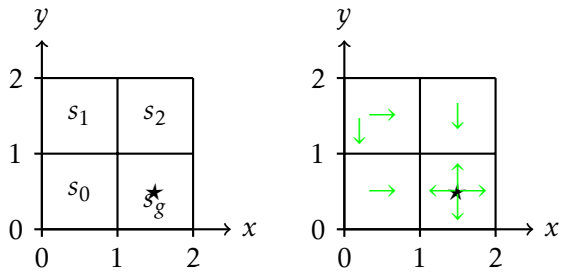


FIGURE 9 – A grid world MDP (left) and optimal actions w.r.t. the objective 4 (right).

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

Exact solutions for Markov decision problems

We begin with the planning setting in which the MDP transitions and rewards (cf. definition 3) are known. Leveraging the Markov property, one can use dynamic programming [14] to compute optimal policies with respect to the RL objective (cf. definition 4). Algorithms based on dynamic programming use the notion of *value* of states and actions :

Definition 5 (Value of a state)**.** *In an MDP $\mathcal{M}$ (cf. definition 3), the value of a state $s \in S$ under policy π is the expected discounted sum of rewards starting from state s and following policy π :*

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

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

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

Definition 6 (Optimal value of a state)**.** *The optimal value of a state $s \in S$, $V^{\star}(s)$, is the value of state s when following the optimal policy $\pi^{\star}$ (the policy that maximizes the RL objective (4)).*

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

Definition 7 (Optimal value of a state–action pair)**.** *The optimal value of a state–action pair $(s, a) \in S \times A$, $Q^{\star}(s, a)$, is the value when taking action a in state s and then following the*

optimal policy.

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

The well-known value iteration ([121], cf. algorithm 2), computes the values of states $^\star(s)$ with respect to the policy $\pi^\star$ maximizing the RL objective (4).

Algorithme 2 : Value iteration

Data : MDP $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$
Result : Optimal values $V^\star$ and optimal policy π^*
Initialize $V(s) = 0$ for all $s \in S$
repeat
  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`
  end
until *Convergence*;
for *each state* $s \in S$ **do**
  $\pi^*(s) \leftarrow \text{argmax}_a [R(s,a) + \gamma \mathbb{E}[V(s')|s' \sim T(s,a)]]$ `// Extract optimal policy`
end

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

Reinforcement learning of approximate solutions to MDPs

When the MDP transition function and reward function are unknown, one can use reinforcement learning algorithms–also known as agents–to learn values or policies maximizing the RL objective (cf. definition 4).

Reinforcement learning algorithms popularized by Richard Sutton [121] don't **compute** an optimal policy but rather **learn** an approximate one based on sequences of transitions $(s_t, a_t, r_t, s_{t+1})_t$. RL algorithms usually fall into two categories : value-based [121] and policy search [120]. Examples of these approaches are shown in algorithms 3, 4 and 5. Q-learning and Sarsa compute an approximation of $Q^\star$ (cf. definition 7) using temporal difference learning [121]. Q-learning is *off-policy* : it collects new transitions

with a random policy, e.g. epsilon-greedy. Sarsa is *on-policy* : it collects new transitions greedily w.r.t. the current Q-values estimates. Policy gradient algorithms [120] leverage the policy gradient theorem to approximate $\pi^\star$.

Q-learning, Sarsa, and policy gradients algorithms are known to converge to the optimal value or (locally) optimal policy under some conditions. There are many other ways to learn policies such as simple random search [79] or model-based reinforcement learning that estimates MDP transitions and rewards before applying e.g. value iteration [6]. Those RL algorithms–also known as tabular RL because they represent policies as tables with $|S| \times |A|$ entries–is limited to small state spaces. To scale to large state spaces, it is common to use a neural network to represent policies or values [122]. In the next section, we present deep reinforcement learning algorithms designed specifically for neural networks.

Algorithme 3 : Q-Learning

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

Deep reinforcement learning for large or continuous state spaces

Reinforcement learning has also been successfully combined with function approximations to solve MDPs with large discrete state spaces or continuous state spaces ($S \subset \mathbb{R}^p$ in definition 3). In the rest of this manuscript, unless stated otherwise, we write $\boldsymbol{s}$ a state vector in a continuous state space [1].

Deep Q-Networks (DQN) [89], described in algorithm 6 achieved super-human performance on a set of Atari games. Authors successfully extended the Q-learning (algorithm 3) to the function approximation setting by introducing target networks

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

Algorithme 4 : Sarsa

Data : MDP $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$, learning rate α, exploration rate ϵ
Result : Policy π
Initialize $Q(s,a) = 0$ for all $s \in S, a \in A$
Initialize state $s_0 \sim T_0$
Choose action a_0 using e.g. ϵ-greedy : $a_0 = \text{argmax}_a Q(s_0, a)$ with prob. $1 - \epsilon$
for *each step t* **do**
  Take action a_t, observe $r_t = R(s_t, a_t)$ and $s_{t+1} \sim T(s_t, a_t)$
  Choose action a_{t+1} using e.g. ϵ-greedy : $a_{t+1} = \text{argmax}_a Q(s_{t+1}, a)$ with prob. $1 - \epsilon$
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
  $s_t \leftarrow s_{t+1}$
  $a_t \leftarrow a_{t+1}$
end
$\pi(s) = \text{argmax}_a Q(s,a)$ // Extract greedy policy

Algorithme 5 : Policy Gradient RL (REINFORCE)

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

to mitigate distributional shift in the temporal difference error and replay buffer to increase sample efficiency.

Proximal Policy Optimization (PPO) [112], described in algorithm 7, is an actor-critic algorithm [121] optimizing a neural network policy. In actor-critic algorithms, cumulative discounted rewards starting from a particular state, also known as *the returns*, are also estimated with a neural network. PPO is known to work well in a variety of domains including robot control in simulation among others.

Algorithme 6 : Deep Q-Network (DQN)

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

In this manuscript we study those two deep reinforcement learning algorithms for various problems.

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

Unlike PPO or DQN for neural networks, there does not exist an algorithm that trains decision tree policies to optimize the RL objective (cf. definition 4). In fact, we

Algorithme 7 : Proximal Policy Optimization (PPO)

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

will show in the first part of the manuscript that training decision trees that optimize the RL objective is very difficult.

Hence, many interpretable reinforcement learning approaches first train a neural network policy–also called an expert policy–to optimize the RL objective (cf. definition 4) using e.g. PPO, and then fit a student policy such as a decision tree using CART (cf. algorithm 1) to optimize the supervised learning objective (cf. definition 2) with the neural policy actions as targets. This approach is known as imitation learning and is essentially training a student policy to optimize the objective :

Definition 8 (Imitation learning objective)**.** *Given an MDP $\mathcal{M}$ (cf. definition 3) expert policy π^* and a policy class Π, e.g. decision trees of depth at most 3, the imitation learning objective is to find a student policy $\hat{\pi} \in \Pi$ that minimizes the expected action disagreement with the expert :*

$$IL(\pi) = \mathbb{E}_{\boldsymbol{s} \sim \rho(\boldsymbol{s})} [\mathcal{L}(\pi(\boldsymbol{s}), \pi^*(\boldsymbol{s}))] \tag{2}$$

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

There are two main imitation learning methods used for interpretable reinforcement learning. Dagger ([131]; algorithm 8) is a straightforward way to fit a decision tree

policy to optimize the imitation learning objective8. VIPER ([12]; algorithm 9) was designed specifically for interpretable reinforcement learning. VIPER re-weights the transitions collected by the neural network expert by a function of the state-action value (cf. definition 4). The authors of VIPER showed that decision tree policies fitted with VIPER tend to have the same RL objective value as Dagger trees while being more interpretable (shallower or with fewer nodes) and sometimes outperform Dagger trees. Dagger and VIPER are two strong baselines for decision tree learning in MDPs, but they optimize a surrogate objective only, even though in practice the resulting decision tree policies often achieve high RL objective value. We use these two algorithms extensively throughout the manuscript. Next we show how to learn a decision tree policy for the example MDP (cf. figure 9).

Algorithme 8 : Dagger

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

Your first decision tree policy

Now the reader should know how to train decision tree classifiers or regressors for supervised learning using CART (cf. section 4). The reader should also know what an MDP is and how to compute or learn policies that optimize the RL objective (cf. definition 4) with (deep) reinforcement learning (cf. section 4). Finally, the reader should now know how to obtain a decision tree policy for an MDP through imitation learning (cf. definition 8) by first using RL to get an expert policy and then fitting a decision tree to optimize the supervised learning objective, using the expert actions as labels.

In this section we present the first decision tree policies of this manuscript obtained

Algorithme 9 : VIPER

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

using Dagger or VIPER after learning an expert Q-function for the grid world MDP of example (9) using Q-learning (cf. algorithm 3). Recall the optimal policies for the grid world, taking the green actions in each state in figure 9. Among the optimal policies, the ones that go left or up in the goal state can be problematic for imitation learning algorithms. Indeed, we know that for this grid world MDP there exists decision tree policies with a very good interpretability-performance trade-off : depth-1 decision trees that are optimal w.r.t. the RL objective. One could even say that those trees have the *optimal* interpretability-performance trade-off because they are the shortest trees that are optimal w.r.t. the RL objective.

In figure 10, we present a depth-1 decision tree policy that is optimal w.r.t. the RL objective and a depth-1 tree that is sub-optimal. The other optimal depth-1 tree is to go right when $y \leq 1$ and down otherwise. Indeed, figure 11 shows that the optimal depth-1 tree achieves exactly the same RL objective value as the optimal policies from figure 9, independently of the discount factor γ.

Now a fair question is : can Dagger or VIPER learn such an optimal depth-1 tree given access to an expert optimal policy from figure 9 ?

We start by running the standard Q-learning algorithm as presented in algorithm 3 with $\epsilon = 0.3$, $\alpha = 0.1$ over 10,000 time steps. The careful reader might wonder how ties are broken in the argmax operation from algorithm 3. While Sutton and Barto break ties by index value in their book [121] (the greedy action is the argmax action with smallest index), we show that the choice of tie-breaking greatly influences the performance

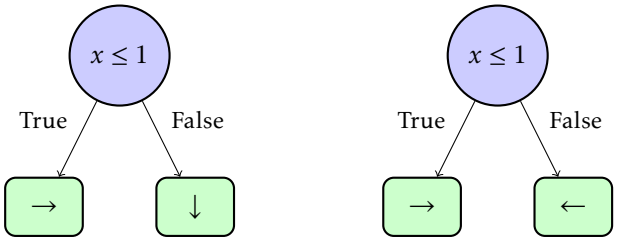


FIGURE 10 – Left, an optimal depth-1 decision tree policy. On the right, a sub-optimal depth-1 decision tree policy.

of subsequent imitation learning algorithms. Indeed, depending on how actions are ordered in practice, Q-learning may be biased toward some optimal policies rather than others. While this does not matter for one who just wants to find an optimal policy, in our example of finding the optimal depth-1 decision tree policy, it matters *a lot*.

In the left plot of figure 12, we see that Q-learning, independently of how ties are broken, consistently converges to an optimal policy over 100 runs (random seeds). However, in the right plot of figure 12, where we plot the proportion over 100 runs of optimal decision trees returned by Dagger or VIPER at different stages of Q-learning, we observe that imitating the optimal policy obtained by breaking ties at random consistently yields more optimal trees than breaking ties by indices. What actually happens is that the most likely output of Q-learning when ties are broken by indices is the optimal policy that goes left in the goal state, which cannot be perfectly represented by a depth-1 decision tree, because there are three different actions taken and a binary tree of depth $D = 1$ can only map to $2^D = 2$ labels.

This short experiment shows that imitation learning approaches can sometimes be very bad at learning decision tree policies with good interpretability-performance trade-offs for very simple MDPs. Despite VIPER almost always finding the optimal depth-1 decision tree policy in terms of the RL objective when ties are broken at random, we have shed light on the sub-optimality of indirect approaches such as imitation learning. This motivates the study of direct approaches (cf. figure 6) to directly search for policies with good interpretability-performance trade-offs with respect to the original RL objective. Next, we present the outline of the thesis.

Outline of the thesis

Throughout our thesis, we make the assumption that constraining models (e.g. policies or classifiers) to be decision trees is enough for ensuring interpretability. In this

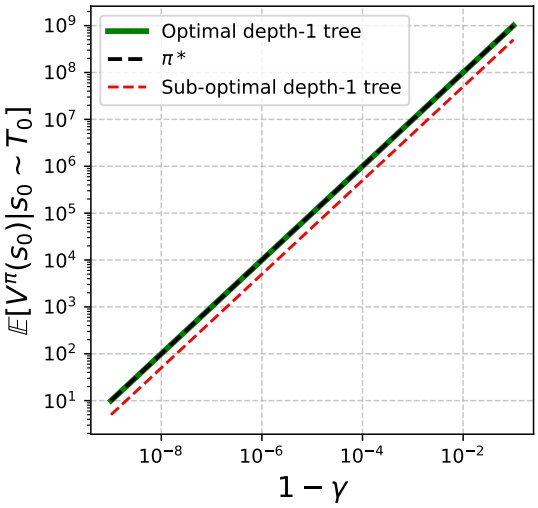


Figure 11 – The RL objective (4) values of the optimal policies from figure 9 and of the decision tree policies from figure 10.

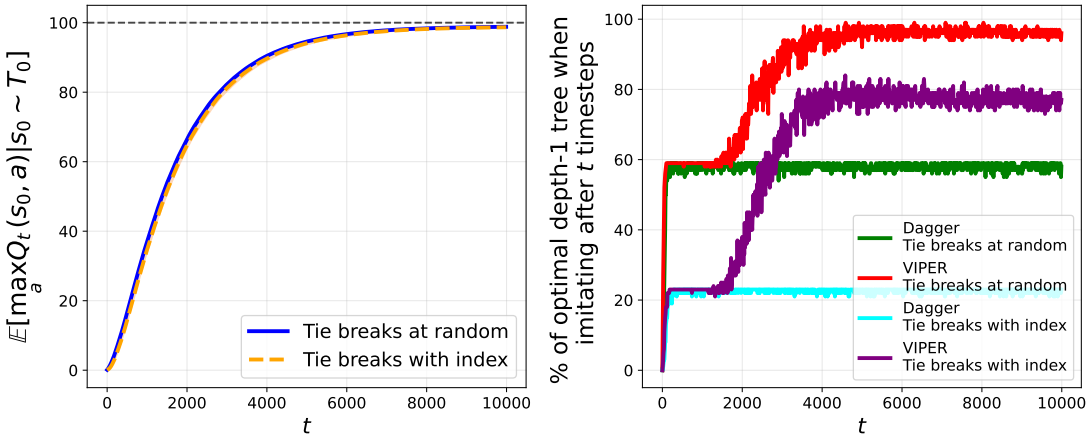


Figure 12 – Left, sample complexity curve of Q-learning with default hyperparameters on the 2×2 grid world MDP over 100 random seeds. Right, performance of indirect interpretable methods when imitating the greedy policy with a tree at different Q-learning stages.

thesis we study different decision tree learning algorithms in different settings. In the first part of the manuscript, we show that direct decision tree learning methods (cf. figure 6) struggle to find decision tree policies even for very simple sequential decision making problems. For that, we first reproduce the work from [124] that presents a formalism for learning decision tree policies that optimize the RL objective (cf. definition 4) directly using reinforcement learning, and then make connections with hardness results from the partially observable MDPs (POMDPs) literature [119, 39]. In the second part of the manuscript, we formulate decision tree induction for supervised learning as solving a sequential decision making problem. By formalizing decision tree induction for the supervised learning objective (cf. definition 2) as solving an MDP (cf. definition 3), we design novel algorithms that achieve very good performances. In the last part of the text, we lift our assumption about decision trees being intrinsically interpretable and study other model classes. In particular, we leverage the simplicity of indirect methods to imitate neural network experts with models from figure 4 and perform a large-scale empirical study of the interpretability-performances trade-offs on various sequential decision making tasks.

We summarize our results as follows :

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

Première partie

A Difficult Problem : Reinforcement Learning of Decision Tree Policies

Chapitre 1

Introduction

In this part of the manuscript, we show that direct reinforcement learning of decision tree policies for MDPs (cf. definition 3), i.e. learning a decision tree that optimizes the RL objective (cf. definition 4) is often very difficult. In particular, we provide some insights as to why it is so difficult and show that indirect imitation of a neural network policy (cf. section 4), while optimizing the imitation learning objective (cf. definition 8) rather than the RL one, often yields very good tree policies in practice.

This first part of the manuscript is organized as follows. In this chapter, we present Nicholay Topin and colleagues' framework for direct reinforcement learning of decision tree policies [124]. In chapter 2, we reproduce experiments from [124] where we compare direct deep reinforcement learning (cf. section 4) of decision tree policies to indirect imitation of neural network policies with decision trees for the simple CartPole MDP [11]. In chapter 3, we show that this direct approach is equivalent to learning a deterministic memoryless policy for partially observable MDP (POMDP)[119, 39]–which is a hard problem [74]–and show that this might be the main reason for failures. In chapter 4, we further support this claim by constructing special instances of such POMDPs where the observations contain all the information about hidden states, and show that in those cases, direct reinforcement learning of decision trees works well.

1.1 Learning decision tree policies for MDPs

In the introductory example (cf. section 4), we have shown that imitation learning algorithms that optimize the objective rather than the RL objective, are, unsurprisingly, prone to sub-optimality w.r.t. the latter objective. This motivates the study and deve-

lopment of *direct* decision tree policy learning algorithms. There already exists such algorithms that return decision tree policies optimizing the RL objective for a given MDP. Those algorithms either learn parametric trees or non-parametric trees.

Parametric trees are not "grown" from the root by iteratively adding internal or leaf nodes (cf. figure 7), but are rather "optimized" : the depth, internal nodes arrangement, and state features to consider in each node are fixed *a priori* and only the thresholds of each node are learned. This is similar to doing gradient descent on neural network weights. As the reader might have guessed, those parametric trees are advantageous in that they can be learned with the policy gradient [120]. In [115], [135] and [84], authors use PPO(cf. algorithm 7, [112]) to learn such differentiable decision trees that optimize directly the RL objective. In particular, [84] explicitly studies the gap in RL objective values between their direct optimization and the imitation learning algorithm VIPER (algorithm 9, [12]). While those methods return decision tree policies that optimize the RL objective well, in general a user cannot know *a priori* what a "good" tree policy structure should be for a particular MDP. It could be that the specified structure is too deep and pruning will be required after training or it could be that the tree structure is not expressive enough to encode a good policy, i.e. parametric trees cannot trade off interpretability and performances during training. Furthermore, authors from [84] show that extra stabilizing tricks, such as adaptive batch sizes, are required during training to outperform indirect imitation in terms of RL objective.

Non-parametric trees are the standard model in supervised learning. Greedy algorithms [20, 105, 104] are fast and return decision tree classifiers (or regressors) that offer good trade-offs between interpretability (depth, or number of nodes) and the supervised learning objective (cf. definition 2). On the other hand, to the best of our knowledge, there exists only one work studying non-parametric trees to optimize a trade-off between interpretability and the RL objective : Topin et. al. [124].

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

There also exists more specialized approaches that can return decision tree policies

only for very specific problem classes. In [81], authors prove that for maze-like MDPs, there always exists an optimal decision tree policy w.r.t. 4 and provide an algorithm to find it. Finally, in [134], authors study decision tree policies for planning in MDPs (cf. algorithm 2), i.e. when the transitions and rewards are known. In the next section we present IBMDPs as introduced in Topin et. al.[124].

1.2 Iterative bounding Markov decision processes

The key thing to know about IBMDPs is that they are, as their name suggests, MDPs (cf definition 3). Hence, IBMDPs admit an optimal deterministic Markovian policy that maximizes the RL objective. In this part we will assume that all the MDP we consider are MDPs with continuous state spaces (cf. section 4) with a finite set of actions, so we use bold fonts for states and observations as they are vector-valued. However all our results generalize to discrete states (in $\mathbb{Z}^m$) MDPs that we can factor using one-hot encodings. Given an MDP for which we want to learn a decision tree policy, the base MDP, IBMDP states are concatenations of the base MDP state features and some observations. Those observations are information about the base state features that are refined–"iteratively bounded"– at each step. Those observations essentially represent some knowledge about where some base state features lie in the state space. Actions available in an IBMDP are : 1. the actions of the base MDP, that change base state features, and 2. *information gathering* actions that change the aforementioned observations. Now, base actions in an IBMDP are rewarded like in the base MDP, this ensures that the RL objective w.r.t. the base MDP is encoded in the IBMDP reward. When taking an information gathering action, the reward is an arbitrary value such that optimizing the RL objective in the IBMDP is equivalent to optimizing some trade-off between interpretability and the RL objective in the base MDP.

Before showing how to get decision tree policies from IBMDP policies, we give a formal definition of IBMDPs following Topin et. al. [124].

Definition 9 (Iterative Bounding Markov decision process)**.** *Given an MDP $\mathcal{M}\langle S, A, R, T, T_0\rangle$ (cf. definition 3), an associated iterative bounding Markov decision process $\mathcal{M}_{IB}$ is a tuple :*

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

— *S are the base MDP state features. Base state features $\boldsymbol{s} = (s_1, \ldots, s_p) \in S$ are bounded :*

$s_j \in [L_j, U_j]$ *where* $\infty < L_j \leq U_j < \infty \forall 1 \leq j \leq p$.

— *O are observations. They represent bounds on the base state features :* $O \subsetneq S^2 = [L_1, U_1] \times \cdots \times [L_p, U_p] \times [L_1, U_1] \times \cdots \times [L_p, U_p]$. *So the complete IBMDP state space is* $S \times O$, *the concatenations of base state features and observations Given some base state features* $\boldsymbol{s} = (s_1, \ldots, s_p) \in S$ *and some observation* $\boldsymbol{o} = (L_1, U_1, \ldots, L_p, U_p)$, *an IBMDP state is* $\boldsymbol{s}_{IB} = (\overbrace{s_1, \ldots, s_p}^{\text{base state features}}, \underbrace{L_1, U_1, \ldots, L_p, U_p}_{\text{observation}})$.

— *A are the base MDP actions.*

— A_{info} *are information gathering actions (IGAs) of the form* $\langle j, v \rangle$ *where* j *is a state feature index* $1 \leq j \leq p$ *and* v *is a real number between* L_j *and* U_j. *So the complete action space of an IBMDP is the set of base MDP actions and information gathering actions* $A \cup A_{info}$.

— $R : S \times A \to \mathbb{R}$ *is the base MDP reward function.*

— ζ *is a reward signal for taking an information gathering action. So the IBMDP reward function is to get a reward from the base MDP if the action is a base MDP action or to get* ζ *if the action is an IGA action.*

— $T_{info} : S \times O \times (A_{info} \cup A) \to \Delta(S \times O)$ *is the transition function of IBMDPs : Given some observation* $\boldsymbol{o}_t = (L'_1, U'_1, \ldots, L'_p, U'_p) \in O$ *and base state features* $\boldsymbol{s}_t = (s'_1, s'_2, \ldots, s'_p)$ *if an IGA* $\langle j, v \rangle$ *is taken, the new observation is :*

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

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

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

1.2.1 From policies to trees

One can notice that information gathering actions (cf. definition 9) resemble the Boolean functions $1_{\{x_{\_,j} \leq v\}}$ that make up internal decision tree nodes (cf. figure 7). Indeed,

a policy taking actions in an IBMDP essentially builds a tree by taking sequences of IGAs (internal nodes) and then a base action (leaf node) and repeats this process over time. In particular, the IGA rewards ζ can be seen as a regularization or a penalty for interpretability : if ζ is very small compared to base rewards, a policy will try to take base actions as often as possible, i.e. build shallow trees with short paths between root and leaves.

Authors from [124] show that not all IBMDP policies are decision tree policies for the base MDP. In particular, they show that only deterministic policies depending solely on the observations of the IBMDP are guaranteed to correspond to decision tree policies for the base MDP.

Proposition 1 (Deterministic partially observable IBMDP policies are decision tree policies)**.** *Given a base MDP $\mathcal{M}\ \langle S, A, R, T, T_0\rangle$ and an associated IBMDP $\mathcal{M}_{IB}\ \langle S \times O, A \cup A_{info}, (R, \zeta), (T_{info}, T, T_0)\rangle$ (cf. definition 9), a deterministic partially observable policy $\pi_{po} : O \to A \cup A_{info}$ is a decision tree policy $\pi_{\mathcal{T}} : S \to A$ for the base MDP $\mathcal{M}$.*

Démonstration. (Sketch) algorithm 10 that takes as input a deterministic partially observable policy (cf. definition 1) for an IBMDP $\mathcal{M}_{IB}\ \langle S \times O, A \cup A_{info}, (R, \zeta), (T_{info}, T, T_0)\rangle$ (cf. definition 9), returns a decision tree policy $\pi_{\mathcal{T}}$ (cf. definition 4) for the base MDP $\mathcal{M}\ \langle S, A, R, T, T_0\rangle$ and always terminates unless the deterministic partially observable policy takes IGAs in every state. □

While the connections with partially observable MDPs [119, 39] is obvious, we defer the implications to chapter 3 as this connection was absent from the original IBMDP paper [124]. Next we present an IBMDP for the MDP of example 9.

1.2.2 Example : an IBMDP for a grid world

We re-formulate the example MDP (example 9) as an MDP with a finite number of vector valued states (x, y-coordinates). The states are $S = \{(0.5, 0.5), (0.5, 1.5), (1.5, 1.5), (1.5, 0.5)\} \subsetneq [0, 2] \times [0, 2]$. The actions are the cardinal directions $A = \{\rightarrow, \leftarrow, \downarrow, \uparrow\}$ that shift the states by one as long as the coordinates remain in the grid. The reward for taking any action is 0 except when in the bottom right state $(1.5, 0.5)$ which is an absorbing state : once in this state, you stay there forever. Standard optimal deterministic Markovian policies were presented for this MDP in example 9.

Suppose an associated IBMDP (definition 9) with two IGAs :

— $\langle x, 1\rangle$ that tests if $x \leq 1$

— $\langle y, 1\rangle$ that tests if $y \leq 1$

Algorithme 10 : Extract a Decision Tree Policy

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

The initial observation is always the grid bounds $\boldsymbol{o}_0 = (0, 2, 0, 2)$ because the base state features in the grid world are always in $[0, 2] \times [0, 2]$. There are only finitely many observations since with those two IGAs there are only nine possible observations that can be attained from $\boldsymbol{o}_0$ following the IBMDP transitions (cf. definition 9). For example when the IBMDP initial base state features are $\boldsymbol{s}_0 = (0.5, 1.5)$, and taking first $\langle x, 1\rangle$ then $\langle y, 1\rangle$ the corresponding observations are first $\boldsymbol{o}_{t+1} = (0, 1, 0, 2)$ and then $\boldsymbol{o}_{t+2} = (0, 1, 1, 2)$. The full observation set is $O = \{(0, 2, 0, 2), (0, 1, 0, 2), (0, 2, 0, 1), (0, 1, 0, 1), (1, 2, 0, 2), (1, 2, 0, 1), (1, 2, 1, 2), (0, 1, 1, 2), (0, 2, 1, 2)\}$. The transitions and rewards are given in definition (cf. definition 9).

In figure 1.1 we illustrate a trajectory in this IBMDP.

1.3 Summary

In this chapter, we presented the approach of Topin et. al. [124] to find decision tree policies that directly trade off interpretability and the RL objective rather than a surrogate imitation loss. In particular, Topin et. al. showed that deterministic partially observable policies for iterative bounding Markov decision processes are decision tree policies for MDPs (cf. definitions 9 and 1, algorithm 10, and example 1.1).

The great promise of Topin et. al. is that one can use any reinforcement learning

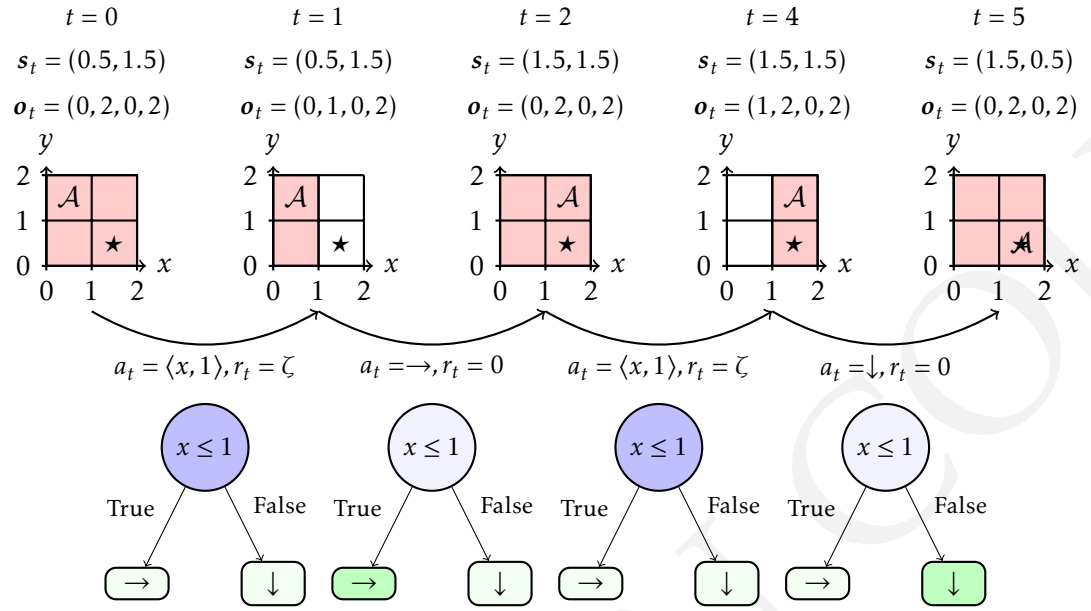


FIGURE 1.1 – An IBMDP trajectory when the base MDP is 2×2 grid world. In the top row, we write the visited base state features and observations, in the middle row, we graphically represent those, and in the bottom row, we present the corresponding decision tree policy traversal. $\mathcal{A}$ tracks the current state features $\boldsymbol{s}_t$ in the grid. The pink obstructions of the grid represent the current observations $\boldsymbol{o}_t$ of the base state features. When the pink covers the whole grid, the information contained in the observation could be interpreted as "the current state features could be anywhere in the grid". The more information gathering actions are taken, the more refined the bounds on the current base state features get. At $t = 0$, the base state features are $\boldsymbol{s}_0 = (0.5, 1.5)$. The initial observation is always the base MDP default state feature bounds, here $\boldsymbol{o}_0 = (0, 2, 0, 2)$ because the base state features are in $[0, 2] \times [0, 2]$. This means that the IBMDP state is $\boldsymbol{s}_{IB} = (0.5, 1.5, 0, 2, 0, 2)$. The first action is an IGA $\langle x, 1 \rangle$ that tests the feature x of the base state against the value 1 and the reward ζ. This transition corresponds to going through an internal node $x \leq 1$ in a decision tree policy as illustrated in the figure. At $t = 1$, after gathering the information that the x-value of the current base state is below 1, the observation is updated with the refined bounds $\boldsymbol{o}_1 = (0, 1, 0, 2)$, i.e. the pink area shrinks, and the base state features remain unchanged. The agent then takes a base action that is to move right. This gives a reward 0, resets the observation to the original base state feature bounds, and changes the features to $\boldsymbol{s}_2 = (1.5, 1.5)$. And the trajectory continues like this until the absorbing base state $\boldsymbol{s}_5 = (1.5, 0.5)$ is reached.

algorithm, e.g. PPO or DQN (cf. section 4), to learn deterministic partially observable policies. Indeed, such policies can be encoded as e.g. neural networks as long as they satisfy proposition 1 : this circumvents the non-differentiability limitations in direct learning of non-parametric decision tree policies. Once a deterministic partially observable policy is obtained, a decision tree policy that trades off naturally, the RL objective in the base MDP and, interpretability through the additional reward signal ζ (cf. definition 9), can be extracted with algorithm 10. We define an interpretable RL objective as follows :

Definition 10 (Interpretable RL objective)**.** *Given a base MDP $\mathcal{M}\langle S, A, R, T, T_0\rangle$ for which we want an interpretable policy, e.g. a decision tree, a discount factor $\gamma \in (0,1]$, some interpretability penalty ζ, and a set of information gathering actions A_{info}, the objective is to find a deterministic partially observable policy $\pi_{po} : O \to A \cup A_{info}$ (cf. definition 1) that maximizes :*

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R((\boldsymbol{s}_t, \boldsymbol{o}_t), a_t) \mid \boldsymbol{s}_0 \sim T_0, a_t = \pi_{po}(\boldsymbol{o}_t), \boldsymbol{s}_{t+1} \sim T(\boldsymbol{s}_t, a_t), \boldsymbol{o}_{t+1} \sim T(\boldsymbol{o_t}, a_t)\right]$$
$$= \mathbb{E}[V^{\pi_{po}}(\boldsymbol{s}_0, \boldsymbol{o}_0) | \boldsymbol{s}_0 \sim T_0]$$

With $V^{\pi_{po}}$ the value function (cf. definition 5) of π_{po} in the IBMDP $\mathcal{M}_{IB}$ $\langle S \times O, A \cup A_{info}, (R, \zeta), (T_{info}, T, T_0)\rangle$ (cf. definition 9).

After optimizing objective with, e.g. reinforcement learning (cf. section 4), we can use algorithm 10 to obtain a decision tree policy that trades off between interpretability and the RL objective for the base MDP of interest. This is exactly what we do in the next chapters.

Chapitre 2

Direct reinforcement learning of decision tree policies

In this chapter, we compare deep reinforcement learning of decision tree policies (cf. chapter 1) to imitation learning of decision tree policies (cf. section 4) for the CartPole MDP [11].

In particular, we attempt to reproduce the results from [124, table 1] in which authors constraint the solution space of decision tree policies to depth-2 trees. In the original results, authors find that deep reinforcement learning to solve the interpretable RL objective (cf. definition 10) and Dagger or VIPER (cf. section 4) to solve the imitation learning objective (cf. definition 8) find similar decision trees. We find that imitation learning, despite not directly optimizing the RL objective for CartPole, outperforms deep RL that optimizes the interpretable RL objective even though the deep RL approach directly optimizes the RL objective for CartPole (up to some trade-off with interpretability).

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

2.1.1 IBMDP formulation

Given a base MDP $\mathcal{M}\langle S, A, R, T, T_0\rangle$ (cf. definition 3), in order to define an IBMDP $\mathcal{M}_{IB}\langle S \times O, A \cup A_{info}, (R, \zeta), (T, T_0, T_{info})\rangle$ (cf. definition 9), the user needs to provide the set of information gathering actions A_{info} and the reward ζ for taking those. Authors of [124] propose to parametrize the set of IGAs with $j \times q$ actions $\langle j, v_k\rangle$ with v_k de-

pending on the current observation $\boldsymbol{o}_t = (L'_1, U'_1, \ldots, L'_j, U'_j, \ldots, L'_p, U'_p) : v_k = \frac{k(U'_j - L'_j)}{q+1}$. This parametric IGAs space keeps the discrete IBMDP action space at a reasonable size while providing a learning algorithm with varied IGAs to try.

For example, if we define an IBMDP with $q = 3$ for the grid world from Example 9, the grid world action space is augmented with six IGAs. At $t = 0$, recall that $\boldsymbol{o}_0 = (0, 2, 0, 2)$, so if an IGA is taken, e.g. $\langle 2, v_2 \rangle$, the effective IGA is $\langle j, v_2 = \frac{k(2-0)}{3+1} \rangle = \langle 1, 2 \rangle$ which in turn effectively corresponds to an internal decision tree node $y \leq 1$. If the current state y-feature value is 0.5, then the next observation at $t = 1$ is $\boldsymbol{o}_1 = (0, 2, 0, 1)$. At $t = 2$ if $a_t = \langle 2, v_2 \rangle$ again, it would be effectively $\langle j, v_2 = \frac{k(1-0)}{3+1} \rangle = \langle 2, 0.5 \rangle$. This would give the next observation at $t = 2$ $\boldsymbol{o}_2 = (0, 2, 0, 0.5)$ and so on.

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

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

We actually find that when $q + 1$, the parameter that defines threshold values in decision tree policy nodes (cf. definition 9), is a prime number, then as a direct consequence of the *Chinese Remainder Theorem* [1], the current tree depth is directly encoded in the current observation $\boldsymbol{o}_t$. Hence, when $q + 1$ is prime, we can control the depth through either transitions or rewards without tracking the tree depth.

We will try various ζ, various q, and various depth control in our experiments but first we describe the reinforcement learning algorithms used in [124].

2.1.2 Modified deep reinforcement learning algorithms

Authors of [124] use two deep reinforcement learning baselines (cf. section 4) to which they apply some modifications in order to learn partially observable policies as required by proposition 1 and by the interpretable RL objective (cf. definition 10).

1. https://en.wikipedia.org/wiki/Chinese_remainder_theorem

Authors modify the standard PPO (cf. algorithm 12) and train a neural network policy $O \to A \cup A_{info}$ while the neural network value function is $S \times O \to A \cup A_{info}$. A similar modification is done to DQN to return a partially observable policy (cf. algorithm 11). The trained Q-function is approximated with a neural network $O \to \mathbb{R}^{|A \cup A_{info}|}$ rather than $S \times O \to \mathbb{R}^{|A \cup A_{info}|}$. In this modified DQN, the temporal difference error target for the Q-function $O \to A \cup A_{info}$ is approximated by a neural network $S \times O \to A \cup A_{info}$ that is in turn trained by bootstrapping the temporal difference error with itself.

Those two variants of DQN and PPO have first been introduced in [98] for robotic tasks with partially observable components, under the name "asymmetric" actor-critic. Asymmetric RL algorithms that have policy and value estimates using different information from a POMDP [119, 39] were later studied theoretically to solve POMDPs in Baisero's work [8, 7]. The connections from Deep RL in IBMDPs for objective is absent from [124] and we defer their connections to direct interpretable reinforcement learning to the next chapter as our primary goal is to reproduce [124] *as is*.

Next, we present the precise experimental setup we use to reproduce [124, table 1] in order to study direct deep reinforcement learning of decision tree policies for the CartPole MDP.

2.2 Experimental setup

2.2.1 (IB)MDP

We use the exact same base MDP and associated IBMDPs for our experiments as [124] except when mentioned otherwise.

Base MDP The task at hand is to optimize the RL objective (cf. definition 4) with a decision tree policy for the CartPole MDP [11]. At each time step a learning algorithm observes the cart's position and velocity and the pole's angle and angular velocity, and can take action to push the CartPole left or right. While the CartPole is roughly balanced, i.e., while the cart's angle remains in some fixed range, the agent gets a positive reward. If the CartPole is out of balance, the MDP transitions to an absorbing terminal state and gets 0 reward forever. Like in [124], we use the gymnasium `CartPole-v0` implementation [125] of the CartPole MDP in which trajectories are truncated after 200 timesteps making the maximum cumulative reward, i.e. the optimal value of the RL objective when $\gamma = 1$, to be 200. The state features of the CartPole MDP are in $[-2, 2] \times [-2, 2] \times [-0.14, 0.14] \times [-1.4, 1.4]$.

Algorithme 11 : Modified Deep Q-Network

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

Result : Partially observable deterministic policy π_{po}

Initialize partially observable Q-network parameters θ

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

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

for *each episode* **do**

  Initialize base state features $\boldsymbol{s}_0 \sim T_0$

  Initialize observation $\boldsymbol{o}_0 = (L_1, U_1, \ldots, L_p, U_p)$

  for *each step* t **do**

    Choose action a_t using ϵ-greedy : $a_t = \text{argmax}_a\, Q_\theta(\boldsymbol{o}_t, a)$ with prob. $1 - \epsilon$

    Take action a_t, observe r_t

    Store transition $(\boldsymbol{s}_t, \boldsymbol{o}_t, a_t, r_t, \boldsymbol{s}_{t+1})$ in $\mathcal{B}$

    Sample random batch $(\boldsymbol{s}_i, \boldsymbol{o}_i, a_i, r_i, \boldsymbol{s}_{i+1}) \sim \mathcal{B}$

    $a' = \underset{a}{\text{argmax}} Q_\theta(\boldsymbol{o}_i, a)$

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

    $\phi \leftarrow \phi - \alpha \nabla_\phi (Q_\phi(\boldsymbol{s}_i, a_i) - y_i)^2$ // Update Q-network

    $\theta \leftarrow \theta - \alpha \nabla_\theta (Q_\theta(\boldsymbol{o}_i, a_i) - y_i)^2$ // Update partially observable Q-network

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

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

    end

    $\boldsymbol{s}_t \leftarrow \boldsymbol{s}_{t+1}$

    $\boldsymbol{o}_t \leftarrow \boldsymbol{o}_{t+1}$

  end

end

$\pi_{po}(\boldsymbol{o}) = \text{argmax}_a\, Q_\theta(\boldsymbol{o}, a)$ // Extract greedy policy

Algorithme 12 : Modified Proximal Policy Optimization

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

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

We try two different approaches to limit the depth of decision tree policies to be at most 2 : terminating trajectories if the agent takes too much information gathering in a row or simply giving a reward of -1 to the agent every time it takes an information gathering action past the depth limit. In practice, we could have tried the action masking approach but we want to abide to the MDP formalism in order to properly understand direct interpretable approaches. We will also try IBMDPs where we do not limit the maximum depth for completeness.

2.2.2 Baselines

Modified DQN as mentioned above, authors use the modified version of DQN from algorithm 11. We use the exact same hyperparameters for modified DQN as the authors when possible. We use the same layers width (128) and number of hidden layers (2), the same exploration strategy (ϵ-greedy with linearly decreasing value ϵ between 0.5 and 0.05 during the first 10% of the training), the same replay buffer size (10^6) and the same number of transitions to be collected randomly before doing value updates (10^5). We also try to use more exploration during training (change the initial ϵ value to 0.9). We use

the same optimizer (RMSprop with hyperparameter 0.95 and learning rate 2.5×10^{-4}) to update the Q-networks. Authors did not share which DQN implementation they used so we use the stable-baselines3 one [107]. Authors did not share which activation functions they used so we try both tanh and relu.

Modified PPO for the modified PPO algorithm (cf. algorithm 12), we can exactly match the authors hyperparameters since they use the open source stable-baselines3 implementation of PPO. We match training budgets : we train modified DQN on 1 million timesteps and modified PPO on 4 million timesteps.

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

Indirect methods We also compare modified RL algorithm to imitation learning (cf. section 4). To do so, we use VIPER or Dagger (cf. algorithms 8 and 9) to imitate greedy neural network policies obtained with standard DQN learning directly on CartPole. And we use Dagger to imitate neural network policies obtained with the standard PPO learning directly on CartPole.

For each indirect method, we imitate the neural network experts by fitting decision trees on 10000 expert transitions using the CART (cf. algorithm 1) implementation from scikit-learn [97] with default hyperparameters and maximum depth of 2 like in [124].

2.2.3 Metrics

The key metric of this section is performance when controlling the CartPole, i.e, the average *undiscounted* cumulative reward of a policy on 100 trajectories (RL objective with $\gamma = 1$).

For modified RL algorithms that learn a partially observable policy (or Q-function) in an IBMDP, we periodically extract the policy (or Q-function) and use algorithm 10 to extract a decision tree for the CartPole MDP. We then evaluate the tree on 100 independent trajectories in the MDP and report the mean undiscounted cumulative reward.

For standard RL applied to IBMDPs, since we can't deploy learned policies directly to the base MDP as the state dimensions mismatch–such policies are $S \times O \to A \cup A_{info}$ but the MDP states are in S–we periodically evaluate those IBMDP policies in a copy

of the IBMDP in which we fix $\zeta = 0$ ensuring that the copied IBMDP undiscounted cumulative rewards only account rewards from the CartPole MDP (non-zero rewards in the IBMDP only occur when a reward from the base MDP is given, i.e. when $a_t \in A$ in the IBMDP (cf. definition 9)). Similarly, we do 100 trajectories of the extracted policies in the copied IBMDP and report the average undiscounted cumulative reward.

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

Since imitation learning baselines train offline, i.e, on a fixed dataset, their performances cannot directly be reported on the same axis as RL baselines. For that reason, during the training of a standard RL baseline, we periodically extract the trained neural policy/Q-function that we consider as the expert to imitate. Those experts are then imitated with VIPER or Dagger using 10 000 newly generated transitions and the fitted decision tree policies are then evaluated on 100 CartPole trajectories. We do not report the imitation learning objective values during VIPER or Dagger training. Every single combination of IBMDP and Modified RL hyperparameters is run 20 times. For standard RL on either an IBMDP or an MDP, we use the paper original hyperparameters when they were specified, with depth control using negative rewards, tanh() activations. We use 20 individual random seeds for every experiment in this chapter. Next, we present our results when reproducing [124, table 1].

2.3 Results

2.3.1 How well do modified deep RL baselines learn in IBMDPs?

On figure 2.3.1, we observe that modified DQN can learn in IBMDPs–the curves have an increasing trend–but we also observe that modified DQN finds poor decision tree policies for the CartPole MDP in average–the curves flatten at the end of the x-axis and have low y-values. In particular, the highest final y-value, among all the learning curves that could possibly correspond to the original paper modified DQN, correspond to poor performances on the CartPole MDP.

On figure 2.3.1, we observe that modified PPO finds decision tree policies with almost 150 cumulative rewards towards the end of training. The performance difference with modified DQN could be because we trained modified PPO longer, like in the original paper.

However it could also be because DQN-like algorithms with those hyperparameters struggle to learn in CartPole (IB)MDPs. Indeed, we notice that for DQN-like baselines,

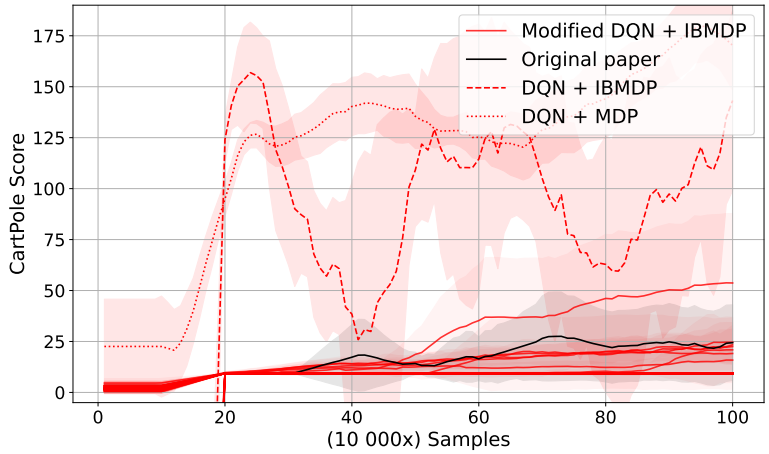


FIGURE 2.1 – Variations of modified DQN and DQN (cf. table A.2), on different CartPole IBMDPs (cf. table A.1). We give different line-styles for the learning curves of DQN applied directly on the CartPole MDP and DQN applied on the IBMDP to learn standard Markovian policies. Since there are multiple possible candidates for the original paper hyperparameters, we choose to color the (modified DQN variant, IBMDP variant) pair that resulted in the best decision tree policy on the CartPole MDP among the instances that could match the original paper. Shaded areas represent the confidence interval at 95% at each measure on the y-axis.

learning seems difficult in general independently of the setting.

On figures 2.3.1 and 2.3.1, we observe that standard RL baselines (RL + IBMDP and RL + MDP), learn better CartPole policies in average than their modified counterparts that learn partially observable policies (1). On figure 2.3.1, it is clear that for the standard PPO baselines, learning is super efficient and algorithms learn optimal policies with reward 200 in few thousands steps.

In tables 2.1 and 2.2 we report the top-5 hyperparameters for Modified RL baselines when learning partially observable IBMDP policies in terms of extracted decision tree policies performances in the CartPole MDP.

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

On figure 2.3, we isolate the best performing algorithms instantiations that learn decision tree policies for the CartPole MDP. We compare the best modified DQN and modified PPO to imitation learning baselines that use the surrogate imitation objective (cf. definition 8) to find CartPole decision tree policies. We find that despite having

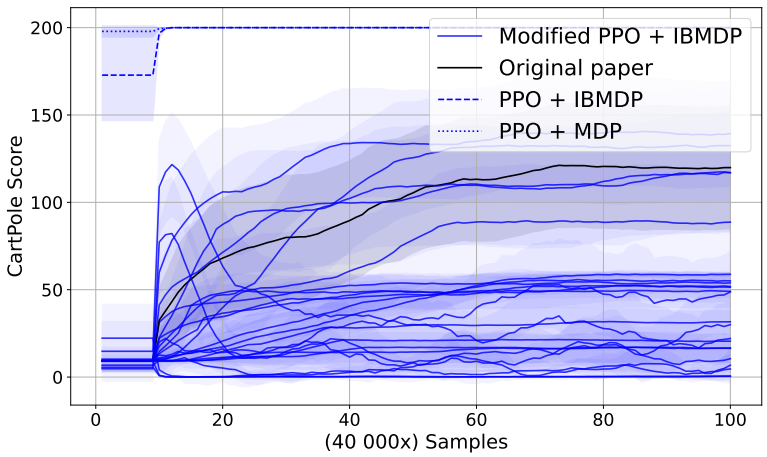


Figure 2.2 – Variations of modified PPO and PPO (cf. table A.3), on different CartPole IBMDPs (cf. table A.1). We give different line-styles for the learning curves of PPO applied directly on the CartPole MDP and PPO applied on the IBMDP to learn standard Markovian policies. Since there are multiple possible candidates for the original paper hyperparameters, we choose to color the (modified PPO variant, IBMDP variant) pair that resulted in the best decision tree policy on the CartPole MDP among the instances that could match the original paper. Shaded areas represent the confidence interval at 95% at each measure on the y-axis.

Tableau 2.1 – Top 5 hyperparameter configurations for modified DQN + IBMDP, bold font represent the original paper hyperparameters.

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

Tableau 2.2 – Top 5 hyperparameter configurations for modified PPO + IBMDP, bold font represent the original paper hyperparameters.

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

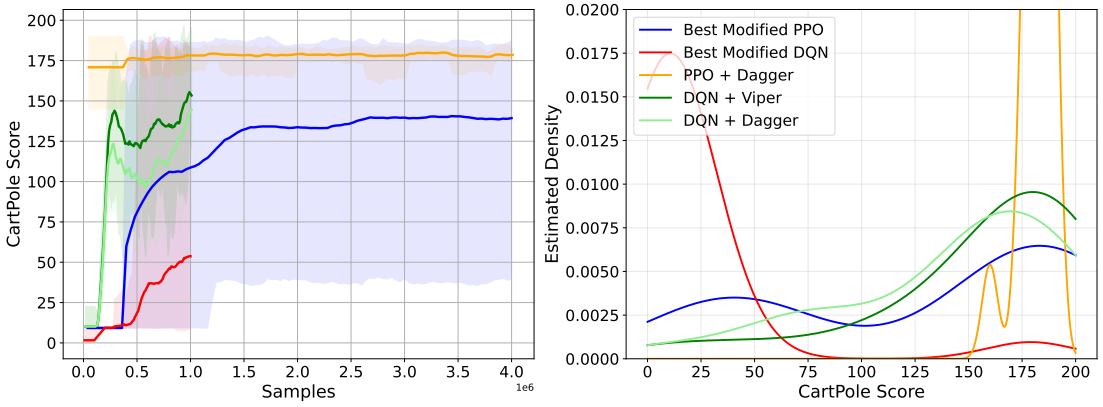


FIGURE 2.3 – (left) Mean performance of the best–w.r.t. the RL objective for CartPole–modified RL + IBMDP combination. Shaded areas represent the min and max performance over the 20 seeds during training. (right) Corresponding scores distribution of the final decision tree policies performances w.r.t. the RL objective for CartPole.

poor performances in *average*, the modified deep reinforcement learning baselines can find very good decision tree policies as shown by the min-max shaded areas on the left of figure 2.3 and the corresponding estimated density of learned trees performances. However this is not desirable, a user typically wants an algorithm that can consistently find good decision tree policies. As shown by the estimated densities, indirect methods consistently find good decision tree policies (the higher modes of distributions are on the right of the plot). On the other hand, the decision tree policies returned by direct RL methods seem equally distributed on both extremes of the scores.

On figure 2.4, we present the best decision tree policies for CartPole returned by modified DQN and modified PPO. We used algorithm 10 to extract 20 trees from the 20 partially observable policies returned by the modified deep reinforcement learning algorithms over the 20 training seeds. We then plot the best tree for each baseline. Those trees get an average RL objective of roughly 175. Similarly, we plot a representative tree for imitation learning baseline as well as a tree that is optimal for CartPole w.r.t. the RL objective obtained with VIPER. Unlike for direct methods, the trees returned by imitation learning are extremely similar across seeds. In particular they often only vary in the scalar value used in the root node but in general have the same structure and test the angular velocity. On the other hand the most frequent trees across seeds returned by modified RL baselines are “trivial” decision tree policies that either repeat the same base action forever or repeat the same IGA (cf. definition 9) forever.

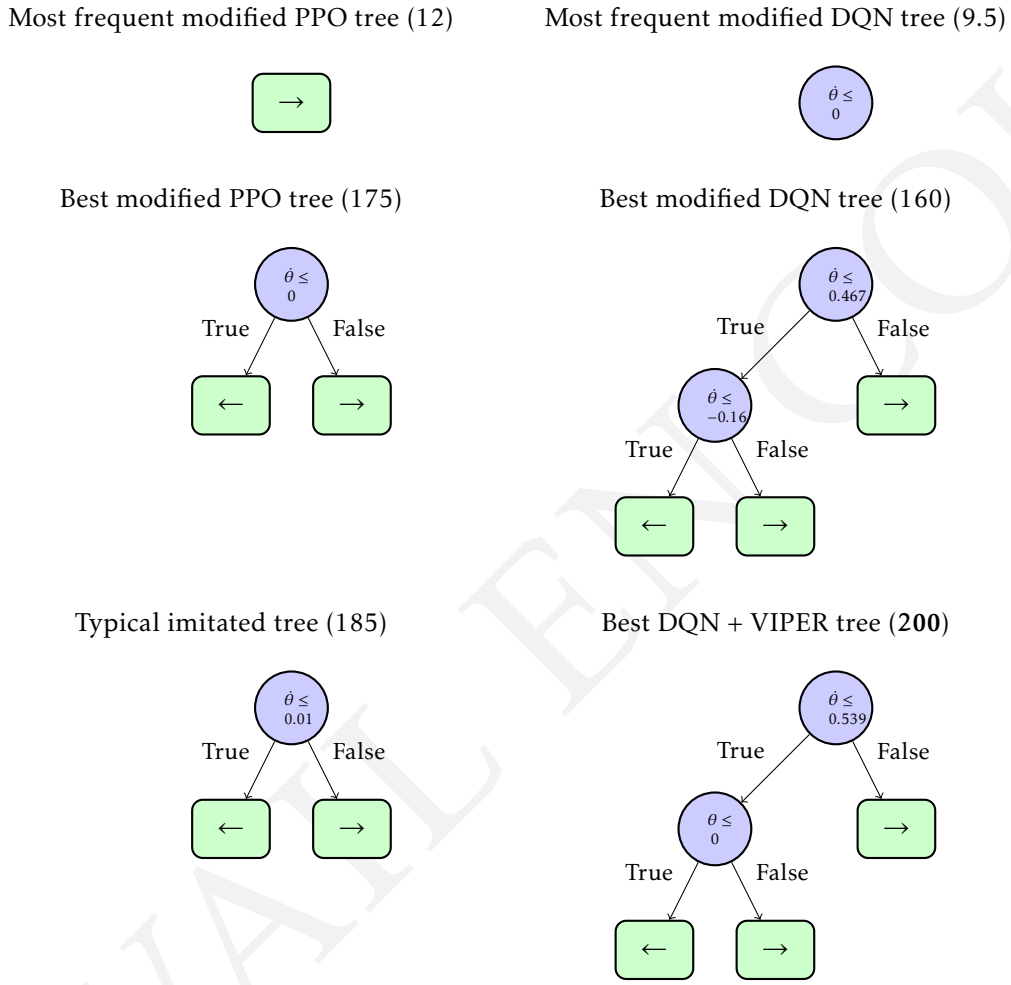


Figure 2.4 – Trees obtained by modified deep RL in IBMDPs against trees obtained with imitation (RL objective value). θ and $\dot{\theta}$ are respectively the angle and the angular velocity of the pole

2.4 Discussion

We have shown that compared to learning non-interpretable but standard Markovian neural network policies for the base MDP or some associated IBMDP, reinforcement learning of partially observable policies in IBMDP is less efficient (cf. figures 2.3.1 and 2.3.1). As a consequence, only a handful of modified RL runs are able to learn decision tree policies that are on par with imitated trees (cf. figure 2.3).

In the next chapter, we highlight the connections between direct interpretable RL (cf. definition 10) and POMDPs to get insights on the hardness of direct reinforcement learning of decision trees.

Chapitre 3

Limits of direct reinforcement learning of decision tree policies

From the previous chapter, we know that to directly learn decision tree policies that optimize the RL objective (cf. definition 4) for an MDP (cf. definition 3), one can learn a deterministic partially observable policy that optimizes the interpretable RL objective (cf. definition 10) in an IBMDP (cf. definition 9 and proposition 1). Such problems are classical instances of partially observable Markov decision processes (POMDPs) [119, 39]. This connection with POMDPs brings novel insights to direct reinforcement learning of decision tree policies. In this chapter, all the decision processes have a finite number of vector-valued states and observations. Hence we will use bold fonts for states and observations but we can still use summations rather than integrals when required.

3.1 Partially observable iterative Markov decision processes

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

Definition 11 (Partially observable Markov decision process)**.** *A partially observable Markov decision process is a tuple* $\langle X, A, O, R, T, T_0, \Omega \rangle$ *where :*

- X *is the hidden state space.*
- A *is a finite set of actions.*
- O *is a set of observations.*
- $T : X \times A \to \Delta(X)$ *is the transition function, where* $T(\boldsymbol{x}_t, a, \boldsymbol{x}_{t+1}) = P(\boldsymbol{x}_t | \boldsymbol{x}_{t+1}, a)$ *is the probability of transitioning to state* $\boldsymbol{x}_t$ *when taking action* a *in state* $\boldsymbol{x}$

— T_0 *: is the initial distribution over states.*
— $\Omega : X \to \Delta(O)$ *is the observation function, where* $\Omega(\boldsymbol{o}, a, \boldsymbol{x}) = P(\boldsymbol{o}|\boldsymbol{x}, a)$ *is the probability of observing* $\boldsymbol{o}$ *in state* $\boldsymbol{x}$
— $R : X \times A \to \mathbb{R}$ *is the reward function, where* $R(\boldsymbol{x}, a)$ *is the immediate reward for taking action a in state* $\boldsymbol{x}$

Note that $\langle X, A, R, T, T_0 \rangle$ *defines an MDP.*

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

Definition 12 (Partially observable iterative bounding Markov decision process)**.** *a partially observable iterative bounding Markov decision process* $\mathcal{M}_{POIB}$ *is a tuple :*

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

Where $\langle S \times O, A \cup A_{info}, (R, \zeta), (T, T_0, T_{info}) \rangle$ *is an IBMDP 9. The transition function* Ω *maps concatenation of state features and observations–IBMDP states–to observations,* $\Omega : S \times O \to O$*, with* $P(\boldsymbol{o}|(\boldsymbol{s}, \boldsymbol{o})) = 1$

One can see POIBMDPs as particular instances of POMDPs where the observation function simply applies a mask over some features of the hidden state. This setting has other names in the literature. For example, POIBMDPs are mixed observability MDPs [4] with base MDP state features as the *hidden variables* and feature bounds as *visible* variables. POIBMDPs can also be seen as non-stationary MDPs (N-MDPs) [116] in which there is one different transition function per base MDP state : these are called hidden-mode MDPs [28]. Following [116] we can write the value of a deterministic partially observable policy $\pi : O \to A \cup A_{info}$ in observation $\boldsymbol{o}$.

Definition 13 (Partially observable value function)**.** *In a POIBMDP (cf. definition 12), the expected cumulative discounted reward of a deterministic partially observable policy* $\pi : O \to A \cup A_{info}$ *starting from observation o is* $V^{\pi}(\boldsymbol{o})$ *:*

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

with $P^{\pi}((\boldsymbol{s}, \boldsymbol{o}')|\boldsymbol{o})$ *the asymptotic occupancy distribution (see [116, section 4] for the full definition) of the hidden POIBMDP state* $(\boldsymbol{s}, \boldsymbol{o}')$ *given the partial observation o and* $V^{\pi}((s, o'))$

the classical state-value function 5. We abuse notation and denote both values of observations and values of states by V since the function input is not ambiguous.

The asymptotic occupancy distribution is the probability of a policy π to arrive in $(s, \boldsymbol{o}')$ while observing $\boldsymbol{o}$ in some trajectory. In this chapter, we use reinforcement learning to train decision tree policies for MDPs by seeking deterministic partially observable policies that optimize the interpretable RL objective (cf. definition 10) in POIBMDPs (cf. definition 12).

The goal of the following sections is to see if, unlike indirect approaches tested in section 4 (cf. figures 11 and 12), direct approaches can consistently learn the depth-1 decision tree policies that have good interpretability-performance trade-offs for the 2×2 grid world from example 9 (cf. figures 10 and 3.1). The direct approach optimizes the interpretable RL objective 10 in POIBMDPs (cf. definition 12). We will use reinforcement learning to learn deterministic partially observable policies for the IBMDP from example 1.1 re-written as a POIBMDP. Next we show how to choose γ and ζ in the POIBMDP such that the optimal deterministic partially observable policies w.r.t. the interpretable RL objective correspond exactly to the depth-1 decision tree policies with good interpretability-performance trade-offs. Hence, if we find that RL can consistently find optimal policies w.r.t. interpretable RL objective, it means that this direct approach can consistently find the depth-1 decision tree policies with good interpretability-performance trade-off that indirect approaches could not consistently find.

3.2 Constructing POIBMDPs which optimal solutions are depth-1 decision tree policies

Because we know all the base states, all the observations, all the actions, all the rewards and all the transitions of our POIBMDP (cf. example 1.1), we can compute exactly the values of different deterministic partially observable policies given, ζ the reward for IGAs, and γ the discount factor. Each of those policies can be one of the trees illustrated in figure 3.1 :

— $\pi_{\mathcal{T}_0}$: a depth-0 tree equivalent to always taking the same base action
— $\pi_{\mathcal{T}_1}$: a depth-1 tree equivalent alternating between an IGA and a base action
— $\pi_{\mathcal{T}_u}$: an unbalanced depth-2 tree that sometimes takes two IGAs then a base action and sometimes a an IGA then a base action
— $\pi_{\mathcal{T}_2}$: a depth-2 tree that alternates between taking two IGAs and a base action

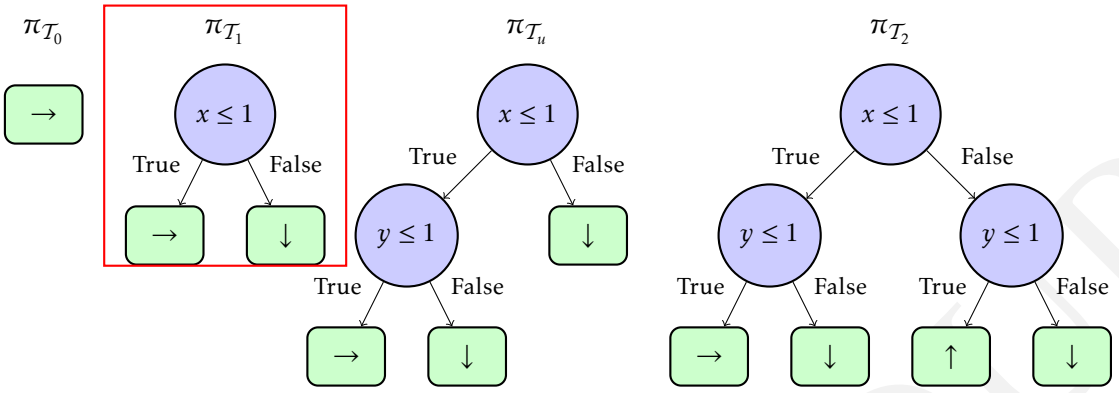


FIGURE 3.1 – For each decision tree structure, e.g., depth-1 or unbalanced depth-2, we illustrate a decision tree which maximizes the RL objective (cf. definition 4) in the grid world MDP.

— an infinite "tree" that only takes IGAs

Furthermore, because from [116] we know that for POMDPs, stochastic partially observable policies can sometimes get better expected discounted rewards than deterministic partially observable policies, we also compute the value of the stochastic policy that randomly alternates between two base actions : $\rightarrow$ and $\downarrow$. Those two base actions always lead to the goal state (cf. figure 9).

Proposition 2 (Depth-1 decision tree objective value)**.** *The interpretable RL objective value (cf. definition 10) of depth-1 decision tree policies with good interpretability–performance trade-offs in the grid world MDP (cf. figures 10 and 3.1) is* $\frac{4\zeta+\gamma+2\gamma^3+\gamma^5}{4(1-\gamma^2)}$.

We defer the lengthy proofs of this proposition, as well as the interpretable RL objective values of other decision tree policies, to the appendix A.2.

We can now plot, in figure 3.2, the interpretable RL objective values of the different partially observable policies as functions of ζ when we fix $\gamma = 0.99$. When $\gamma = 0.99$, despite objective values being very similar for the depth-1 and unbalanced depth-2 tree, we now know from the green shaded area that a depth-1 tree is the optimal one, w.r.t. the interpretable RL objective, deterministic partially observable POIBMDP policy for $0 < \zeta < 1$.

Let us now define a POIBMDP with the grid world (cf. figure 9) as the base MDP, with IGAs as in the IBMDP from example 1.1, with $\gamma = 0.99$ and $0 < \zeta < 1$ and verify if RL can learn the optimal deterministic partially observable policies w.r.t. the interpretable RL objective, which are equivalent to depth-1 decision tree policies, in this very controlled experiment.

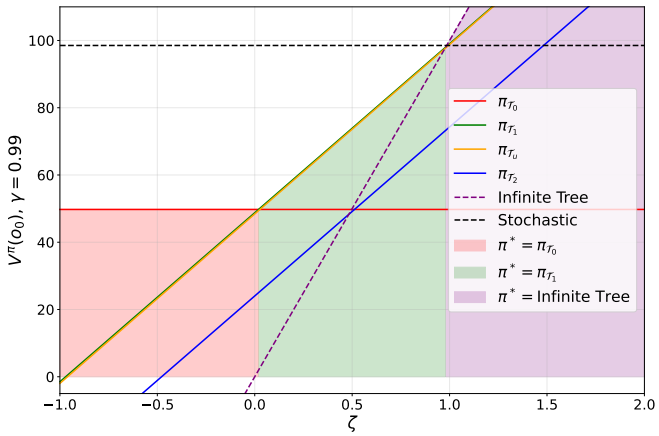


FIGURE 3.2 – Interpretable RL objective values (cf. definition 10) of different partially observable policies as functions of ζ. Shaded areas show the optimal *deterministic* partially observable policies in different ranges of ζ values.

3.3 Reinforcement learning in PO(IB)MDPs

In general, the policy that maximizes the RL objective (cf. definition 4) in a POMDP (cf. definition 11) maps "belief states" or observations histories [39] to actions. Hence, those policies are not solutions to our problem since we require that policies depend only on the current observation. If we did not have this constraint, we could apply any standard RL algorithm to solve POIBMDPs by seeking such policies because both histories and belief states are sufficient statistics for POMDPs hidden states [39, 64].

In particular, the problem of finding the optimal deterministic partially observable policies for POMDPs is NP-HARD, even with full knowledge of transitions and rewards [74, section 3.2]. It means that it is impractical to enumerate all possible policies and take the best one. For even moderate-sized POMDPs, a brute-force approach would take a very long time since there are $|A|^{|O|}$ deterministic partially observable policies. Hence it is interesting to study reinforcement learning for finding the best deterministic partially observable policy since it would not search the whole solution space. However applying RL to our interpretable RL objective (cf. definition 10) is non-trivial.

In [116, Fact 2], authors show that the optimal partially observable policy can be stochastic. Hence, policy gradient algorithms [120]–that return stochastic policies–are to avoid since we seek the best *deterministic* policy. Furthermore, the optimal deterministic partially observable policy might not maximize all the values of all observations simultaneously [116, Fact 5] which makes it difficult to use TD-learning (cf. algorithms 3

and 4). Indeed, doing a TD-learning update of one partially observable value 13 with, e.g. Q-learning, can change the value of *all* other observations in an uncontrollable manner because of the dependence in $P^\pi((s, \boldsymbol{o}')|\boldsymbol{o})$ (cf. definition 13). Interestingly, those two challenges of learning in POMDPs described in [116] are visible in figure 3.2. First, there is a whole range of ζ values for which the optimal partially observable policy is stochastic. Second, for e.g. $\zeta = 0.5$, while a depth-1 tree is the optimal deterministic partially observable policy, the value of state $(\boldsymbol{s}_2, \boldsymbol{o}_0) = (1.5, 1.5, 0, 2, 0, 2)$ is not maximized by this partially observable policy but by the sub-optimal policy that always goes down.

Despite those hardness results, empirical results of applying RL to POMDPs by naively replacing $\boldsymbol{x}$ by $\boldsymbol{o}$ in Q-learning or Sarsa, has already demonstrated successful in practice [75]. More recently, the framework of Baisero et. al. called asymmetric RL [8, 7] has also shown promising results to learn POMDP solutions. Asymmetric RL algorithms train a model–a policy or a value function–depending on hidden state (only available at train time) and a history dependent (or observation dependent) model. The history or observation dependent model serves as target or critic to train the hidden state dependent model. The history dependent (or observation dependent) model can thus be deployed in the POMDP after training since it does not require access to the hidden state to output actions. In algorithms 13 and 14 we present asymmetric Q-learning and asymmetric Sarsa : given a POMDP, both train a partially observable Q-function $Q : O \times A \to \mathbb{R}$ and a Q-function $U : X \times A \to \mathbb{R}$.

In [58], authors introduce a policy search algorithm 4 that learns a (stochastic) policy $\pi : O \to \Delta(A)$ and a critic $V : X \to \mathbb{R}$ using Monte Carlo estimates to guide policy improvement. We write this algorithm that we call JSJ (for the authors names Jaakkola, Singh, Jordan) in algorithm 15. JSJ is equivalent to a tabular asymmetric policy gradient algorithm (cf. algorithm 5).

Until recently, the benefits of asymmetric RL over standard RL was only shown empirically and only for history-dependent models. The work of Gaspard Lambrechts [65] proves that some asymmetric RL algorithms learn better history-dependent **or** partially observable policies for solving POMDPs. This is exactly what we wish for. However, those algorithms are not practical because they require estimation of the asymptotic occupancy distribution $P^\pi((s, \boldsymbol{o}')|\boldsymbol{o})$ (cf. definition 13) for candidate policies which in turn would require to gather a lot of on-policy samples. We leave it to future work to use those algorithms that combine asymmetric RL and estimation of future visitations since those results are contemporary to the writing of this manuscript.

Note that, in the previous chapter, modified DQN (cf. algorithm 11) and modified PPO (cf. algorithm 12) are respectively asymmetric DQN and asymmetric PPO from [8,

7]. In the next section, we use (asymmetric) RL to optimize the interpretable RL objective in POIBMDPs.

Algorithme 13 : Asymmetric Q-Learning

Data : POMDP $\mathcal{M}_{po} = \langle X, O, A, R, T, T_0, \Omega \rangle$, learning rates α_u, α_q, exploration rate ϵ
Result : $\pi : O \to A$
Initialize $U(\boldsymbol{x}, a) = 0$ for all $x \in X, a \in A$
Initialize $Q(\boldsymbol{o}, a) = 0$ for all $o \in O, a \in A$
for *each episode* **do**
  Initialize state $x_0 \sim T_0$
  Initialize observation $\boldsymbol{o}_0 \sim \Omega(\boldsymbol{x}_0)$
  for *each step* t **do**
    Choose action a_t using ϵ-greedy : $a_t = \text{argmax}_a Q(\boldsymbol{o}_t, a)$ with prob. $1 - \epsilon$
    Take action a_t, observe $r_t = R(\boldsymbol{x}_t, a_t)$, $x_{t+1} \sim T(x_t, a_t)$, and $\boldsymbol{o}_{t+1} \sim \Omega(\boldsymbol{x}_{t+1})$
    $y \leftarrow r + \gamma U(\boldsymbol{x}_{t+1}, \text{argmax}_{a'} Q(\boldsymbol{o}_{t+1}, a'))$ // TD target
    $U(\boldsymbol{x}_t, a_t) \leftarrow (1 - \alpha_u) U(\boldsymbol{x}_t, a_t) + \alpha_u y$
    $Q(\boldsymbol{o}_t, a_t) \leftarrow (1 - \alpha_q) Q(\boldsymbol{o}_t, a_t) + \alpha_q y$
    $x_t \leftarrow \boldsymbol{x}_{t+1}$
    $\boldsymbol{o}_t \leftarrow \boldsymbol{o}_{t+1}$
  end
end
$\pi(o) = \text{argmax}_a Q(\boldsymbol{o}, a)$ // Extract greedy policy

In the next section, we apply asymmetric and standard RL algorithms to the problem of learning the optimal depth-1 tree for the grid world MDP (cf. section 3.2) by optimizing the interpretable RL objective in POIBMDPs.

3.4 Results

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

3.4.1 Experimental setup

Baselines : we consider two groups of RL algorithms. The first group is standard tabular RL algorithms to optimize the interpretable RL objective in POIBMDPs ; Q-learning, Sarsa, and Policy Gradient with a softmax policy (cf. section 4, algorithms 3, 4, and 5). In theory the Policy Gradient algorithm should not be a good candidate for our

Algorithme 14 : Asymmetric Sarsa

Data : POMDP $\mathcal{M}_{po} = \langle X, O, A, R, T, T_0, \Omega \rangle$, learning rates α_u, α_q, exploration rate ϵ
Result : $\pi : O \to A$
Initialize $U(x, a) = 0$ for all $x \in X, a \in A$
Initialize $Q(o, a) = 0$ for all $o \in O, a \in A$
for *each episode* **do**
  Initialize state $x_0 \sim T_0$
  Initialize observation $\boldsymbol{o}_0 \sim \Omega(x_0)$
  Choose action a_0 using ϵ-greedy : $a_0 = \text{argmax}_a Q(\boldsymbol{o}_0, a)$ with prob. $1 - \epsilon$
  for *each step t* **do**
    Take action a_t, observe $r_t = R(x_t, a_t)$, $x_{t+1} \sim T(x_t, a_t)$, and $\boldsymbol{o}_{t+1} \sim \Omega(x_{t+1})$
    Choose action a_{t+1} using ϵ-greedy : $a_{t+1} = \text{argmax}_a Q(\boldsymbol{o}_{t+1}, a)$ with prob. $1 - \epsilon$
    $y \leftarrow r + \gamma U(x_{t+1}, a_{t+1})$ // TD target using actual next action
    $U(x_t, a_t) \leftarrow (1 - \alpha_u) U(x_t, a_t) + \alpha_u y$
    $Q(\boldsymbol{o}_t, a_t) \leftarrow (1 - \alpha_q) Q(\boldsymbol{o}_t, a_t) + \alpha_q y$
    $x_t \leftarrow x_{t+1}$
    $\boldsymbol{o}_t \leftarrow \boldsymbol{o}_{t+1}$
    $a_t \leftarrow a_{t+1}$
  end
end
$\pi(bo) = \text{argmax}_a Q(\boldsymbol{o}, a)$ // Extract greedy policy

Algorithme 15 : JSJ algorithm. Uses Monte Carlo estimates of the average reward value functions to perform policy improvements

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

problem since it searches for stochastic policies that we showed can be better than our sought depth-1 decision tree policy (cf. figure 3.2).

In addition to the traditional tabular RL algorithms above, we also apply asymmetric Q-learning, asymmetric Sarsa, and JSJ (algorithms 13, 14 and 15). We use at least 200 000 POIBMDP time steps per experiment. Each experiment, i.e an RL algorithm learning in a POIBMDP, is repeated 100 times.

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

Metrics : We will consider two metrics. First, the sub-optimality gap during training, w.r.t. the interpretable RL objective, between the learned partially observable policy and the optimal deterministic partially observable policy : $|\mathbb{E}\left[V^{\pi^\star}(\boldsymbol{s}_0, \boldsymbol{o}_0)|\boldsymbol{s}_0 \sim T_0\right] - \mathbb{E}[V^{\pi}(\boldsymbol{s}_0, \boldsymbol{o}_0)|\boldsymbol{s}_0 \sim T_0]|$. Because we know the whole POIBMDP model that we can represent exactly as tables, and because we know for each ζ the interpretable RL objective value of the optimal deterministic partially observable policy (cf. figure 3.2), we can report the *exact* sub-optimality gaps.

Second, we consider the distribution of the learned trees over the 100 training seeds. Indeed, since for every POIBMDP we run each algorithm 100 times, at the end of training we get 100 deterministic partially observable policies (we compute the greedy policy for stochastic policies returned by JSJ and Policy Gradient), from which we can extract the equivalent 100 decision tree policies using algorithm 10 and we can count which one are of e.g. depth-1. This helps understand which trees RL algorithms tend to learn.

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

In figure 3.3, we plot the sub-optimality gaps–averaged over 100 seeds–of learned policies during training. We do so for 200 different POIBMDPs where we change the reward for information gathering actions : we sample 200 ζ values uniformly in $[-1, 2]$. In figure 3.3, a different color represents a different POIBMDP.

Recall from figure 3.2 that for :

— $\zeta \in [-1, 0]$, the optimal deterministic partially observable policy is a depth-0 tree
— $\zeta \in]0, 1[$, the optimal deterministic partially observable is a depth-1 tree
— $\zeta \in [1, 2]$, the optimal deterministic partially observable is a "infinite" tree that contains infinite number of internal nodes.

We observe that, despite all sub-optimality gaps converging independently of the ζ values, not all algorithms in all POIBMDPs fully minimize the sub-optimality gap. In particular, all algorithms seem to consistently minimize the gap, i.e. learn the optimal policy or Q-function, only for $\zeta \in [1, 2]$ (all the yellow lines go to 0). However, we are interested in the range $\zeta \in]0, 1[$ where the optimal decision tree policy is non-trivial, i.e. not taking the same action forever. In that range, no baseline consistently minimizes the sub-optimality gap.

In figure 3.4, we plot the distributions of the final learned trees over the 100 random seeds in function of ζ from the above runs. For example, in figure 3.4, in the top left plot, when learning 100 times in a POIBMDP with $\zeta = 0.5$, Q-learning returned almost 100 times a depth-0 tree. Again, on none of those subplots do we see a high rate of learned depth-1 trees for $\zeta \in]0, 1[$. It is alerting that the most frequent learned trees are the depth-0 trees for $\zeta \in]0, 1[$ because such trees are way more sub-optimal w.r.t. the interpretable RL objective (cf. definition 10) than e.g. the depth-2 unbalanced trees (cf. figure 3.2). One interpretation of this phenomenon is that the learning in POIBMDPs is very difficult and so agents tend to converge to trivial policies, e.g., repeating the same base action.

However, on the positive side, we observe that asymmetric versions of Q-learning and Sarsa have found the optimal deterministic partially observable policy–the depth-1 decision tree–more frequently throughout the optimality range $]0, 1[$, than their symmetric counter-parts for $\zeta \in]0, 1[$. Next, we quantify how difficult it is to do RL to learn partially observable policies in POIBMDPs.

3.4.3 How difficult is it to learn in POIBMDPs ?

In this section we run the same (asymmetric) reinforcement learning algorithms to optimize either the RL objective (cf. definition 4) in MDPs (cf. definition 3) or IBMDPs (cf. definition 9), or the interpretable RL objective in POIBMDPs 10. This essentially results in three distinct problems :

1. Learning an optimal standard Markovian policy in an MDP, i.e. optimizing the RL objective in an MDP.
2. Learning an optimal standard Markovian policy in an IBMDP, i.e. optimizing the RL objective in an IBMDP.
3. Learning an optimal deterministic partially observable policy in a POIBMDP.

In order to see how difficult each of these three problems is, we can run a *great* number of experiments for each problem and compare solving rates. To make solving

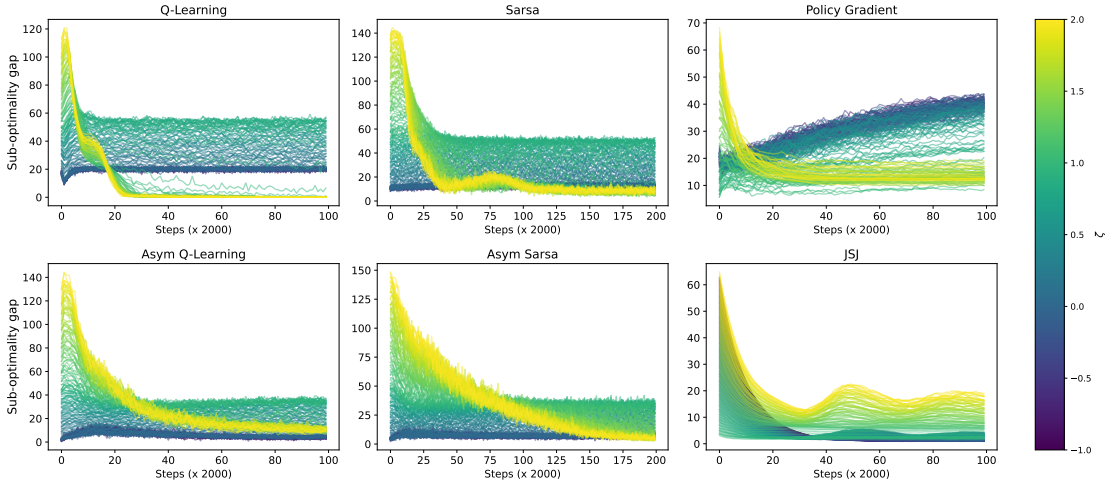


FIGURE 3.3 – (Asymmetric) reinforcement learning in POIBMDPs. In each subplot, each single line is colored by the value of ζ in the corresponding POIBMDP in which learning occurs. Each single learning curve represent the sub-optimality gap averaged over 100 seeds.

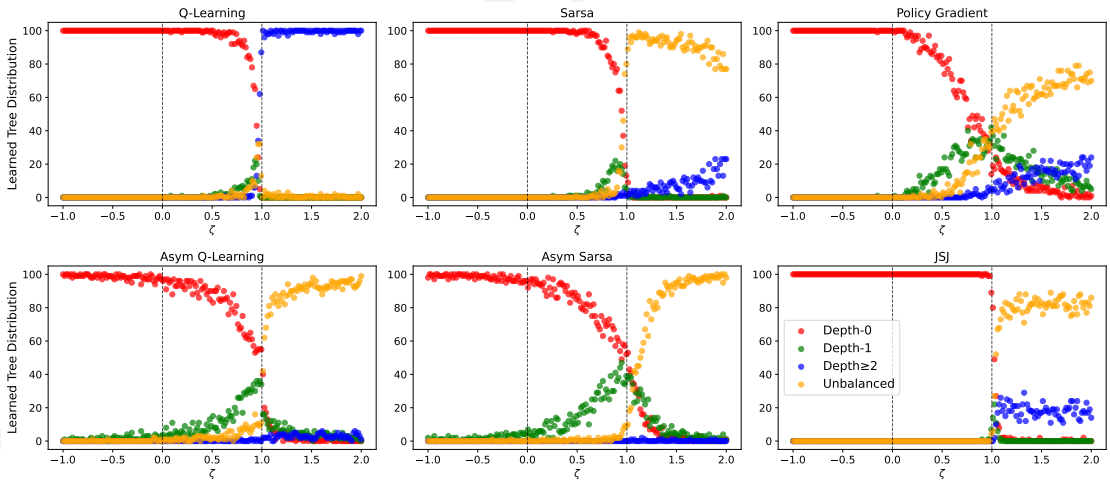


FIGURE 3.4 – Distributions of final tree policies learned across the 100 seeds. For each ζ value, there are four colored points. Each point represent the share of depth-0 trees (red), depth-1 trees (green), unbalanced depth-2 trees (orange) and depth-2 trees (blue).

Tableau 3.1 – Asymmetric sarsa hyperparameters (768 combinations each run 10 times)

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

rates comparable we consider a unique instance for each of those problems. Problem 1 is learning one of the optimal standard Markovian deterministic policy from figure 9 for the grid world from example 9 with $\gamma = 0.99$. Problem 2 is learning one of the optimal standard Markovian deterministic for the IBMDP from figure 1.1 with $\gamma = 0.99$ and $\zeta = 0.5$. This is similar to the previous chapter's experiments where we applied DQN or PPO to an IBMDP for CartPole without constraining the search to partially observable policies (see e.g. figure2.3.1). Problem 3 is what has been done in the previous section to learn deterministic partially observable policies where in addition of fixing $\gamma = 0.99$ we also fix $\zeta = 0.5$.

We use the six (asymmetric) RL algorithms from the previous section and try a wide set of hyperparameters and additional learning tricks (optimistic Q-function, eligibility traces, entropy regularization and ϵ-decay, all are described in [121]). We only provide the detailed hyperparameters for asymmetric Sarsa and an overall summary for all the algorithms in tables 3.1 and 3.2. The complete detailed lists of hyperparameters are given in the appendix A.3. Furthermore, the careful reader might notice that there is no point running asymmetric RL on MDPs or IBMDPs when the problem does not require partial observability. Hence, we only run asymmetric RL for POIBMDPs and otherwise run all other RL algorithms and all problems.

Each unique hyperparameters combination for a given algorithm on a given problem is run 10 times on 1 million learning steps. For example, for asymmetric Sarsa, we run a total of $10 \times 768 = 7680$ experiments for learning deterministic partially observable policies for a POIBMDP (cf. table 3.1). To get a success rate, we can simply divide the number of learned depth 1 tree by 7680 (recall that for $\gamma = 0.99$ and $\zeta = 0.5$, the optimal policy is a depth-1 tree (e.g. figure 3.1) as per figure 3.2).

The key observations from figure 3.5 is that reinforcement learning a deterministic partially observable policy in a POIBMDP, is way harder than learning a standard

Tableau 3.2 – Summary of RL baselines Hyperparameters

algorithm	Problem	Total Hyperparameter Combinations
Policy Gradient	PO/IB/MDP	420
JSJ	POIBMDP	15
Q-learning	PO/IB/MDP	192
Asym Q-learning	POIBMDP	768
Sarsa	PO/IB/MDP	192
Asym Sarsa	POIBMDP	768

Markovian policy. For example, Q-learning only finds the optimal solution to (cf. definition 10) in only 3% of the experiments while the same algorithms to optimize the standard RL objective (cf. definition 4) in an MDP or IBMDP found the optimal solutions 50% of the time. Even though asymmetry seems to increase performances ; learning a decision tree policy for a simple grid world directly with RL using the framework of POIBMDP originally developed in [124] seems way too difficult and costly as successes might require a million steps for such a seemingly simple problem. An other difficulty in practice that we did not cover here, is the choice of information gathering actions. For the grid world MDP, choosing good IGAs ($x \leq 1$ and $y \leq 1$) is simple but what about more complicated MDPs : how to instantiate the (PO)IBMDP action space such that internal nodes in resulting trees are useful for predictions ?

To go even further, on figure 3.6 we re-run experiments from figure 3.3 and figure 3.4 using the top performing hyperparameters for asymmetric Q-learning (given in appendix A.12). While those hyperparameters resulted in asymmetric Q-learning returning 10 of out 10 times an optimal depth 1 tree, the performances didn't transfer. On figure 3.6 despite higher success rates in the region $\zeta \in]0, 1[$ compared to figure 3.4.

3.5 Conclusion

In this chapter, we have shown that direct learning of decision tree policies for MDPs can be reduced to learning deterministic partially observable policies in POMDPs that we called POIBMDPs. By crafting a POIBMDP for which we know exactly the optimal deterministic partially observable policy w.r.t. the interpretable RL objective (cf. definition 10), we were able to benchmark the sub-optimality of solutions learned with (asymmetric) reinforcement learning.

Across our experiments, we found that no algorithm could consistently learn a depth-1 decision tree policy for a grid world MDP despite it being optimal both w.r.t.

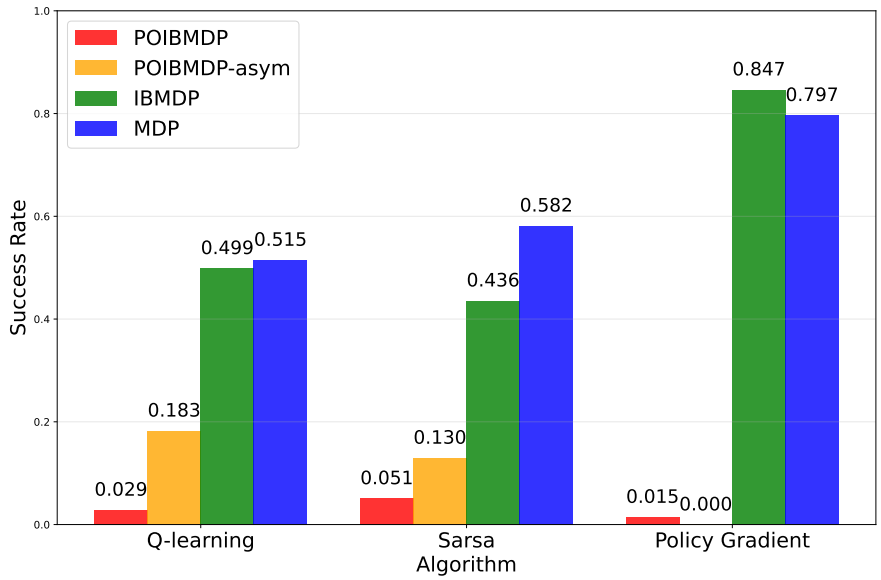
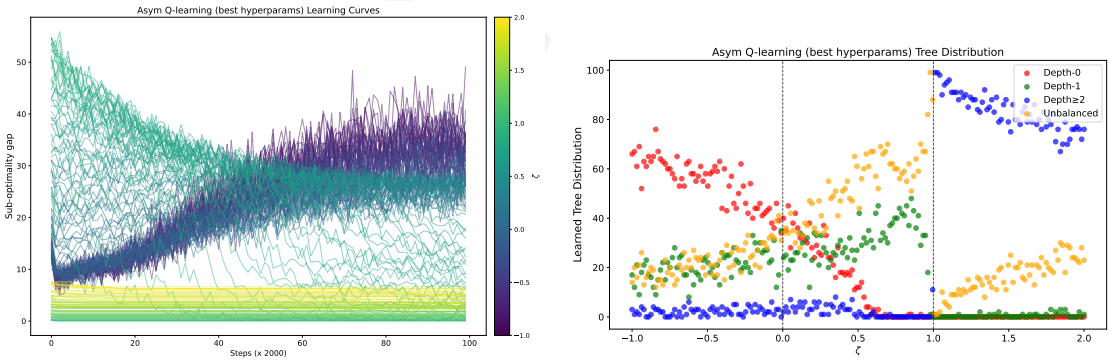


FIGURE 3.5 – Success rates of different (asymmetric) RL algorithms over thousands of runs when applied to learning deterministic partially observable policies in a POIBMDP or learning deterministic policies in associated MDP and IBMDP.



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

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

FIGURE 3.6 – Analysis of the top-performing asymmetric Q-learning instantiation. (left) Learning curves, and (right) tree distributions across different POIBMDP configurations.

the interpretable RL objective and standard RL objective (cf. definition 4). When compared to the results of VIPER from figure 12, direct RL is worse at retrieving decision tree policies with good interpretability-performance trade-offs. This echoes the results from the previous chapter in which we saw that direct deep RL performed worse than imitation learning to find decision tree policies for CartPole.

In the next chapter, we find that RL can find optimal deterministic partially observable policies for a special class of POIBMDPs that we believe makes for a convincing argument as to why direct learning of decision tree policies that optimize the RL objective (cf. definition 4) is so difficult.

Chapitre 4

Direct reinforcement learning of decision tree policies for classification tasks

In this section, we show that for a special class of POIBMDPs(cf. definition 12), reinforcement learning (cf. section 4) can learn optimal deterministic partially observable policies w.r.t to the interpretable RL objective (cf. definition 10), i.e. we can do direct decision tree policy learning for MDPs. This class of POIBMDPs are those for which base MDPs have uniform transitions, i.e. $T(\boldsymbol{s}, a, \boldsymbol{s}') = \frac{1}{|S|}$ (cf. definitions 3 and 9). The supervised learning objective (cf. definition 2) can be re-formulated in terms of the RL objective (cf. definition 4) and MDPs with such uniform transitions. Indeed a supervised learning task can be formulated as maximizing the RL objective in an MDP where, actions are class (or target) labels, states are training data, the reward at every step is 1 if the correct label was predicted and 0 otherwise, and the transitions are uniform : the next state is given by uniformly sampling a new training datum. This implies that learning deterministic partially observable policies in POIBMDPs where the base MDP encodes a supervised learning task is equivalent to doing decision tree induction to optimize the supervised learning objective. If RL does work for such fully observable POIBMDPs, this would mean that : 1) the difficulty of direct learning of decision tree policies for *any* MDP using POIBMDPs, exhibited in the previous chapters, is most likely due to the partial observability, and 2), it means that we can design new decision tree induction algorithms for the supervised learning objective by solving MDPs. Let us show that, POIBMDPs associated with MDPs encoding supervised learning tasks, are in

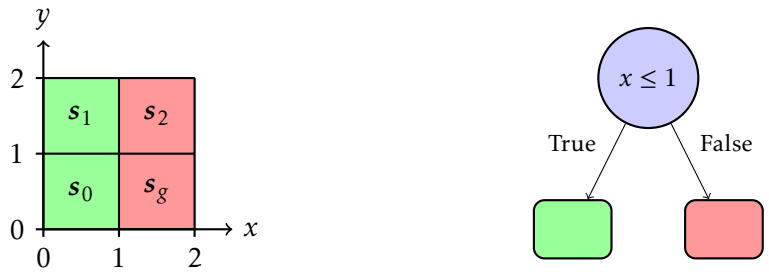


FIGURE 4.1 – Classification MDP optimal actions. In this classification MDP, there are four data to which to assign either a green or red label. On the right, there is the unique optimal depth-1 tree for this particular classification MDP. This depth-1 tree also maximizes the accuracy on the corresponding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

fact MDPs themselves. Let us define such supervised learning MDPs in the context of a classification task (this definition extends trivially to regression tasks).

Definition 14 (Classification Markov decision process)**.** *Given a set of N examples denoted* $\mathcal{E} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ *where each datum* $\boldsymbol{x}_i \in \mathcal{X}$ *is described by a set of p features* x_{ij} *with* $1 \leq j \leq p$*, and* $y_i \in \mathbb{Z}^m$ *is the label associated with* $\boldsymbol{x}_i$*, a classification Markov decision Process is an MDP* $\langle S, A, R, T, T_0 \rangle$ *3 where :*

- *the state space is* $S = \{\boldsymbol{x}_i\}_{i=1}^N$*, the set of training data features*
- *the action space is* $A = \mathbb{Z}^m$*, the set of unique labels*
- *the reward function is* $R : S \times A \to \{0, 1\}$ *with* $R(\boldsymbol{s} = \boldsymbol{x}_i, a) = 1_{\{a=y_i\}}$
- *the transition function is* $T : S \times A \to \Delta(S)$ *with* $T(\boldsymbol{s}, a, \boldsymbol{s}') = \frac{1}{N} \quad \forall \boldsymbol{s}, a, \boldsymbol{s}'$
- *the initial distribution is* $T_0(\boldsymbol{s}_0 = \boldsymbol{s}) = \frac{1}{N}$

One can be convinced that policies that maximize the RL objective (cf. definition 4) in classification MDPs are classifiers that maximize the prediction accuracy because $\sum_{i=1}^N 1_{\pi(\boldsymbol{x}_i)=y_i} = \sum_{i=1}^N R(\boldsymbol{x}_i, \pi(\boldsymbol{x}_i))$. We defer the formal proof in the next part of the manuscript in which we extensively study supervised learning problems.

In figure 4.1 we give an example of such classification MDP with 4 data in the training set and 2 classes :

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

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

Definition 15 (Classification POIBMDP)**.** *Given a classification MDP* $\langle \{\boldsymbol{x}_i\}_{i=1}^N, \mathbb{Z}^m, R, T, T_0 \rangle$ *(cf. definition 14), and an associated POIBMDP* $\langle S, O, A, A_{info}, R, \zeta, T_{info}, T, T_0 \rangle$ *(cf. definition 12), a classification POIBMDP is an MDP (cf. definition 3) :*

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

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

Those classification POIBMDPs are essentially MDPs with stochastic transitions. It means that deterministic partially observable policies (cf. definition 1) $O :\to A \cup A_{info}$ are in fact Markovian policy for those classification POIBMDPs. More importantly, it means that, for a given γ and ζ, if we were to know the whole POIBMDP model, we could use planning, e.g. value iteration (cf. definition 2), to compute *optimal* decision tree policies. Similarly, standard RL algorithms like Q-learning (cf. definition 4) should work as well as for any MDP to learn optimal decision tree policies.

This is exactly what we check next. We use the same direct approach to learn decision tree policies as in previous chapters, except that now the base MDP is a classification task and not a sequential decision making task.

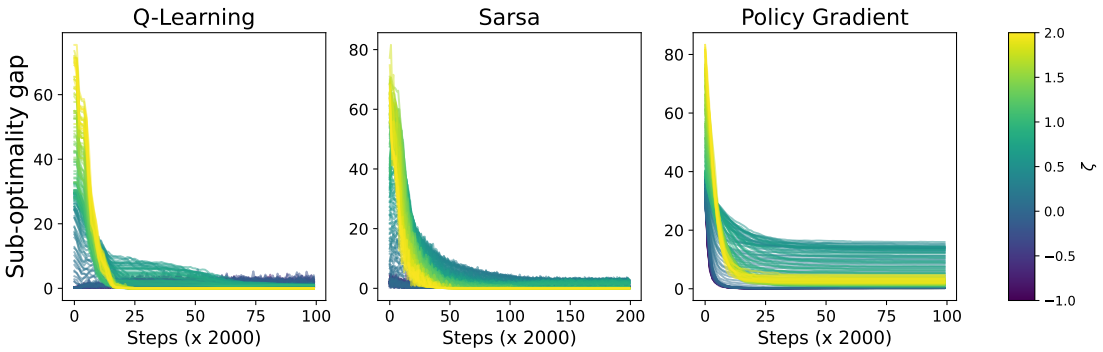


FIGURE 4.2 – We reproduce the same plot as in figure 3.3 for classification POIBMDPs. Each individual curve is the sub-optimality gap of the learned policy during training averaged over 100 runs for a single ζ value.

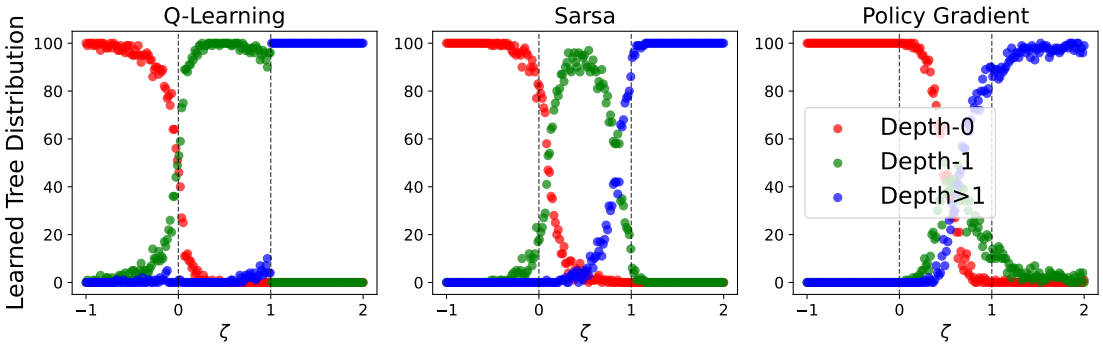


FIGURE 4.3 – We reproduce the same plot as in figure 3.4 for classification POIBMDPs. Each colored dot is the number of final learned trees with a specific structure for a given ζ.

4.1 How well do RL agents learn in classification POIBMDPs?

Similarly to the previous chapter, we are interested in a very simple classification POIBMDP. We study classification POIBMDPs associated with the example classification MDP from figure 4.1.

We construct classification POIBMDPs with $\gamma = 0.99$, 200 values of $\zeta \in [0,1]$ and IGAs $x \leq 1$ and $y \leq 1$. Since classification POIBMDPs are MDPs, we do not need to analyze asymmetric RL and JSJ baselines like in the previous chapter (cf. algorithms 13, 14, and 15).

Fortunately this time, compared to general POIBMDPs, RL can be used to learn optimal deterministic partially observable policies $O :\rightarrow A \cup A_{info}$ w.r.t. the interpretable

RL objective (cf. definition 10) in classification POIBMDPs. Such policies are equivalent to decision tree classifiers. We observe on figure 4.2 that both Q-learning and Sarsa consistently minimize the sub-optimality gap independently of the interpretability-performance trade-off ζ. Hence they are able to learn the optimal depth-1 decision tree classifier (figure 4.1) most of the time in the optimality range $\zeta \in]0, 1[$ (cf. figure 4.3).

4.2 Conclusion

In this part of the manuscript we were interested in algorithms that can learn decision tree policies that optimize some trade-off of interpretability and performance w.r.t. the RL objective 4 in MDPs. In particular, using the framework of Topin et. al. [124], we were able to explicitly write an interpretable RL objective function (cf. definition 10).

In chapter 2, we compared the algorithms proposed in [124] that directly optimize this objective, to imitation learning algorithms that only solve a proxy problem. While those direct RL algorithms are able to *learn*, i.e. find better and better solutions with time (cf. figures 2.3.1 and 2.3.1), the decision tree policies returned perform worse in average than imitated decision trees w.r.t. the RL objective of interest (cf. figure 2.3) for similar number of nodes and depth.

We further analyzed the failure mode of direct learning of decision tree policies by making connections with POMDPs [119, 39]. In chapter 3, we showed that learning decision tree policies for MDPs could be explicitly formulated as learning a deterministic partially observable (also known as memoryless or reactive) policy in a specific POMDP that we called POIBMDP 12. We showed that both RL and asymmetric RL, a class of algorithms specifically designed for POMDPs [7, 8], were unable to consistently learn an optimal depth-1 decision tree policies for a very small grid world MDP when using the POIBMDP framework. In particular, we compared, in a very controlled experiment, the success rates of the same learning algorithms when seeking standard Markovian policies versus partially observable policies in decision processes that shared the same transitions and rewards (cf. section 3.4.3). We demonstrated on figure 3.5 that introducing partial observability greatly reduced the success rates (we also observed this implicitly on figures 2.3.1 and 2.3.1).

Finally, in this chapter we showed that using RL to optimize the interpretable RL objective in fully observable POIBMDPs, i.e. POIBMDPs that are just MDPs, could learn optimal decision tree policies (cf. figures 4.2 and 4.3) adding new evidence that direct interpretable RL is difficult because it involves POMDPs.

This class of fully observable POIBMDPs (cf. definition 15) contains the decision

tree induction problem for supervised learning tasks (cf. definition 2). This sparks the question : what kind of decision tree induction algorithm can we get using the MDP formalism ? This is exactly what we study in the next part of this manuscript.

Those few chapters raise other interesting questions. We focused on non-parametric tree learning because RL algorithms can learn decision tree policies with potentially optimal interpretability-performance trade-offs through the reward of information gathering actions in (PO)IBMDPs (cf. definitions 9 and 12). However this comes at a cost of partial observability which makes learning difficult. Parametric tree policies on the other hand, can be computed with reinforcement learning directly in the base MDP. However existing RL algorithms for parametric decision tree policies [115, 135, 84] require to re-train a policy entirely for each desired level of interpretability, i.e. each unique tree structure, future research in this direction should focus on algorithms for parametric tree policies that can re-use samples from one tree learning to train a different tree structure more efficiently. This would reduce the required quantity of a priori knowledge on the decision tree policy structure mentioned in section 1.1.

Attempting to overcome the partial observability challenges highlighted so far seems like a bad research direction. Indeed, while algorithms tailored specifically for the problem of learning deterministic partially observable policies for POIBMDPs might exist, we clearly saw that imitation learning was in practice a good alternative to direct interpretable reinforcement learning. Some limitations that we did not cover still exist such as how to choose good candidates information gathering actions or simply how to choose ζ for a target interpretability-performance trade-off. In the next part of the manuscript, we design a novel decision tree induction algorithm for the supervised learning objective using heuristics to generate information gathering actions in MDPs similar to classification POIBMDPs.

Deuxième partie

An Easier Problem : Decision Tree Induction as Solving MDPs

Chapitre 5

Introduction

[1] In this part of the manuscript we design a novel decision tree induction algorithm for supervised learning (cf. definition 2) using the MDP formalism. There already exist works formulating the decision tree induction problem as solving a Markov decision process [38, 47, 124, 26]. In particular, we showed in the previous part how Topin et al. [124] could be used to do that using classification POIBMDPs (cf. section 4). We move away from classification POIBMDPs and use a novel MDP formulation that we detail later. The key differences compared to, e.g. [124] is that our MDP states are sets of training data rather than feature bounds (cf. definition 15) and that we dynamically define the set of information gathering actions (cf. definition 9). An other novelty, is that unlike existing works that use reinforcement learning (cf. section 4), we use dynamic programming [14] to solve exactly the decision tree induction problem formulated as an MDP.

This part of the manuscript is organized as follows. In this chapter, we motivate the need for new decision tree induction algorithms and present the related works. In chapter 6, we introduce a novel MDP formulation of decision tree induction. Finally, in chapter 7, we show that our new decision tree induction framework performs very well both in terms of training loss and generalization capabilities.

5.1 Why do we want new decision tree induction algorithms ?

In supervised learning (cf. definition 2), decision trees (cf. figure 7) are valued for their interpretability and performance. While greedy decision tree algorithms like CART

1. This work was published at the 31st ACM SIGKDD conference in 2025 :https://dl.acm.org/doi/10.1145/3711896.3736868

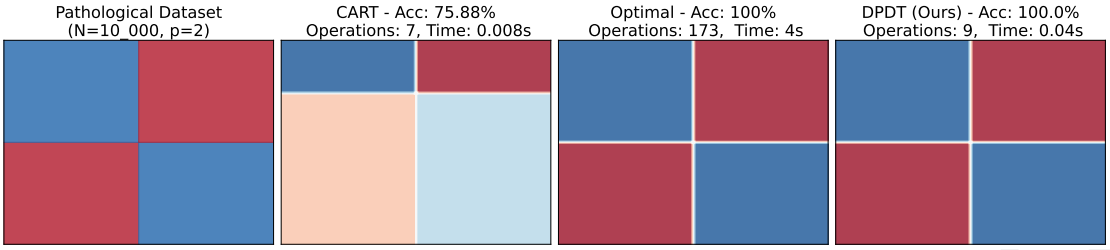


FIGURE 5.1 – Pathological dataset and learned depth-2 trees with their scores, complexities, runtimes, and decision boundaries.

(cf. algorithm 1) remain widely used due to their computational efficiency, they often produce sub-optimal solutions with respect to a regularized training loss (2). 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 (3) 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, extensive 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.

We already saw in the introduction of the manuscript that decision trees are well studied in supervised learning (cf. section 4). Decision tree inductions algorithms [105, 104, 20] are at the core of various machine learning applications. Ensembles of decision trees such as tree boosting [46, 45, 27, 100] are the state-of-the-art for supervised learning on tabular data [51].

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, 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 [85] iterates over all possible splits, that is, $N \times p = 20,000$ operations to find one node of the solution tree.

In this chapter, 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 [51] methodology to benchmark DPDT against both CART and optimal trees on hard datasets. Following the same methodology, we compare boosted DPDT [44] 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 (algorithm 1 [20]) 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 (2) [92]. But CART remains the default decision tree algorithm in many machine learning libraries [97, 27, 60, 140] 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* [20, chapter 3]. In recent years, more complex optimal decision tree induction algorithms have shown consistent gains over CART in terms of generalization capabilities [16, 132, 35].

Optimal decision tree approaches [16, 2, 133, 85, 35, 36, 71, 26] optimize a regularized training loss while using a minimal number of splits. However, direct optimization is not a convenient approach, as finding the optimal tree is known to be NP-Hard [57]. Despite the large number of algorithmic tricks to make optimal decision tree solvers efficient [35, 85], 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 [85] can only compute optimal trees up to depth 3. `PySTreeD`, the latest optimal decision tree library [71], 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 [91, 70, 29, 72]. 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. For the reader curious about optimal decision tree algorithms, we recommend reading Ayman Chaouki's thesis on the topic [25, chapter 2].

In this part, we only consider the induction of non-parametric (cf. section 1.1) binary depth-constrained axis-aligned trees. We write this model class $\mathcal{T}_D$ where D is the maximum tree depth. Axis-aligned tree nodes only test one data feature only against one thresholds only as opposed to e.g. oblique trees [94]. In our definition (cf. definition 4), this means that we only consider Boolean functions $1_{\{x_{\_,j} \leq v\}}$ (the careful reader might notice the clear similarity with information gathering actions in IBMDPs (definition 9, [124])). Like for the reinforcement learning setting, there exists a line of work on parametric trees : Tree Alternating Optimization (TAO) algorithm [23, 139, 24] that only optimizes tree nodes threshold values for fixed nodes features similarly to optimizing neural network weights with gradient-based methods.

Now we write explicitly the supervised learning objective for decision trees :

Definition 16 (Supervised learning of decision trees (decision tree induction))**.** *Assume that we have access to a set of N examples denoted $\mathcal{E} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$. Each datum $\boldsymbol{x}_i$ is described by a set of p features x_{ij} with $1 \leq j \leq p$. $y_i \in \mathcal{Y}$ is the label associated with $\boldsymbol{x}_i$. The objective of decision tree induction is to find a tree with maximum depth D, $T \in \mathcal{T}_D$ that minimizes :*

$$\mathcal{L}_\alpha(T) = \frac{1}{N} \sum_{i=1}^{N} l(y_i, T(\boldsymbol{x}_i)) + \alpha C(T)$$

where $C : \mathcal{T}_D \to \mathbb{R}$ is a complexity penalty that helps prevent or reduce overfitting such as the number of nodes [20, 85], or the expected number of splits to label a data[91]. The complexity penalty is weighted by $\alpha \in [0, 1]$.

In the rest of this part we focus on classification tasks : we use the 0–1 loss $l(y_i, T(\boldsymbol{x}_i)) = 1_{\{y_i \neq T(\boldsymbol{x}_i)\}}$. Please note while we focus on classification tasks, our framework extends naturally to regression tasks.

In the supervised learning setting, there exists many other areas of decision tree research [76] such as inducing non-axis parallel decision trees [94, 59], splitting criteria of greedy trees [72], different optimization of parametric trees [96, 100], or pruning methods [41, 90]. In the next chapter, we give a novel formulation of decision tree induction tasks as MDPs.

Chapitre 6

Decision tree induction as solving an MDP

6.1 The Markov decision process

We now formulate the decision tree induction problem (cf. definition 16) as finding the optimal policy in an MDP (cf. definition 4). 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 an MDP $\mathcal{M}\langle S, A, R_\alpha, T, T_0\rangle$ (cf. definition 3) that we present next. A state is a pair made of a subset of examples $\mathcal{X} \subseteq \mathcal{E}$ and a depth d. Like for classification POIBMDPs (cf. definition 9 and algorithm 10), the depth d is actually not necessary to extract a tree from a policy. However, it is necessary in order to solve the MDP. Indeed, because we constrain the model class to limited-depth trees, the horizon of our MDP is finite and hence as per [102], the optimal policy is non-Markovian and depends on the time.

The set of states is $S = \{(\mathcal{X}, d) \in P(\mathcal{E}) \times \{0, \dots, D\}\}$ 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 $A = \mathcal{F} \cup \{1, \dots, K\}$ (these are essentially information gathering actions). From state $s = (X, d)$ and a splitting action $a \in \mathcal{F}$, the transition function P moves to the next state $s_l = (\mathcal{X}_l, d + 1)$ with probability $p_l = \frac{|\mathcal{X}_l|}{|\mathcal{X}|}$ where $\mathcal{X}_l = \{(\boldsymbol{x}_i, y_i) \in \mathcal{X} : \boldsymbol{x}_i \leq x_{ij}\}$, or to state $s_r = (\mathcal{X} \setminus \mathcal{X}_l, d + 1)$ 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 (this is ζ in a classification POIBMDP (cf. definition 15)) and the proportion of misclassified examples of $\mathcal{X}$ $-\frac{1}{|\mathcal{X}|}\sum_{(\boldsymbol{x}_i,y_i)\in\mathcal{X}} l(y_i,a)$ for class assignment actions. $\alpha \in [0,1]$ controls the accuracy-complexity trade-off defined in the regularized training objective 16.

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 $s_0 = (\mathcal{E}, 0)$. This objective is a re-writing of the RL objective (cf. definition 4) where the trajectory have finite lengths D rather than being infinite. Any such policy can be converted into a binary decision tree through a recursive extraction function $E(\pi, s)$ (equivalent to algorithm 10) that returns, either a leaf node with class $\pi(s)$ if $\pi(s)$ is a class assignment, or a tree with root node containing split $\pi(s)$ and left/right sub-trees $E(\pi, s_l)/E(\pi, s_r)$ if $\pi(s)$ is a split. The final decision tree T is obtained by calling $E(\pi, s_0)$ on the initial state s_0.

Proposition 3 (Objective Equivalence)**.** *Let π be a deterministic policy of the MDP (cf. section 6.1) 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 the decision tree induction objective (cf. definition 16).*

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 w.r.t the decision tree induction objective.

Démonstration. 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 $J_\alpha(\pi, \mathcal{X}_l, D) = \mathcal{L}_\alpha(T_l, \mathcal{X}_l)$ and $J_\alpha(\pi, \mathcal{X}_r, D) = \mathcal{L}_\alpha(T_r, \mathcal{X}_r)$, where $\mathcal{X}_l$ and $\mathcal{X}_r$ 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

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

□

6.2 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 6.1. The second step solves it to obtain a policy that maximizes the objective from definition 6.1 and that is equivalent to a decision tree. Both steps are now detailed.

6.2.1 Constructing the MDP

An algorithm constructing the MDP of section 6.1 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.

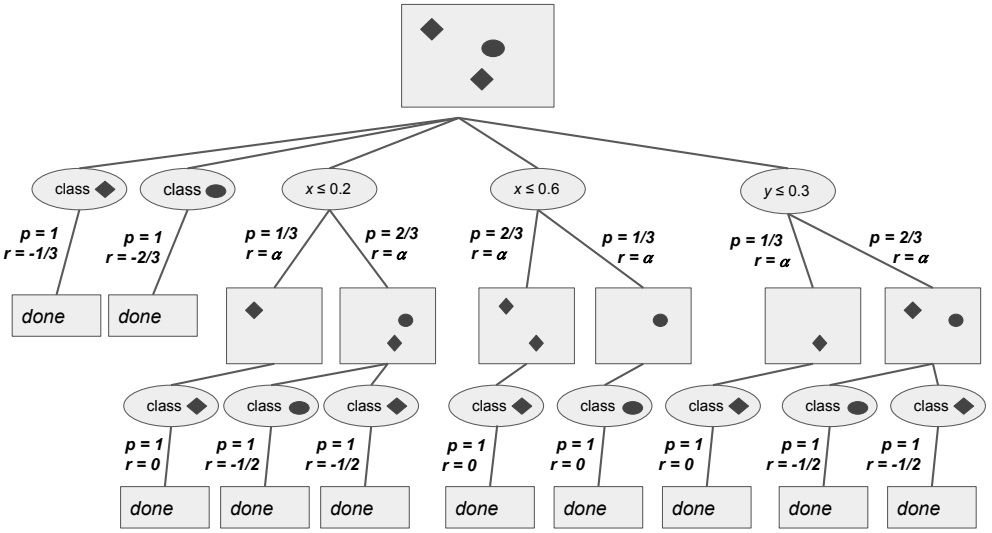


FIGURE 6.1 – Schematics of the MDP from section 6.1 to learn a decision tree of depth 2 to classify a toy dataset with three samples, two features (x,y), and two classes (oval, diamond) and using an arbitrary splits generating function.

6.2.2 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 for the decision tree induction problem (cf. definition 16), 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 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 [132, 16, 71, 85, 133, 35, 2, 36, 70, 26].

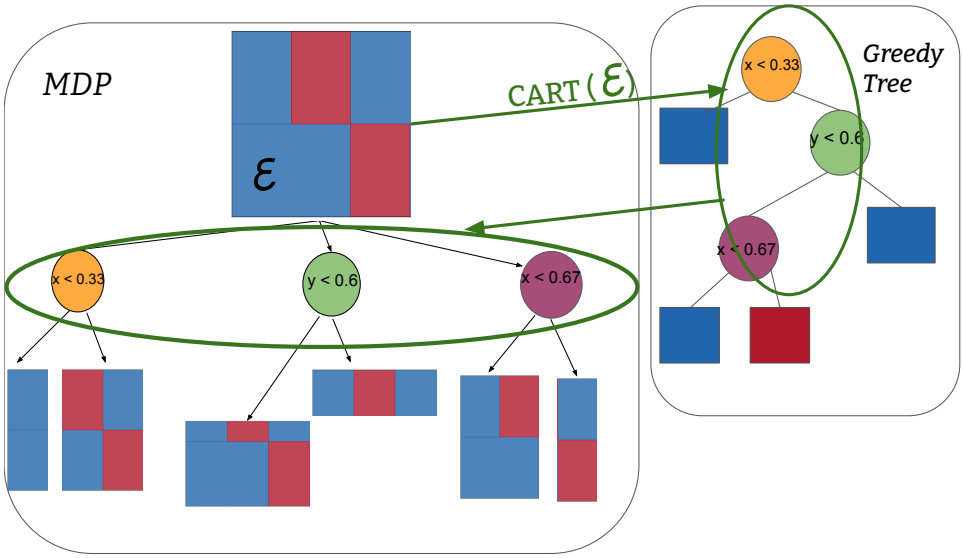


FIGURE 6.2 – How CART is used in DPDT to generate candidate splits given the example data in the current state.

Top B most informative splits [18] proposed to generate splits with a function that returns, for any state $s = (\mathcal{X}, d)$, the B most informative splits over $\mathcal{X}$ 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 $s = (\mathcal{X}, d)$, we propose to find the most discriminative split, i.e. the feature splits that best predicts the class of data in $\mathcal{X}$. 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.

6.2.3 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

Algorithme 16 : 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
$\pi^*(s) = \text{argmax}_{a \in A_s^{\mathcal{M}}} Q^*(s,a) \forall s \in \mathcal{M}$
// Extracting tree from policy
$T \leftarrow E(\pi^*, s_0^{\mathcal{M}})$

backward to the initial state, we compute the optimal state-action values using Bellman's optimality equation [14], and then deducing the optimal policy.

From now on, we write DPDT to denote algorithm 16 when the split function is a call to CART. We discuss key bottlenecks when implementing DPDT in subsequent sections. We now state theoretical results when using DPDT with the CART heuristic.

6.2.4 Performance guarantees for DPDT

We now show that : 1) DPDT minimizes the loss from decision tree induction objective (cf. definition 16) 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. 16. However, we are still guaranteed to find trees that are better or equivalent to those induced by CART :

Theorem 1 (MDP solutions are not worse than the greedy tree)**.** *Let π^* be an optimal deterministic policy of the MDP, where the action space at every state is restricted to the top B most informative or discriminative splits. Let T_0 be the tree induced by CART and $\{T_1, \ldots, T_M\}$ 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

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 [41] for a review of pruning methods for decision trees.

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 [92, 91]. We aim to show that DPDT is a cheap way to alleviate the weaknesses of greedy trees in this type of problems. The following theorem states that there exist classification problems such that DPDT optimizes the regularized training loss strictly better than greedy algorithms such as CART, ID3 or C4.5.

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

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

6.2.5 Proof of improvement over CART

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

Definition 17 (XOR dataset)**.** *Let us defined the XOR dataset as $\mathcal{E}_{XOR} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$. $\boldsymbol{x}_i = (\boldsymbol{x}_i, y_i) \sim \mathcal{U}([0,1]^2)$ are i.i.d 2-features samples. $y_i = f(\boldsymbol{x}_i)$ are alternating classes with $f(x, y) = (\lfloor 2x \rfloor + \lfloor 2y \rfloor) \bmod 2$.*

Lemma 1. *The first greedy split is chosen at random on the XOR dataset from definition 17.*

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 17. So in both R_{left} and R_{right} the probabilities of observing class 0 or class 1 at random are $\frac{1}{2}$. Since the class distributions in left and right regions are independent of the split location, all splits have the same objective value when the objective is a measure of information gain like the entropy or the Gini impurity. Hence, the first split in a greedily induced tree is chosen at random. □

Lemma 2. *When the first split is greedy on the XOR dataset from definition 17, 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 18 (Forced- Tree)**.** *Let us define the forced-tree as a greedy tree that is forced to make its first split at $y = \frac{1}{2}$.*

Lemma 3. *The forced-tree of depth 2 has a 0 loss on the XOR dataset from definition 17 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 6.2.5, there are only two splits ($x = \frac{1}{2}$ and $y = \frac{1}{2}$) out of $2|\mathcal{E}_{XOR}|$ that do not lead to sub-optimality. □

We can now formally prove theorem 2.

Démonstration. By definition of DPDT, all instances of DPDT with the CART nodes parameter $B \geq 2$ include the forced-tree from definition 18 in their solution set when applied to the XOR dataset (definition 17). We know from lemma 3 that with high probability, the forced-tree of depth 2 is strictly more accurate than the greedy tree of depth 2 on the XOR dataset. Because we know by proposition 3 that DPDT returns the tree with maximal accuracy from its solution set, we can say that DPDT depth-2

trees are strictly better than depth-2 greedy trees returned by e.g. CART on the XOR dataset. □

6.2.6 Practical implementation

The key bottlenecks lie in the MDP construction step of DPDT (section 6.1). 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 complexity. 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 $s = (\mathcal{X}, d)$ 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 $(B_1, B_2, \ldots, B_D)$, the splits generating function in algorithm 16 becomes $\phi(s_i) = \phi(\boldsymbol{x}_i, d = 1) = nodes(\text{CART}(s, max\_nodes = B_1))$ and $\phi(s_j) = \phi(\mathcal{X}_j, d = 2) = nodes(\text{CART}(s, max\_nodes = B_2))$. 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 16 are training samples $\mathcal{X} \subseteq \mathcal{E}$. 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 6.1 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 [22] with depth-first search and state-depth-dependent splits generating. In the next chapter, we evaluate DPDT using the above implementation.

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

Chapitre 7

Dynamic programming decision trees in practice

In this section, we empirically demonstrate advantageous properties of DPDT trees. The first part of our experiments focuses on the quality of solutions obtained by DPDT for the decision tree induction objective (cf. definition 16) compared to greedy and optimal trees. We know by theorems 1 and 2 that DPDT trees should find better solutions than greedy algorithms for certain problems ; but what about real problems ? After showing that DPDT can find optimal trees by considering much less solutions and thus performing orders of magnitude less operations, we will study the generalization capabilities of the latter : do DPDT trees label unseen data accurately ?

7.1 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 objective from definition 16.

7.1.1 Setup

Metrics : we are interested in the regularized training loss of algorithms optimizing the decision tree induction objective (cf. definition 16) 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

TABLEAU 7.1 – 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
Dataset	N	p	Quant-BnB	CART	light	full	light	full	Modified DQN
room	8103	16	**0.992**	0.968	0.991	**0.992**	0.990	**0.992**	0.715
bean	10888	16	**0.871**	0.777	0.812	0.853	0.804	0.841	0.182
eeg	11984	14	**0.708**	0.666	0.689	0.706	0.684	0.699	0.549
avila	10430	10	**0.585**	0.532	0.574	**0.585**	0.563	0.572	0.409
magic	15216	10	**0.831**	0.801	0.822	0.828	0.807	0.816	0.581
htru	14318	8	**0.981**	0.979	0.979	0.980	0.979	0.980	0.860
occup.	8143	5	**0.994**	0.989	0.991	**0.994**	0.990	0.992	0.647
skin	196045	3	**0.969**	0.966	0.966	0.966	0.966	0.966	0.612
fault	1552	27	**0.682**	0.553	0.672	0.674	0.672	0.673	0.303
segment	1848	18	**0.887**	0.574	0.812	0.879	0.786	0.825	0.137
page	4378	10	**0.971**	0.964	0.970	0.970	0.964	0.965	0.902
bidding	5056	9	**0.993**	0.981	0.985	**0.993**	0.985	**0.993**	0.810
raisin	720	7	**0.894**	0.869	0.879	0.886	0.875	0.883	0.509
rice	3048	7	**0.938**	0.933	0.934	0.937	0.933	0.936	0.519
wilt	4339	5	**0.996**	0.993	0.994	0.995	0.994	0.994	0.984
bank	1097	4	**0.983**	0.933	0.971	0.980	0.951	0.974	0.496
			Operations						
			Opt	Greedy	DPDT		Top-B		
Dataset	N	p	Quant-BnB	CART	light	full	light	full	
room	8103	16	10^6	15	286	16100	111	16100	
bean	10888	16	$5{\cdot}10^6$	15	295	25900	112	16800	
eeg	11984	14	$2{\cdot}10^6$	13	289	26000	95	11000	
avila	10430	10	$3{\cdot}10^7$	9	268	24700	60	38900	
magic	15216	10	$6{\cdot}10^6$	15	298	28000	70	4190	
htru	14318	8	$6{\cdot}10^7$	15	295	25300	55	2180	
occup.	8143	5	$7{\cdot}10^5$	13	280	16300	33	510	
skin	196045	3	$7{\cdot}10^4$	15	301	23300	20	126	
fault	1552	27	$9{\cdot}10^8$	13	295	24200	111	16800	
segment	1848	18	$2{\cdot}10^6$	7	220	16300	68	11400	
page	4378	10	10^7	15	298	22400	701	4050	
bidding	5056	9	$3{\cdot}10^5$	13	256	9360	58	2700	
raisin	720	7	$4{\cdot}10^6$	15	295	20900	48	1440	
rice	3048	7	$2{\cdot}10^7$	15	298	25500	49	1470	
wilt	4339	5	$3{\cdot}10^5$	13	274	11300	33	465	
bank	1097	4	$6{\cdot}10^4$	13	271	7990	26	256	

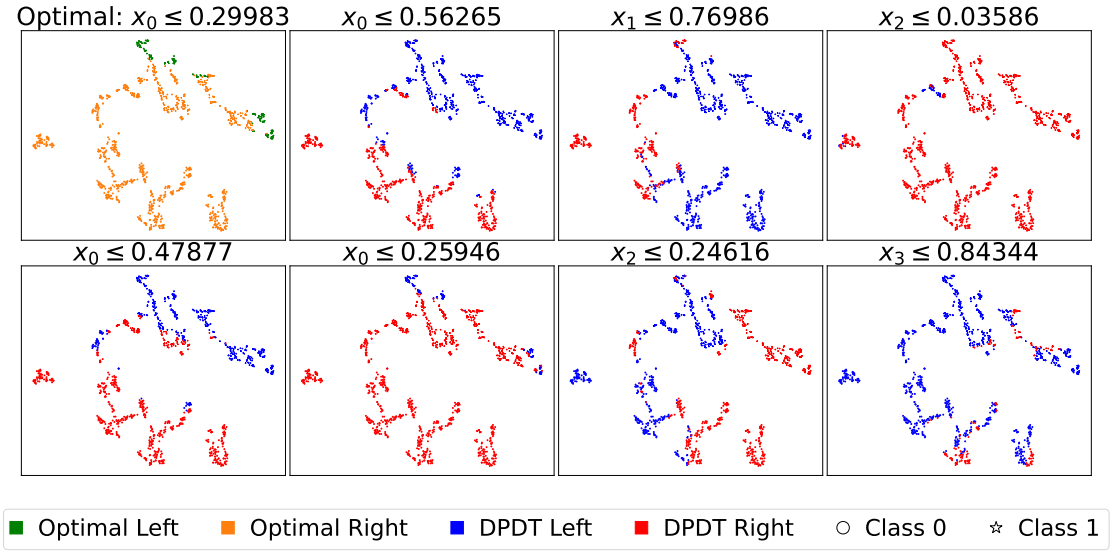


FIGURE 7.1 – Root splits candidate obtained with DPDT compared to the optimal root split on the Bank dataset. Each split creates a partition of p-dimensional data that we projected in the 2-dimensional space using t-SNE.

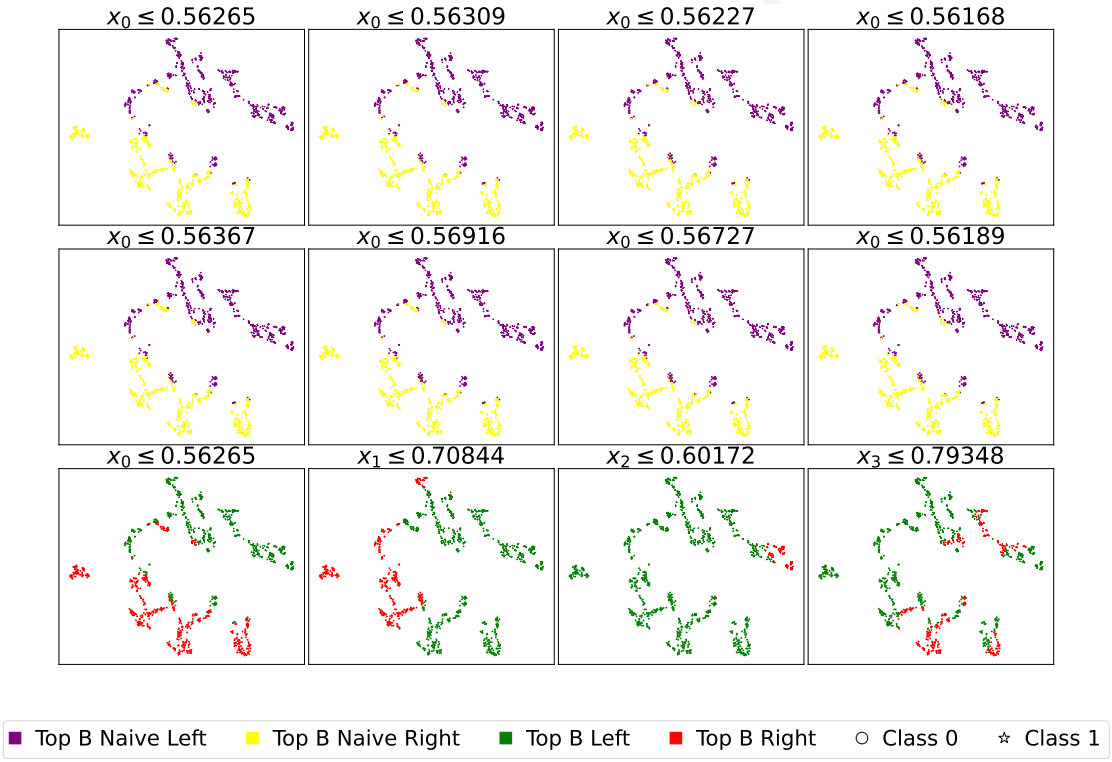


FIGURE 7.2 – Root splits candidate obtained with Top-B[18] on the Bank dataset. Each split creates a partition of p-dimensional data that we projected using t-SNE.

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 [20]. For optimal trees we compare DPDT to Quant-BnB [85] which is the only solver specialized for depth 3 trees and continuous features. We also consider the non-greedy baseline Top-B [18]. 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` [97] in `CPython` with a maximum depth of 3. Optimal trees are obtained by running the `Julia` implementation of the Quant-BnB solver from [85] 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 16 implemented in pure `Python` and the calls to CART and Top-B most informative splits generating functions from section 6.1 respectively. We use the modified DQN from section 2 on classification POIBMDPs with $\zeta = 1$ (we want to learn the best tree possible disregarding interpretability) and maximum depth control using rewards or termination signals.

Datasets : we us the same datasets as the Quant-BnB paper [85].

7.1.2 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 7.1, we observe several key patterns : first, light DPDT with 16 candidate root splits consistently outperforms the greedy baseline in all datasets. This shows that in practice DPDT can be strictly netter than CART outside of theorem 2 assumptions. Second, when comparing DPDT to Top-B, we see that DPDT generally achieves better accuracy for the same configuration. For example, on the bean dataset, full DPDT reaches 85.3% accuracy while full Top-B achieves 84.1%. This pattern holds on most datasets, suggesting that DPDT is more effective than selecting splits based purely on information gain.

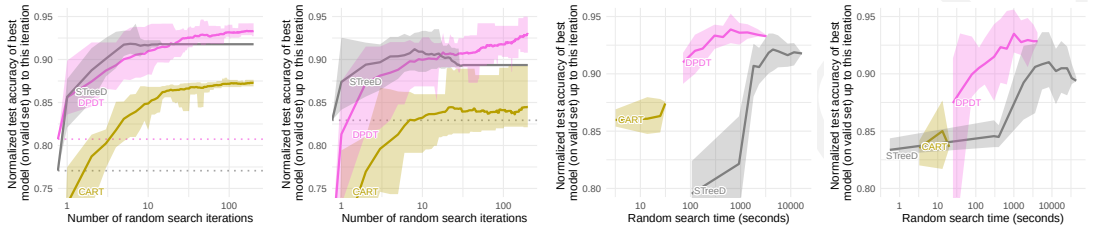
Third, both approaches achieve impressive computational efficiency compared to exact solvers. While optimal solutions require between 10^4 to 10^8 operations, DPDT and

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

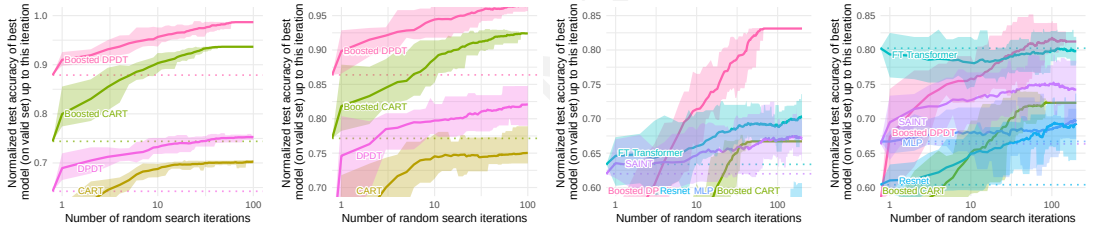
DPDT splits To understand why the CART-based split generation yields more accurate DPDT trees than the Top-B heuristic, we visualize how splits partition the feature space (figures 7.1, 7.2). We run both DPDT with splits from CART and DPDT with the Top-B most informative splits on the bank dataset. We 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 7.1 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 7.2 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.

7.2 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 [51]. We choose this benchmark because it was used to establish XGBoost [27] as the SOTA tabular learning model.



(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 7.3 – 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.

7.2.1 Setup

Metrics : We re-use the code from [51][1]. It relies on random searches for hyper-parameter tuning [15]. 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 [51], 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 algorithm. We use the test set accuracy (classification) to measure model performance. The aggregation metric is discussed in details in [51, section 3].

Datasets : we use the datasets curated by [51]. They are available on `OpenML` [128] and described in details in [51, 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 [51, section 3] to focus on core learning challenges. Some datasets are very large (millions of samples) like Higgs or Covertype [136, 17]. To ensure non-trivial learning tasks, datasets where simple models (e.g. logistic regression) performed within 5% of complex models (e.g. ResNet [50], HistGradientBoosting [97]) are removed. We use the same data partitioning strategy as [51] : 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 B.2 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 [62] for CART and DPDT. For DPDT we additionally consider eight different splits functions parameters configurations for the maximum nodes in the calls to CART. Surprisingly, after computing the importance of each hyperparameter of DPDT, we found that the maximum node numbers in the calls to CART are only the third most important hyperparameter behind

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

Tableau 7.2 – 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

classical ones like the minimum size of leaf nodes or the minimum impurity decrease (table 7.2). We use the `CPython` implementation of STreeD [2]. All hyperparameter grids are given in table B.4 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.

7.2.2 Observations

Generalization In figure 7.3, 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 7.3a and 7.3b, 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 7.3c and 7.3d, 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 7.3 are appealing for machine learning practitioners and data scientists that have to do hyperparameters search to find good models for their data while having computation constrains.

2. PySTreeD : `https://github.com/AlgTUDelft/pystreed`

Tableau 7.3 – 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

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 [3]. 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 [4] [86] 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 7.3 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 7.3 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 [86, section 5.3]). In table B.3 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.

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

7.3 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 [44, 46, 45] 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 [46, 45], or on efficient implementation of those algorithms [27, 100]. We show next that Boosted-DPDT (boosting DPDT trees with AdaBoost [44]) improves over recent deep learning algorithms.

7.3.1 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 [118] and other deep learning architectures from [50]).

On figures 7.3e and 7.3f 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 7.3g and 7.3h show that boosting DPDT trees using the default AdaBoost procedure [44] 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.

7.3.2 (X)GB-DPDT

We also boost DPDT trees with Gradient Boosting and eXtreme Gradient Boosting [45, 46, 27](X(GB)-DPDT). For each dataset from [51], 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

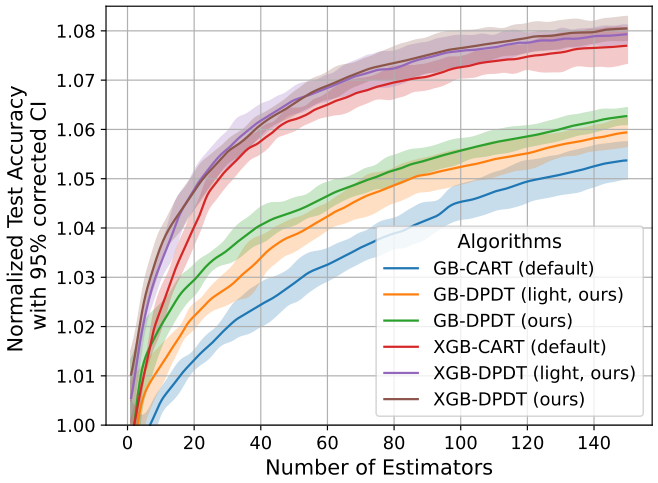


FIGURE 7.4 – Aggregated mean test accuracies of Gradient Boosting models as a function of the number of single trees.

computed using 5 different random seeds.

figure 7.4 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 [51] and potentially claim the state-of-the-art.

7.4 Conclusion

In this part of the manuscript, 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. While DPDT is designed for supervised learning (4), an other interesting future work direction could be to use DPDT in the tree induction routine of imitation learning algorithms like VIPER (algorithm 9, [12]). This could lean to better decision tree policy learning for the RL objective (4).

In the next and final part of this manuscript, we circle back to our original sequential decision making problems. We will move away from *learning* and focus on evaluating the interpretability-performance trade-offs of different policy classes from figure 4.

Troisième partie

Beyond Decision Trees : Evaluation of Interpretable Policies

Chapitre 8

Introduction

[1] While in parts I and II we quantified the performances of a given interpretable model class, e.g. decision trees of some maximum depth, this time we focus on quantifying the interpretability-performance trade-offs of different model classes. Most research in interpretable machine learning, including [124], relies on beliefs such as "decision trees are more interpretable than neural networks" illustrated for example in figure 4. In this part we propose a methodology to empirically compare the interpretability-performance trade-offs of multiple models from potentially different model classes.

In this chapter, we present results from existing empirical evaluations of model interpretability and introduce our methodology. In chapter 9, we validate our methodology against results obtained with user studies. In chapter 10, using our methodology, we present a large scale study of interpretability-performance trade-offs of different model classes when optimizing the RL objective (cf. definition 4).

8.1 Evaluating interpretability with humans

User studies typically evaluate interpretability through tasks such as simulation (whether a person can predict the model's output for a given input), verification (whether a person can check if the model's prediction is correct), and "what-if" reasoning (whether a person can predict how the output changes if the input is slightly modified) [117, 63]. Results consistently show that for rule- and tree-based models are more "interpretable" as models involving equations like linear models or neural networks, users are faster

1. Parts of this work was presented at the European Workshop in Reinforcement Learning in 2024 and at the Workshop on Programmatic Reinforcement Learning in 2025 :https://openreview.net/group?id=EWRL/2024/Workshop#tab-accept, https://prl-workshop.github.io/

and more accurate when asked to simulate or verify their predictions, with trees and rule sets both in terms of speed and correctness [42, 43, 83, 130, 56]. Indeed, when using neural networks or linear models, human participants struggle to simulate or reason about their predictions, largely because these models impose a higher operation count, i.e., more mental steps needed to reproduce a decision [63].

Linear models can also be relatively easy for users to simulate when they involve only a small number of parameters [66]. Indeed, as stated in [42], optimizing interpretability can also be seen as regularizing the learned model.

Finally, for a fixed class of models, humans gave different values of interpretability to models with different numbers of parameters [66], e.g. a tree with less nodes was deemed more interpretable than a tree with lots of nodes.

Some works [73, 37] argue that model interpretability can only be measured by the the specific end user and only for the specific downstream tasks. This prompts a methodology that could give good measures of interpretability without requiring to perform time and resource consuming user studies. However, measuring model interpretability without humans is still an open problem [49] and there probably do not exist a decisive solution.

8.2 The mythos of interpretability

Work comparing interpretability of models inside the same class [35, 36, 71, 72, 84, 12, 124] do not have to worry about their methodology since the number of model parameters is enough as a measure [66]. However comparing the interpretability of models from different classes, without user studies, requires caution : which is more interpretable, a depth-50 decision tree or a neural network with two layers of 16 neurons (figure)?

Work that compare model interpretability across different model classes without humans, rely on the notion of *simulatibility* introduced in [73]. Inspired, by results from user studies, Zachary Lipton postulates that interpretability of a model should relate to two things. First, it should relate to how a user simulates the prediction of the model given an input [117, 63]. Second, it should relate to how a user gets a global idea of the model's internals, i.e. without any inputs, what does the user understands from reading the model.

This notion of simulatibility is closely related to space and time complexities of programs. In [12], authors use verification time as a proxy to compare the interpretability of neural network policies and imitated decision tree policies. Since decision trees and

neural networks are implemented differently in their experiments, they can't use the same verification softwares ([30] against [137]) which makes it difficult to conclude if the verification times are different because of interpretability or because of the computations hidden in the verifiers. Similarly, in [78] authors compare the inference speed of neural network policies to first-order logic policies [32] but one uses GPU acceleration while the other is run on CPU.

If one wants to use the notion of simulatibility and ultimately of time or space complexity of programs as a proxy for interpretability, one has to implement and compare policies from different classes similarly.

The key contribution of this part is to offer a sound methodology to compare the interpretability-performance trade-offs of models from different model classes without requiring humans.

8.3 Methodology overview

While our methodology could be applied to evaluate the interpretability-performance trade-offs of different model classes when optimizing the supervised learning objective (cf. definition 4), we focus in this part on sequential decision making tasks.

One might argue that once the policy is trained, one could use directly the actual space or time complexity of the policy as the measure of interpretability. However, we argue that for decision tree-like models, the time complexity of a prediction will be different for each input (tree traversal will differ) and computing an expected complexity is difficult as the inputs distribution can only be obtained by running the policy in the MDP it was trained for (e.g. with Monte-Carlo estimates). Hence, we use the two metrics presented next in order to evaluate policy interpretability without requiring human intervention :

1. Policy Inference Time : we measure policy inference time in seconds given states averaged over many trajectories. *2. Policy Size :* we measure the policy 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.

Those two metrics are proxies for the notion of simulatability [73] that gives insights on how a human being would read a policy to understand how actions are inferred. Measure the average inference time over entire trajectories rather than over different states could also make sense. However it would also be difficult to make sense of the results as different policies can have different trajectory lengths.

```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[1]+0.575
    y1 = -0.205*s[0]+22.592*s[1]-0.63
    y2 = -0.763*s[0]+8.237*s[1]+0.054
    max_val = y0
    action = 0
    if y1 > max_val:
        max_val = y1
        action = 1
    if y2 > max_val:
        action = 2
    s, r, terminated, truncated, \
    infos = env.step(action)
    done = terminated or truncated
```

FIGURE 8.1 – 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 8.2 – Unfolded tiny rely neural network for Pendulum.

Most importantly, 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 policies into a simple representation that mimics how a human being "reads" a policy. We call this process of standardizing policies language “unfolding”.

In figures 8.1, 8.2 and 9.2, we present some unfolded policy programs. Other works have distilled neural networks into programs [131] or even directly learn programmatic policies [103] 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 [126]. We will discuss later on the limitations of unfolding policies in the overall methodology.

Chapitre 9

Validating our methodology

In this chapter, we validate our methodology to evaluate the interpretability. In particular we verify that our methodology that relies on policy unfolding can retrieve conclusions from user studies, e.g. we want that our measures of interpretability change with parameter number inside a given policy class.

Because in this part of the manuscript we do evaluate learning capabilities of interpretable machine learning algorithms but solely their outputs, we use indirect imitation learning (cf. section 4) to obtain policies from different classes. Imitation learning is a great baseline to obtain policies from different classes. Indeed, as long as there exists a supervised learning algorithm for a given model class, this algorithm can be used in e.g. algorithms 8 or 9 to distillate the behavior of any expert policy into the desired policy class. Furthermore, if policies are already available for some MDPs, we can simply re-use them as experts in e.g. algorithms 8 or 9.

9.1 Obtaining policies from different classes

9.1.1 Policy classes

Policy Class	Parameters	Training algorithm
Linear policies	Determined by state-action dimensions	Linear/logistic Regression
Decision trees	{4, 8, 16, 64, 128} nodes	CART
Oblique decision trees	{4, 8, 16, 64, 128} nodes	CART
Relu neural networks	{(2 ,2), (4, 4), (8, 8), (16, 16)} weights	SGD

Tableau 9.1 – Summary of policy classes parameters and supervised learning algorithms to fit experts.

```
def play(x):
  if (x[12] - x[15]) <= 8.5:
      if (x[2] - x[7]) <= 139.5:
          if (x[12] - x[13]) <= 5.5:
              if (x[2] - x[7]) <= 130.5:
                  if (x[3] - x[14]) <= 110.5:
                      if (x[5] - x[11]) <= 13.5:
                          return 5
                      else:
                          return 4
                  else:
                      return 5
              else:
                  if (x[3] - x[11]) <= 173.5:
                      return 4
                  else:
                      return 1
          else:
              if (x[1] - x[2]) <= -80.5:
                  if (x[1] - x[3]) <= -28.5:
                      return 5
                  else:
                      return 3
              else:
                  if (x[3] - x[9]) <= 73.5:
                      return 2
                  else:
                      if (x[2] - x[17]) <= 100.5:
                          return 2
                      else:
                          return 4
      else:
          return 4
  else:
      if (x[1] - x[13]) <= 156.5:
          if (x[3] - x[12]) <= 143.0:
              if (x[1] - x[14]) <= 114.5:
                  return 4
              else:
                  return 1
          else:
              if (x[2] - x[11]) <= 150.5:
                  return 0
              else:
                  return 1
      else:
          return 4
```

FIGURE 9.1 – Unfolded oblique decision tree policy with 11 nodes for Pong (RL objective value, i.e. episodic rewards of 16).

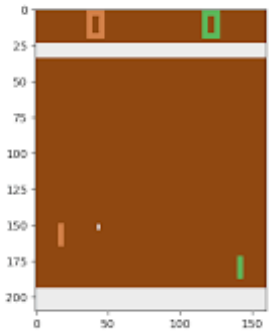


FIGURE 9.2 – The Pong game on Atari 2600.

We consider four policy classes for our baselines. We consider linear policies that have been shown to be able to solve MuJoCo tasks [80]. We fit linear policies to expert policies using simple linear (logistic) regressions with scikit-learn [97] default implementation. We also consider decision trees [20] and oblique decision trees [94]. Oblique decision trees are decision trees which does not necessarily partitions the input domain parallel to axes. A test node in an oblique decision tree can be the linear combinations of two state features. We give an example of an oblique decision tree unfolded for the Pong game in figure 9.1. We train trees using the default CART (cf. algorithm 1) implementation of scikit-learn with varying numbers of parameters (number of nodes in the tree).

Envs	BC 50K	BC 100K	Dagger 50K	Dagger 100K	VIPER 50K	VIPER 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 9.2 – Repetitions of each imitation learning algorithm on each environment. We specify from which deep RL algorithms expert policies from the zoo [106] come from in parentheses.

We explain in more details how we adapt CART to return oblique decision trees in appendix C. We also consider neural networks with relu activations [55] with varying number of parameters (total number of weights).

In table 9.1 we summarize the different policy classes we will consider for our study. There is one class of linear policies, five classes of decision trees and oblique decision trees, each defined by the maximum number of nodes in the tree, and four classes or relu networks each defined by the number and sizes of hidden layers.

9.1.2 Which expert policies to imitate with which algorithm ?

We do not have to re-run deep reinforcement learning algorithms [89, 112, 54], we can simply use the pre-trained policies from the stables-baselines3 zoo [106] and try our methodology to evaluate their interpretability. Depending on the MDPs, also called environments, we choose neural network policies trained with different deep reinforcement learning algorithms. It is important to note that not all experts are compatible with all the variants of imitation learning algorithms. Indeed, SAC experts [54] are not compatible with VIPER [12] because the latter only works for discrete actions. We do not use PPO experts with VIPER either : despite working with discrete actions PPO do not compute a Q-function necessary for the samples re-weighting in VIPER. All experts are compatible with Dagger [111]. We also consider an additional imitation learning algorithm : behavior cloning (BC) [99]. BC is essentially just one iteration of Dagger imitation i.e. samples are only collected with the expert policy once before fitting a new policy class. For Dagger and VIPER we use 10 iterations of samples collections with experts and teacher policy (cf. algorithms 8 and 9) with different total samples budget. In table 9.2, we summarize the different algorithms used to imitate each expert. We also specify the total number of samples collected during the imitations as well as the number of times we repet the imitations for statistical significance.

9.1.3 Which environments to consider ?

We evaluate the interpretability of policies for common environments in reinforcement learning research. We consider the classic control tasks from gymnasium [125], MuJoCo robots from [123], and Atari games from [13]. 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 [34] in which states are objects in the frame and their positions.

In table 9.3, we summarize which environment we consider with their state-action space sizes and a threshold representing the rewards of a "good" policy. In total, we compute across policy classes, imitation algorithms, experts origins, and environments, roughly 40 000 unique policies.

Using those 40 000 policies as baselines, we describe next the experimental setup used to evaluate their interpretability using our methodology. 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.

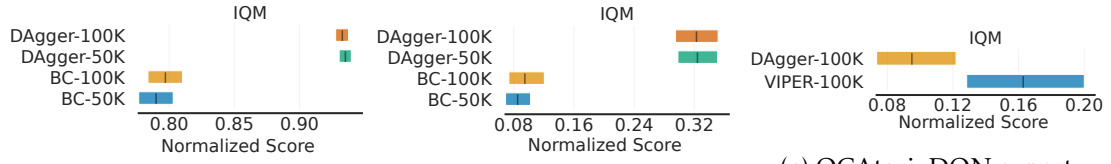
Classic	**MuJoCo**	**OCAtari**
CartPole (4, 2, **490**)	Swimmer (8, 2, **300**)	Breakout (452, 4, **30**)
LunarLander (8, 4, **200**)	Walker2d (17, 6, **2000**)	Pong (20, 6, **14**)
// Continuous (8, 2, **200**)	HalfCheetah (17, 6, **3000**)	SpaceInvaders (188, 6, **680**)
BipedalWalker (24, 4, **250**)	Hopper (11, 3, **2000**)	Seaquest (180, 18, **2000**)
MountainCar (2, 3, **90**)		
// Continuous (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 [125]. For OCAtari environments, we choose the thresholds as the minimum between the DQN expert from [106] and the human scores. We also adapt subjectively some thresholds that we find too restrictive especially for MuJoCo (for example, the PPO expert from [106] has 2200 reward on Hopper while the default threshold was 3800).

9.2 Running the imitation learning algorithms

Using the reinforcement learning evaluation library rliable [1], we plot on figure 9.3 the interquantile 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.



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

Figure 9.3 – Performance of different imitation learning algorithms on different environments. We plot the 95% stratified bootstrapped confidence intervals around the IQMs.

9.2.1 What is the best imitation algorithm?

The key observation is that for tested environments (cf. figures 9.3a, 9.3b), behavior cloning is not an efficient way to train baseline policies compared to Dagger. This is probably because BC trains a policy to match the expert's actions on states visited by the expert, while Dagger trains a policy to take the expert's actions on the states visited by the trained policy [111]. An other observation is that the best performing imitation algorithms for MuJoCo (Dagger from figure 9.3b) and OCAtari (VIPER from figure 9.3c) obtain baselines that in average cannot match well the performances of the experts. However baseline policies almost always match the experts on simple tasks like classic control (cf. figure 9.3a).

9.2.2 What is the best policy class in terms of reward?

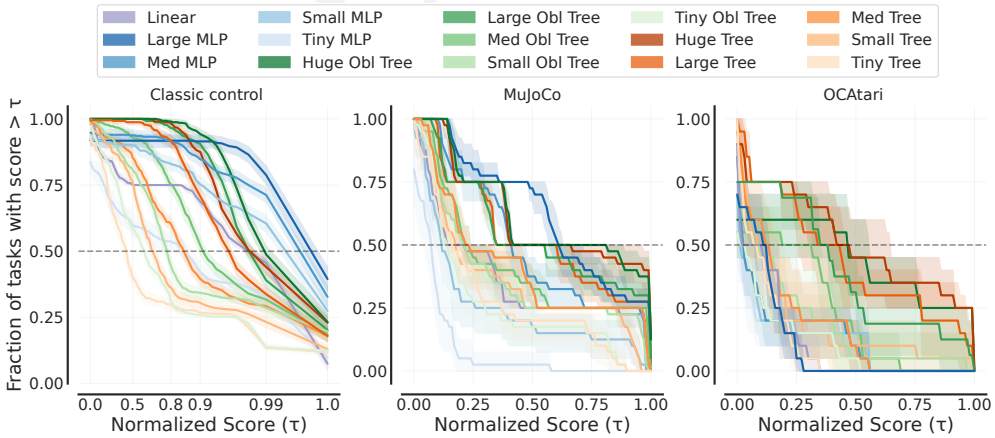


Figure 9.4 – Performance profiles of different policy classes on different environments.

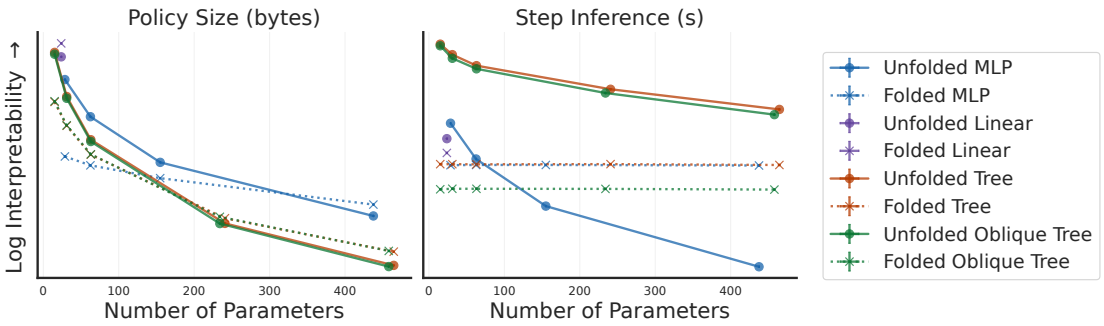


FIGURE 9.5 – Policies interpretability on classic control environments. We plot 95% stratified bootstrapped confidence intervals around means in both axes. In each sub-plot, interpretability is measured with either bytes or inference speed.

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 9.3. From figure 9.4 we see that on classic control environments, neural networks tend to perform better than other classes while on OCAtari games, trees tend to perform better than other classes. Now we move on to the interpretability evaluation.

9.3 Is our methodology sound with respect to user studies ?

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 (cf. section 8.3) obtained for classic control environments with Dagger-100K. To unfold policies, we convert them into Python programs formatted with PEP 8 (comparing other unfolding formats such as ONNX [1] is left to future work). We ensure that all policies operations are performed sequentially, similarly to how a human would operate, and compute the metrics for each policy on 100 episodes using the same CPUs.

9.3.1 Is it necessary to unfold policies to compute interpretability metrics ?

We see on figure 9.5 that folded policies of the same class almost always give similar interpretability values (dotted lines) despite having very different number of parame-

1. https://github.com/onnx/onnx

ters. 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 [66].

9.3.2 Is there a best policy class in terms of interpretability ?

User studies from [43, 83, 130] show that decision trees are easier to understand than models involving mathematical equations like oblique trees, linear maps, and neural networks. However, [73] states that for a human wanting to have a global idea of the inference of a policy, a compact neural network can be more interpretable than a very deep decision tree. In figure 9.5, we show that inference speed and memory size of unfolded policies 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 neural networks. However, when measuring interpretability with policy size, we observe that neural networks can be more interpretable than trees for similar number of parameters. Because there seem to not be a more interpretable policy class across proxy metrics, we will keep studying both metrics at the same time.

9.4 Discussion

To validate the methodology proposed in the previous chapter, we imitated pre-trained experts neural network policies with various less complex policies. We obtain diverse baselines policies with diverse interpretability measures and performances for a wide range of environments. For each environment, we save the best baseline policy of each class in terms of RL objective (cf. definition 4) after unfolding. We open source those best in class policies here : `https://github.com/KohlerHECTOR/interpretable-rl-zoo`. We hope that those transparent policies can help teaching and research in (interpretable) sequential decision making. In the next chapter, we focus on those best in class unfolded policies and study how they trade off our metrics of interpretability and the RL objective.

Chapitre 10

Interpretability-performance trade-offs

In this chapter, we study the interpretability-performance trade-offs of different policies. For each environment considered we experiment with the best in class policies obtained in the previous chapter. For each environment we hence study five decision tree policies with varying number of nodes, five oblique decision tree policies with varying number of nodes, one linear policy, and four relu neural network policies with varying number of hidden units (cf. table 9.1). To obtain fair interpretability measurements, in our case inference speeds in seconds or sizes in bytes (cf. section 8.3), we run each unfolded policy **on the same dedicated CPU** for 100 new environment episodes. Since all the policies process computations roughly the same way, and since they are executed on the exact same device in the same conditions we believe the results presented next are a good illustration of our methodology (cf. section 8.3) can be used for interpretability research.

10.1 Is it possible to compute interpretable policies for high-dimensional environments ?

In figure 10.1, we present results only for part of the environments that we believe are the most representative. This allows us to have a compact comparison of policy trade-offs with either of the metrics from section 8.3. The full trade-offs are presented in appendix C.1. In [49], authors claim that computing an interpretable policy for high-dimensional MDPs, i.e. environments where the number of state features is high, is

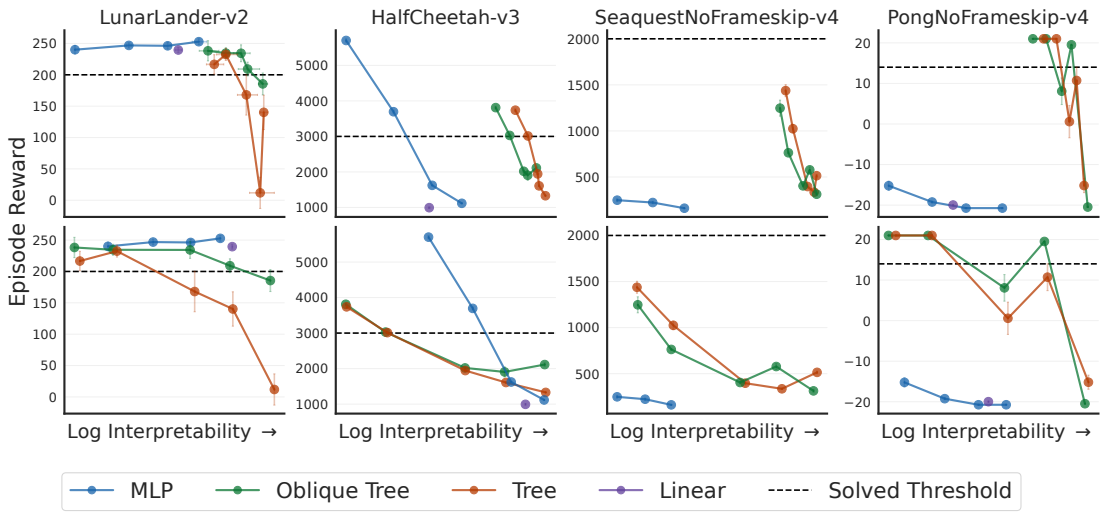


FIGURE 10.1 – interpretability-Performance trade-offs. Top row, interpretability is measured with step inference times. Bottom row, the interpretability is measured with policy size. We plot 95% bootstrapped confidence intervals around means on both axes.

difficult since it is similar to program synthesis which is known to be NP-hard [53]. Using our measures of interpretability, we can corroborate this claim. On figure 10.1, we can indeed observe that some relatively interpretable policies can solve object-centric Pong (20 state features) or HalfCheetah (17 state features). For example, the oblique tree policy from figure 9.1, represented by the second right most green dot on figure 10.1, is among the most interpretable policies for Pong w.r.t to both policy size and policy inference time, and has RL objective value (cf. definition 4) above the solving threshold. However, for very high-dimensional environments like Seaquest (180 state features), none of our policies, even the less interpretable ones (relu networks with two hidden layer units of size sixteen), can solve the game. This observation resonates with the first part of the manuscript where we showed that even in controlled experiments where an interpretable policy is optimal, no RL algorithm can learn it. This yields the following question.

10.2 For what environment are there good interpretable policies?

We fitted a random forest regressor [21] to predict the interpretability values of our best in class policies using environment attributes. In table 10.1, we report the

Environment Attributes	Importance for Step inference	Importance for Policy size
State features	**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 10.1 – Environment attributes importance to predict interpretability of the best in class policies obtained in chapter 9.

importance of each environment attribute when it comes to accurately predicting interpretability scores. We show that as hinted previously, the number of state features 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.

10.3 How does interpretability influence performance ?

In [80, 82], authors show the existence of linear and tree policies respectively that solve MuJoCo and continuous maze environments respectively. This means 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 10.1 we observe that on, e.g., LunarLander, increasing policy interpretability up to a certain point does not decrease reward. For example, for neural networks, all the the blue dots, both for the policy size and policy inference time metrics, are above the solving thresholds, which means that relu networks with two hidden layers of two units each perform as well as a relu network with two hidden layers of sixteen units each. Surprisingly, we can observe that for Pong a minimum level of interpretability is required to solve the game. Indeed, as stated in [42], 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 metric, we see on figure 10.1 that for LunarLander it is a neural network that achieves the best trade-off while for Pong it is a tree. This means that, when research interpretability one needs to be extra careful when using model interpretability assumptions presented in

e.g. figure 4. Next, we compare our proxies for interpretability with another one ; the verification time of policies used in [12, 9].

10.4 Verifying interpretable policies

[9] states that the cost of formally verifying properties of a relu neural network scales exponentially with the number of parameters (hidden units). Hence, they propose to measure interpretability of a policy as the computations required to verify properties of actions given state features subspaces, what they call local explainability queries [52]. Interestingly, VIPER (cf. algorithm 9) was designed in hope to get policies tht were easy to verify. In practice, this amounts to passing a subspace of state features and a subspace of (continuous) action features and solving the SAT problem of finding at least one state in the subspace for which the policy outputs an action in the action subspace. For example, for the LunarLander problem, a verification 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 returns `True`, then there is a risk that the lander crashes when deployed because it might push towards the ground even at low altitude.

Designing interesting queries covering all risks is an open problem, hence to evaluate the verification times of our best in class policies, we generate 500 random queries per environment by sampling state and action subspaces uniformly. 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 example query mentioned above to be UNSAT. We also only verify instances of neural networks and linear policies using the solver [137] for this experiment as verifying decision trees requires a different software [30] for which verification times would not be comparable. We verify each policy using the same verification algorithm with the same hyperparameters and queries on the same CPU to obtain faire measurements of verification times. We leave it to future work to write decision tree policies as relu networks to then compare their verification times fairly. Indeed it is known that oblique decision trees can be written as a relu neural networks [68].

On figure 10.2, we can observe that verification time decreases exponentially with our measures of interpretability as shown theoretically in [9]. This is another good validation of our proposed methodology as well as a motivation to learn interpretable policies.

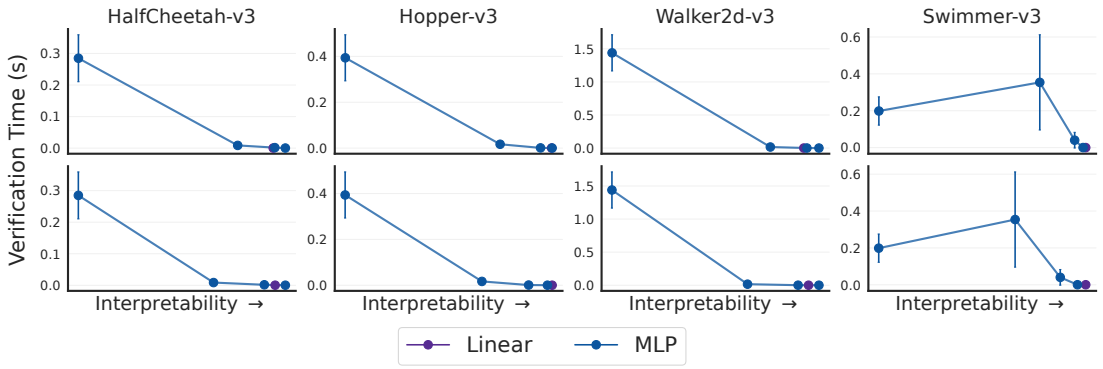


FIGURE 10.2 – Verification time as a function of policy interpretability. Top row, interpretability is measured with step inference times. Bottom row, the interpretability is measured with policy size. We plot 95% confidence intervals around means on both axes.

10.5 Limitations and conclusions

In this part of the manuscript, we have proposed to write policies for MDPs from different model classes in a unified language. We believe that unfolding policies (cf. section 8.3) allows for fair measurements of interpretability proxies. In particular, we unfolded thousands of policies from different model classes (cf. table 9.1), and showed that comparing inference speed and sizes gave conclusions about interpretability similar to existing user studies (cf. section 9).

Using the proposed methodology, we were able to illustrate the trade-offs between the RL objective (cf. definition 4) at the core of this manuscript and interpretability of policies. We showed the crucial need of careful methodology as different different policy classes yield different trade-offs on different environment : decision trees are not always more interpretable than neural networks.

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 [80].

Furthermore, there should be no limitation to use our methodology to evaluate the interpretability of arbitrary compositions of linear policies, trees and oblique trees, and neural networks, such as the hybrid policies from [114].

Even though using episodic inference does not change the trade-offs compared to step inference time (cf. appendix C.2 and C.3), 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 neural network policies even thought this could greatly influence the inference time. For completeness, when unfolding policies, we should delete operations including near-zero parameters. In the future we should also evaluate the influences of different unfolding procedures. For example, the number of significant digits in the values of parameters might affect the measures of interpretability.

Most importantly, we should further validate our methodology with user studies of the unfolded policies we open sourced [1].

We hope that our methodology as well as the provided baselines will pave the way to a more rigorous science of interpretable reinforcement learning.

1. https://github.com/KohlerHECTOR/interpretable-rl-zoo

General conclusion

In this manuscript we have tackled the problem of interpretable sequential decision making. We have studied interpretability through the prism of model classes used as policies for Markov decision processes or predictors for classification tasks. We put great emphasis on decision tree models that are often considered more interpretable than neural networks, now used in many machine learning applications.

In the first part of this manuscript, we studied algorithms that learn non-parametric decision tree policies for Markov decision processes (cf. chapter 1). In particular, we studied the reinforcement learning algorithms from [124] that use the reward signal of some augmented Markov decision process to train decision tree policies that optimize some trade-off of interpretability and performance in a sequential decision task. In chapter 2, we performed a reproducibility study of those algorithms and concluded that imitation learning algorithms, despite not optimizing the performances on the downstream tasks, yielded better decision tree policies with similar number of internal nodes. To better understand why directly optimizing an interpretability-performance trade-off using reinforcement learning yielded bad decision tree policies, we showed in chapter 3 that the latter could be re-written as an optimization problem in a partially observable Markov decision processes. Through extensive experimentation with standard and specialized reinforcement learning algorithms in a very controlled environment, we identified partial observability as the main failure mode of direct reinforcement learning for decision tree policies. To further corroborate this conclusion, we showed that for the same class of partially observable Markov decision problems where this time the observations contained all the information about the hidden state, those reinforcement learning algorithms converged to the optimal decision tree policies. Surprisingly, this class of easy to solve partially observable Markov decision problems contains supervised learning problems.

The previous observation prompted the design of a new decision tree induction algorithm for supervised learning. In the second part of this manuscript, we introduced

dynamic programming decision trees (DPDT), a novel decision tree induction framework. In chapter 6, we showed how to construct a Markov decision process where states are training data and actions are candidate decision tree nodes. In chapter 7, we showed that adaptively choosing actions based on some greedy heuristics and then solving the Markov decision process with dynamic programming yielded state-of-the-art decision tree classifiers. Extensive benchmarking highlights that DPDT computes near-optimal decision trees in a fraction of the time required by existing optimal solvers. Most importantly, we showed that DPDT can be competitive in many machine learning applications as its generalization capabilities when generating single or ensemble of trees are also state-of-the-art compared to greedy and optimal algorithms. While this part of the manuscript may have seemed orthogonal to the other two as the primary focus of algorithms studied there is downstream task performances rather than interpretability, this justifies studying interpretability for more than just transparency and safety, e.g. for performances.

In the last part of this manuscript, we reflect on the notion of interpretability of a model and how to measure it. While it seems obvious that interpretability only exists with respect to a user and a task, the speed at which machine learning is moving as a research field prompts new methodologies that don't require humans. In chapter 8, we proposed to measure the interpretability of a given model with inference speed and size in memory. Those two metrics echo the notion of simulatability introduced in [73] in which the interpretability of a model is related to the classical notion of complexity in computer science. In chapter 9, we showed that for those measures to be meaningful when comparing models from different classes, one needs to unfold models in a common language such as python. Hence, after sequentializing all operations inside a model and running them on the same hardware, we obtain measures for proxies of interpretability that we can meaningfully compare and that aligns with existing results from user studies. In chapter 10, we show that there is no model class that obtain better interpretability-performance trade-off across all tasks. This results further highlights the need for careful methodology when researching interpretability as it might not be enough to rely on the assumption that neural networks are less interpretable than decision trees.

We already identified specific future research directions in sections 4.2, 7.4 and 10.5. Here, we discuss broader perspectives on interpretable sequential decision making, focusing on two key challenges in the field.

The first challenge concerns the fundamental tension between model interpretability and performance. Black-box models like large language models (LLMs) [129], which

achieve unprecedented performance but lack transparency, are increasingly studied. While interpretability research often aims to ensure model safety, this approach faces two significant obstacles. First, constraining models to be interpretable typically results in lower performance compared to advanced neural architectures. Second, in practice the indirect interpretation of complex models only provides local understanding (cf. figure 5), undermining the safety guarantees we seek. This limitation is intrinsic to emerging approaches like mechanistic interpretability and chain-of-thoughts analysis [10] that exclusively focus on local interpretations of LLMs.

However, this tension also reveals opportunities. Not all effective machine learning requires large, opaque models. Formally verifiable relu networks, small neural networks, and decision trees (as demonstrated in this manuscript) can achieve strong generalization while remaining interpretable. The challenge lies in identifying domains where these models can be most impactful.

The second challenge involves identifying contexts where interpretability is genuinely necessary. Surprisingly, traditional justifications for interpretability may be weaker than assumed. For instance, studies show that in medical applications, doctors rarely examine model interpretations even when available [95]. This suggests that human trust in AI systems might actually reduce the perceived need for interpretability.

Paradoxically, interpretability might be most crucial in systems where human oversight is minimal or impossible after deployment. Consider two contrasting scenarios. In a military drone system where humans make final decisions, the need for interpretability might seem less critical. However, in an autonomous spacecraft controlled by a neural network, where human intervention is impossible post-deployment, interpretability through verification becomes essential. Yet this raises another question : is developing a comprehensive verification framework (cf. section 10.4) any more efficient than designing a traditional, interpretable controller ?

We conclude that the most challenging aspect of interpretability research does not lie in developing new methods, but in rigorously identifying where and why interpretability is truly indispensable.

Bibliographie

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

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

[3] Arash Ahmadian et al. *Back to Basics : Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs*. 2024. arXiv : 2402.14740 [cs.LG]. url : https://arxiv.org/abs/2402.14740.

[4] Mauricio Araya-López et al. « A Closer Look at MOMDPs ». In : *Proceedings of the 22nd International Conference on Tools with Artificial Intelligence*. Proceedings of the 22nd International Conference on Tools with Artificial Intelligence. Arras, France : IEEE, oct. 2010. url : https://inria.hal.science/inria-00535559.

[5] Akanksha Atrey, Kaleigh Clary et David Jensen. « Exploratory Not Explanatory : Counterfactual Analysis of Saliency Maps for Deep Reinforcement Learning ». In : *International Conference on Learning Representations*. 2020. url : https://openreview.net/forum?id=rkl3m1BFDB.

[6] Mohammad Gheshlaghi Azar, Ian Osband et Rémi Munos. « Minimax regret bounds for reinforcement learning ». In : *International conference on machine learning*. PMLR. 2017, p. 263-272.

[7] Andrea Baisero et Christopher Amato. « Unbiased Asymmetric Reinforcement Learning under Partial Observability ». In : *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '22. Virtual Event, New Zealand : International Foundation for Autonomous Agents et Multiagent Systems, 2022, p. 44-52. isbn : 9781450392136.

[8] Andrea Baisero, Brett Daley et Christopher Amato. « Asymmetric DQN for partially observable reinforcement learning ». In : *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*. Sous la dir. de James Cussens et Kun Zhang. T. 180. Proceedings of Machine Learning Research. PMLR, jan. 2022, p. 107-117. url : https://proceedings.mlr.press/v180/baisero22a.html.

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

[10] Fazl Barez et al. « Chain-of-Thought Is Not Explainability ». In : (2025).

[11] Andrew G. Barto, Richard S. Sutton et Charles W. Anderson. « Neuronlike adaptive elements that can solve difficult learning control problems ». In : *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5 (1983), p. 834-846. doi : 10.1109/TSMC.1983.6313077.

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

[13] 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.

[14] Richard Bellman. *Dynamic Programming*. 1957.

[15] 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.

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

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

[18] 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.

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

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

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

[22] 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.

[23] 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.

[24] 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.

[25] Ayman Chaouki. « Optimal Decision Trees via Search : A Reinforcement Learning framework ». Theses. Institut Polytechnique de Paris ; Université de Waikato, nov. 2024. url : https://theses.hal.science/tel-04970181.

[26] 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.

[27] 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.

[28] Samuel Ping-Man Choi, Nevin Lianwen Zhang et Dit-Yan Yeung. « Solving Hidden-Mode Markov Decision Problems ». In : *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*. Sous la dir. de Thomas S. Richardson et Tommi S. Jaakkola. T. R3. Proceedings of Machine Learning Research. Reissued by PMLR on 31 March 2021. PMLR, avr. 2001, p. 49-56. url : https://proceedings.mlr.press/r3/choi01a.html.

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

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

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

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

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

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

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] Alain Dutech et Bruno Scherrer. « Partially Observable Markov Decision Processes ». In : *Markov Decision Processes in Artificial Intelligence*. John Wiley Sons, Ltd, 2013. Chap. 7, p. 185-228. isbn : 9781118557426. doi : https://doi.org/10.1002/9781118557426.ch7. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118557426.ch7. url : https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118557426.ch7.

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

[41] 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.

[42] 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.

[43] 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.

[44] 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/pinterii/S002200009791504X.

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

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

[47] 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.

[48] 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.

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

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

[51] 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.

[52] 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.

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

[54] 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.

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

[56] Johan Huysmans et al. « An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models ». In : *Decis. Support Syst.* 51.1 (avr. 2011), p. 141-154. ISSN : 0167-9236. DOI : 10.1016/j.dss.2010.12.003. URL : https://doi.org/10.1016/j.dss.2010.12.003.

[57] 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.

[58] Tommi Jaakkola, Satinder P. Singh et Michael I. Jordan. « Reinforcement learning algorithm for partially observable Markov decision problems ». In : *Proceedings of the 8th International Conference on Neural Information Processing Systems*. NIPS'94. Denver, Colorado : MIT Press, 1994, p. 345-352.

[59] 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.

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

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

[62] 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.

[63] Isaac Lage et al. *An Evaluation of the Human-Interpretability of Explanation*. 2019. arXiv : 1902.00006 [cs.LG]. url : https://arxiv.org/abs/1902.00006.

[64] Gaspard Lambrechts, Adrien Bolland et Damien Ernst. « Informed POMDP : Leveraging Additional Information in Model-Based RL ». In : *Reinforcement Learning Journal* 2 (2025), p. 763-784.

[65] Gaspard Lambrechts, Damien Ernst et Aditya Mahajan. « A Theoretical Justification for Asymmetric Actor-Critic algorithms ». In : *Forty-second International Conference on Machine Learning*. 2025. url : https://openreview.net/forum?id=F1yANMCnAn.

[66] 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.

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

[68] Guang-He Lee et Tommi S. Jaakkola. « Oblique Decision Trees from Derivatives of ReLU Networks ». In : *International Conference on Learning Representations*. 2020. url : https://openreview.net/forum?id=Bke8UR4FPB.

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

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

[71] 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.

[72] 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.

[73] 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.

[74] Michael L. Littman. « Memoryless policies : theoretical limitations and practical results ». In : *Proceedings of the Third International Conference on Simulation of Adaptive Behavior202f : From Animals to Animats 3 : From Animals to Animats 3*. SAB94. Brighton, United Kingdom : MIT Press, 1994, p. 238-245. isbn : 0262531224.

[75] John Loch et Satinder P. Singh. « Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes ». In : *Proceedings of the Fifteenth International Conference on Machine Learning*. ICML '98. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1998, p. 323-331. isbn : 1558605568.

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

[77] Scott M. Lundberg et Su-In Lee. « A unified approach to interpreting model predictions ». In : *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA : Curran Associates Inc., 2017, p. 4768-4777. isbn : 9781510860964.

[78] Lirui Luo et al. « End-to-End Neuro-Symbolic Reinforcement Learning with Textual Explanations ». In : *International Conference on Machine Learning (ICML)* (2024).

[79] Horia Mania, Aurelia Guy et Benjamin Recht. « Simple random search of static linear policies is competitive for reinforcement learning ». In : *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada : Curran Associates Inc., 2018, p. 1805-1814.

[80] 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.

[81] 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.

[82] 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.

[83] 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.

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

[85] 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.

[86] Hayden McTavish et al. « Fast sparse decision tree optimization via reference ensembles ». In : *Proceedings of the AAAI conference on artificial intelligence*. T. 36. 9. 2022, p. 9604-9613.

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

[88] 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.

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

[90] 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.

[91] 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.

[92] 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.

[93] 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. Montreal, Quebec, Canada : Morgan Kaufmann Publishers Inc., 1995, p. 1025-1031. isbn : 1558603638.

[94] 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.

[95] Myura Nagendran et al. « Eye tracking insights into physician behaviour with safe and unsafe explainable AI recommendations ». In : *NPJ Digital Medicine* 7.1 (2024), p. 202.

[96] 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.

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

[98] Lerrel Pinto et al. *Asymmetric Actor Critic for Image-Based Robot Learning*. 2017. arXiv : 1710.06542 [cs.RO]. url : https://arxiv.org/abs/1710.06542.

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

[100] 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.

[101] Nikaash Puri et al. « Explain Your Move : Understanding Agent Actions Using Specific and Relevant Feature Attribution ». In : *International Conference on Learning Representations*. 2020. url : https://openreview.net/forum?id=SJgzLkBKPB.

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

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

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

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

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

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

[108] « Regional Tree Regularization for Interpretability in Deep Neural Networks ». In : 34 (avr. 2020), p. 6413-6421. doi : 10.1609/aaai.v34i04.6112. url : https://ojs.aaai.org/index.php/AAAI/article/view/6112.

[109] 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.

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

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

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

[113] Wenjie Shi et al. « Self-Supervised Discovering of Interpretable Features for Reinforcement Learning ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.5 (2022), p. 2712-2724. doi : 10.1109/TPAMI.2020.3037898.

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

[115] Andrew Silva et al. « Optimization Methods for Interpretable Differentiable Decision Trees Applied to Reinforcement Learning ». In : *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Sous la dir. de Silvia Chiappa et Roberto Calandra. T. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, p. 1855-1865. url : https://proceedings.mlr.press/v108/silva20a.html.

[116] Satinder P. Singh, Tommi S. Jaakkola et Michael I. Jordan. « Learning without state-estimation in partially observable Markovian decision processes ». In : *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*. ICML'94. New Brunswick, NJ, USA : Morgan Kaufmann Publishers Inc., 1994, p. 284-292. isbn : 1558603352.

[117] Dylan Slack et al. *Assessing the Local Interpretability of Machine Learning Models*. 2019. arXiv : 1902.03501 [cs.LG]. url : https://arxiv.org/abs/1902.03501.

[118] 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.

[119] Edward J. Sondik. « The Optimal Control of Partially Observable Markov Processes Over the Infinite Horizon : Discounted Costs ». In : *Operations Research* 26.2 (1978), p. 282-304. issn : 0030364X, 15265463. url : http://www.jstor.org/stable/169635 (visité le 14/08/2025).

[120] Richard S Sutton et al. « Policy Gradient Methods for Reinforcement Learning with Function Approximation ». In : *Advances in Neural Information Processing Systems*. Sous la dir. de S. Solla, T. Leen et K. Müller. T. 12. MIT Press, 1999. url : https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

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

[122] Gerald Tesauro. « Temporal difference learning and TD-Gammon ». In : *Commun. ACM* 38.3 (mars 1995), p. 58-68. issn : 0001-0782. doi : 10.1145/203330.203343. url : https://doi.org/10.1145/203330.203343.

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

[124] 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.

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

[126] 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.

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

[128] 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.

[129] Ashish Vaswani et al. « Attention is All you Need ». In : *Advances in Neural Information Processing Systems*. Sous la dir. d'I. Guyon et al. T. 30. Curran Associates, Inc., 2017. url : https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[130] 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.

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

[132] 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.

[133] 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.

[134] 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. Macao, P.R.China, 2023. isbn : 978-1-956792-03-4. doi : 10.24963/ijcai.2023/606. url : https://doi.org/10.24963/ijcai.2023/606.

[135] 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.

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

[137] 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.

[138] Tong Zhang. « Solving large scale linear prediction problems using stochastic gradient descent algorithms ». In : *Proceedings of the Twenty-First International Conference on Machine Learning*. ICML '04. Banff, Alberta, Canada : Association for Computing Machinery, 2004, p. 116. isbn : 1581138385. doi : 10.1145/1015330.1015332. url : https://doi.org/10.1145/1015330.1015332.

[139] 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.

[140] 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

Appendix for part I

A.1 Reproducig Topin et. al. 2021

A.2 Tree value computations

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

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

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

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

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

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

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

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

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

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

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

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

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

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

□

Depth-0 decision tree : has only one leaf node that takes a single base action indefinitely. For this type of tree the best reward achievable is to take actions that maximize the probability of reaching the objective → or ↓. In that case the objective value of such tree

is : In the goal state $G = (1, 0)$, the value of the depth-0 tree $\mathcal{T}_0$ is :

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

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

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

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

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

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

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

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

We get :

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

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

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

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

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

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

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

objective value :

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

Solving these equations :

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

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

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

A.3 Hyperparameters

Tableau A.4 – PG Hyperparameter Space (140 combinations)

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

Tableau A.5 – PG-IBMDP Hyperparameter Space (140 combinations)

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

Tableau A.6 – QL Hyperparameter Space (192 combinations)

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

Tableau A.7 – QL-Asym Hyperparameter Space (768 combinations)

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

Tableau A.8 – QL-IBMDP Hyperparameter Space (192 combinations)

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

Tableau A.9 – SARSA Hyperparameter Space (192 combinations)

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

Tableau A.10 – SARSA-Asym Hyperparameter Space (768 combinations)

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

Tableau A.11 – SARSA-IBMDP Hyperparameter Space (192 combinations)

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

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

Tableau A.12 – Best hyperparameters for each algorithm on the POIBMDP problem

Annexe B

Appendix for dynamic programming decision trees (part II)

In this chapter we provide additional experimental results. In table B.2, we compare DPDT trees to CART and STreeD trees using 50 train/test splits of regression datasets from [51]. 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 B.3, 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 B.1 compare DPDT to lookahead depth-3 trees when optimizing Eq.16. Unlike the other greedy approaches, lookahead decision trees [93] 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 7.1. Both lookahead trees and DPDT [1] are in `Python` which makes them slow but comparable.

We also provide the hyperparameters necessary to reproduce experiments from section 7.2 and 7.3.1 in table B.4.

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

Tableau B.1 – 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 B.2 – Mean train and test scores (with standard errors) for regression datasets over 50 cross-validation runs.

	Depth 3						**Depth 5**				
Dataset	DPDT		Optimal		CART		DPDT		Optimal		CA
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train
nyc-taxi	39.0 ± 0.0	38.9 ± 0.2	33.8 ± 0.0	33.8 ± 0.1	39.0 ± 0.0	38.9 ± 0.2	45.8 ± 0.0	45.7 ± 0.2	33.8 ± 0.0	33.8 ± 0.1	42.7 ± 0.0
medical_charges	95.2 ± 0.0	95.2 ± 0.0	90.1 ± 0.0	90.1 ± 0.1	95.2 ± 0.0	95.2 ± 0.0	97.7 ± 0.0	97.7 ± 0.0	90.1 ± 0.0	90.1 ± 0.1	97.7 ± 0.0
diamonds	93.0 ± 0.0	92.9 ± 0.1	90.1 ± 0.0	90.1 ± 0.1	92.7 ± 0.0	92.6 ± 0.1	94.2 ± 0.0	94.0 ± 0.1	90.1 ± 0.0	90.1 ± 0.1	94.1 ± 0.0
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 ± 4.1	32.8 ± 0.1	29.4 ± 1.6	51.5 ± 0.1
house_sales	67.0 ± 0.0	66.0 ± 0.4	65.1 ± 0.0	64.4 ± 0.4	66.8 ± 0.0	66.1 ± 0.4	77.6 ± 0.0	76.1 ± 0.3	65.1 ± 0.0	64.4 ± 0.4	76.8 ± 0.0
superconduct	73.1 ± 0.0	72.7 ± 0.5	70.9 ± 0.0	70.5 ± 0.5	70.4 ± 0.0	69.7 ± 0.5	83.0 ± 0.0	81.7 ± 0.4	70.9 ± 0.0	70.5 ± 0.5	78.2 ± 0.0
houses	51.7 ± 0.0	50.7 ± 0.7	48.5 ± 0.1	47.3 ± 0.7	49.5 ± 0.0	48.4 ± 0.7	69.1 ± 0.0	67.6 ± 0.5	48.5 ± 0.1	47.3 ± 0.7	60.4 ± 0.1
Bike_Sharing	55.2 ± 0.0	54.7 ± 0.5	45.1 ± 0.1	44.8 ± 0.7	48.1 ± 0.0	47.9 ± 0.5	65.2 ± 0.0	63.3 ± 0.5	45.1 ± 0.1	44.8 ± 0.7	59.1 ± 0.0
elevators	48.0 ± 0.0	46.8 ± 1.1	40.2 ± 0.1	38.2 ± 1.0	46.8 ± 0.0	45.5 ± 1.2	65.6 ± 0.0	61.2 ± 1.0	40.2 ± 0.1	38.2 ± 1.0	61.9 ± 0.0
pol	72.2 ± 0.0	71.3 ± 0.6	67.8 ± 0.1	67.5 ± 0.9	72.0 ± 0.0	71.2 ± 0.8	93.3 ± 0.0	92.4 ± 0.3	67.8 ± 0.1	67.5 ± 0.9	92.1 ± 0.0
MiamiHousing2016	62.3 ± 0.0	60.4 ± 0.8	60.8 ± 0.0	58.3 ± 0.8	62.3 ± 0.0	60.6 ± 0.8	79.8 ± 0.0	77.5 ± 0.5	60.8 ± 0.0	58.3 ± 0.8	77.3 ± 0.1
Ailerons	63.5 ± 0.0	62.6 ± 0.7	61.6 ± 0.0	60.3 ± 0.7	63.5 ± 0.0	62.6 ± 0.7	76.0 ± 0.0	72.9 ± 0.6	61.6 ± 0.0	60.3 ± 0.7	75.5 ± 0.0
Brazilian_houses	90.7 ± 0.0	90.3 ± 0.8	89.2 ± 0.0	89.4 ± 0.8	90.7 ± 0.0	90.4 ± 0.8	97.6 ± 0.0	96.6 ± 0.4	89.2 ± 0.0	89.4 ± 0.8	97.3 ± 0.0
sulfur	72.5 ± 0.1	66.6 ± 2.2	35.7 ± 0.1	19.1 ± 6.7	72.0 ± 0.1	68.0 ± 2.2	89.0 ± 0.0	68.4 ± 6.7	35.7 ± 0.1	19.1 ± 6.7	81.8 ± 0.1
yprop_41	6.3 ± 0.0	2.3 ± 0.7	3.6 ± 0.0	1.5 ± 0.4	6.2 ± 0.0	2.1 ± 0.8	13.2 ± 0.0	1.2 ± 1.7	3.6 ± 0.0	1.5 ± 0.4	10.8 ± 0.0
cpu_act	93.4 ± 0.0	92.0 ± 0.6	89.0 ± 0.0	86.5 ± 1.9	93.4 ± 0.0	92.0 ± 0.6	96.5 ± 0.0	94.7 ± 0.5	89.0 ± 0.0	86.5 ± 1.9	96.3 ± 0.0
wine_quality	27.9 ± 0.0	23.3 ± 0.9	25.2 ± 0.0	23.7 ± 0.8	27.7 ± 0.0	24.5 ± 0.8	37.4 ± 0.0	26.7 ± 1.0	25.2 ± 0.0	23.7 ± 0.8	35.9 ± 0.1
abalone	46.3 ± 0.0	39.6 ± 1.6	42.5 ± 0.0	40.4 ± 1.4	43.3 ± 0.0	39.2 ± 1.2	58.6 ± 0.0	44.7 ± 1.8	42.5 ± 0.0	40.4 ± 1.4	54.5 ± 0.0

Tableau B.3 – 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 Accuracy depth-5				
Names	Samples	Features	Classes	DPDT	OCT	MFOCT	BinOCT	CART	DPDT	OCT	MFOCT	BinOCT	CART
balance-scale	624	4	3	90.9%	71.8%	82.6%	67.5%	86.5%	77.1%	66.9%	71.3%	61.6%	76.4%
breast-cancer	276	9	2	94.2%	88.6%	91.1%	75.4%	87.9%	66.4%	67.1%	73.8%	62.4%	70.3%
car-evaluation	1728	6	4	92.2%	70.1%	80.4%	84.0%	87.1%	90.3%	69.5%	79.8%	82.3%	87.1%
hayes-roth	160	9	3	93.3%	82.9%	95.4%	64.6%	76.7%	75.4%	77.5%	77.5%	54.2%	69.2%
house-votes-84	232	16	2	100.0%	100.0%	100.0%	100.0%	99.4%	95.4%	93.7%	94.3%	96.0%	95.1%
soybean-small	46	50	4	100.0%	100.0%	100.0%	76.8%	100.0%	93.1%	94.4%	91.7%	72.2%	93.1%
spect	266	22	2	93.0%	92.5%	93.0%	92.2%	88.5%	73.1%	75.6%	74.6%	73.1%	75.1%
tic-tac-toe	958	24	2	90.8%	68.5%	76.1%	85.7%	85.8%	82.1%	69.6%	73.6%	79.6%	81.0%

Tableau B.4 – Hyperparameter search spaces for tree-based models. More details about the hyperparamters meaning are given in [62].

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

Annexe C

Appendix for part III

Oblique decision trees. One can imitate oracles with programs that make tests of linear combinations of features. Many oracles learn oblique or more complex decision rules over an MDP state space. This is illustrated in figure C.1 where a PPO neural oracle creates oblique partitions of the state-space for the Pong environments. Programs that test only individual features would fail to fit this partition (cf. figure C.1). We thus modify CART [20], an algorithm returning an axes-parallel trees for regression and supervised classification problems, for it to return oblique decision trees. In addition to single feature tests, our oblique trees consider linear combinations of two features with weights 1 and −1, e.g. for MDP states $s_i \in \mathbb{R}^p$, the oblique features values are $s_i^{oblique} = \{s_{i1} - s_{i0}, s_{i2} - s_{i0}, ..., s_{ip} - s_{i0}, ..., s_{ip-1} - s_{ip}\} \in \mathbb{R}^{p^2}$. For example, using an oracle dataset with n state-actions pairs : $(\bar{S}, \bar{A} = \pi^*(\bar{S})) \subsetneq \mathbb{R}^{n\cdot(p+\dim(A))}$, we obtain oblique decision trees by fitting $(\bar{S}, \bar{S}^{oblique}, \bar{A} = \pi^*(\bar{S})) \subsetneq \mathbb{R}^{n\cdot(p(p+1)+\dim(A))}$. Given $\bar{S}$, computing $\bar{S}^{oblique}$ can be done efficiently by computing the values of the lower (or upper) triangles in the $\bar{S} \otimes \bar{S} - (\bar{S} \otimes \bar{S})^T$ tensor (excluding the diagonals).

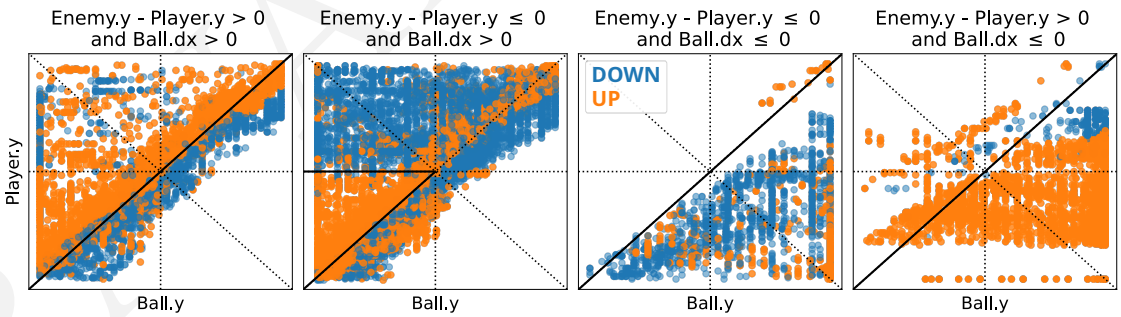


FIGURE C.1 – **Oracle decision rules are oblique** illustrated on PPO for different state space partitions of the Pong environment. Decisions boundaries are both oblique and parallel.

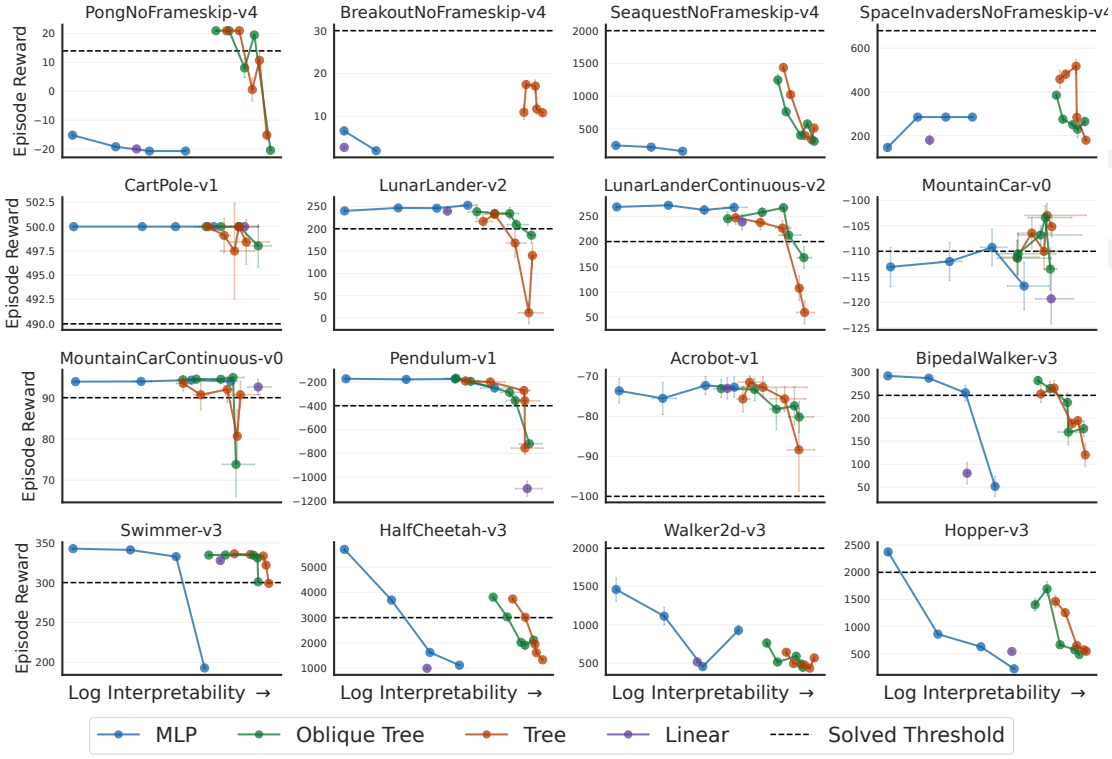


FIGURE C.2 – Trade-off Cumulative Reward vs. Step Inference Time

C.1 All interpretability-performance trade-offs

In this appendix we provide the interpretability-performance trade-offs of all the tested environments.

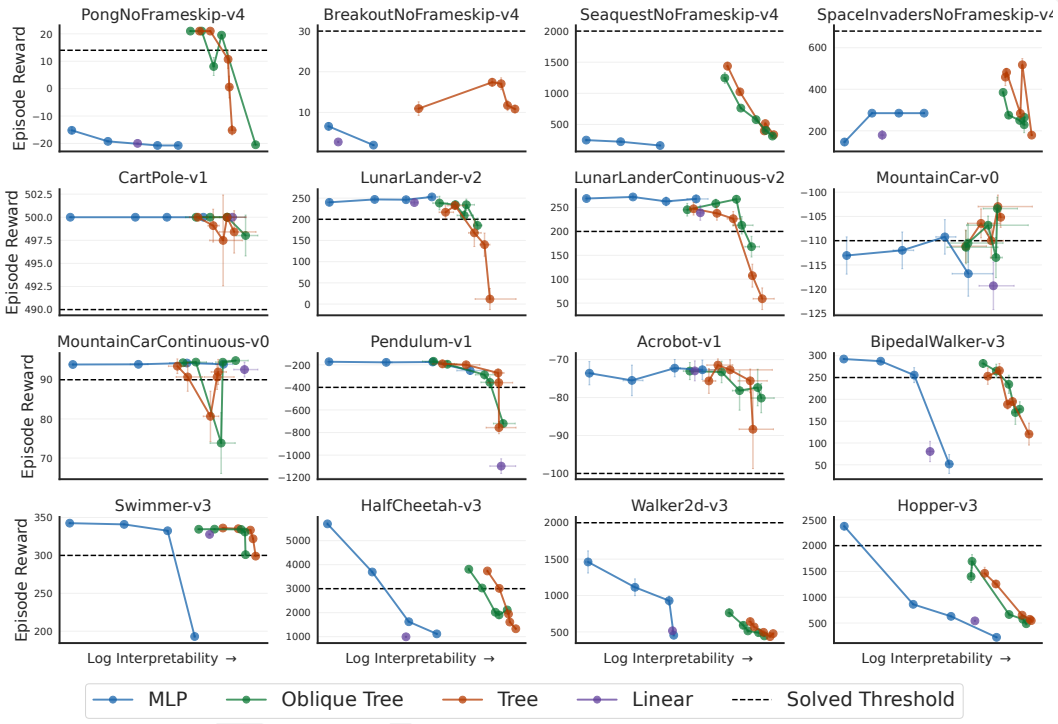


Figure C.3 – Trade-off Cumulative Reward vs. Episode Inference Time

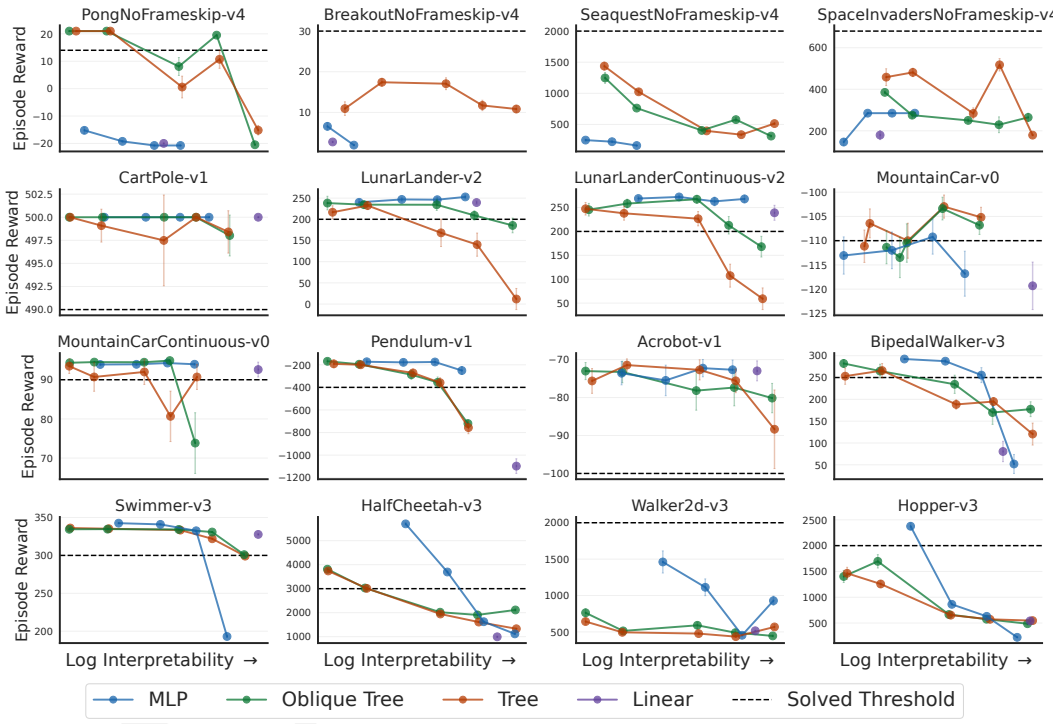


Figure C.4 – Trade-off Cumulative Reward vs. Policy Size