Université de Lille

Inria

Numéro d'ordre : 42

Université de Lille
Inria

École doctorale **École Graduée MADIS-631**
Unité de recherche **Centre de Recherche en Informatique, Signal et Automatique de Lille**

Thèse présentée par **Hector Kohler**

Soutenue le **1er décembre 2025**

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

Discipline **Informatique**
Spécialité **Informatique et Applications**

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

Thèse dirigée par	Philippe Preux	directeur
	Riad Akrour	co-directeur

Composition du jury

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

Colophon

Mémoire de thèse intitulé « Interpretabilité via l'Apprentissage Supervisé ou par Renforcement d'Arbres de Décisions », écrit par Hector Kohler, achevé le 22 juin 2025, composé au moyen du système de préparation de document LaTeX et de la classe yathesis dédiée aux thèses préparées en France.

Université de Lille

Inria

Université de Lille
Inria

École doctorale **École Graduée MADIS-631**

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

Thèse présentée par **Hector Kohler**

Soutenue le **1er décembre 2025**

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

Discipline **Informatique**

Spécialité **Informatique et Applications**

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

Thèse dirigée par	Philippe Preux	directeur
	Riad Akrour	co-directeur

Composition du jury

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

Université de Lille

Inria

Université de Lille
Inria

Doctoral School **École Graduée MADIS-631**

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

Thesis defended by **Hector Kohler**

Defended on **December 1, 2025**

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

Academic Field **Computer Science**

Speciality **Computer Science and Applications**

Interpretability through Supervised or Reinforcement Learning of Decision Trees

Thesis supervised by	Philippe Preux	Supervisor
	Riad Akrour	Co-Supervisor

Committee members

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

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

Résumé

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum. Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

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

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 through Supervised or Reinforcement Learning of Decision Trees

Abstract

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

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

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

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

Keywords: reinforcement learning, deicision trees, interpretability, methodology

Sommaire

Preliminary Concepts

Interpretable Sequential Decision Making

What is Sequential Decision Making?

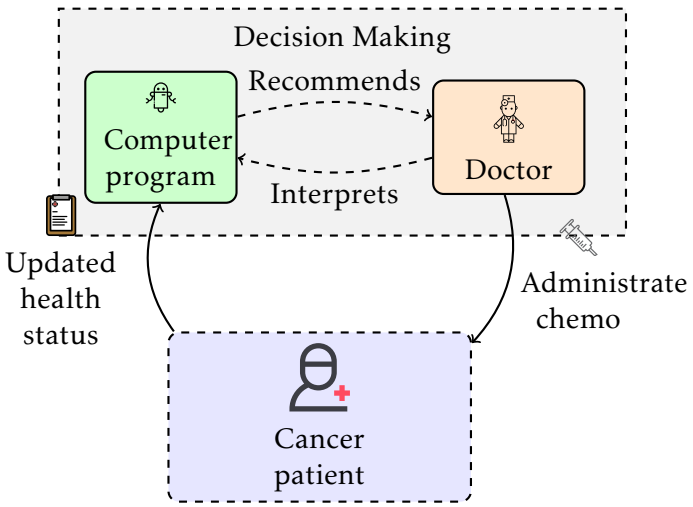


FIGURE 1 – Sequential decision making in cancer treatment. The AI system reacts to the patient's current state (tumor size, blood counts, etc.) and makes a recommendation to the doctor, who administers the chemotherapy to the patient. The patient's state is then updated, and this cycle repeats over time.

In this manuscript we study algorithms for sequential decision making. Humans engage in sequential decision making in all aspects of life. In medicine, doctors have to decide how much chemotherapy to administrate based on the patient's current health in order to cure [6]. In agriculture, agronomists have to decide when to fertilize next based on the current soil and weather conditions in

order to maximize plant growth [7]. In automotive, the auto-pilot system has to decide how to steer the wheel next based on lidar sensors in order to maintain a safe trajectory [11]. Those sequential decision making processes exhibits key similarities : an agent takes actions based on some current information to achieve some goal.

As computer scientists, we ought to design computer programs [9] that can help humans during those sequential decision making processes. For example, as depicted in Figure 1, a doctor could benefit from a program that would recommend the "best" treatment given the patient's state. Machine learning algorithms [22] 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 [12]. In the cancer treatment example, the doctor follows its patient through time and adapts its treatment to the changing health of the patient. In that case, machine learning, and in particular, *reinforcement* learning [19], can be used to teach the program how take decisions that lead to the patient recovery in the future based on how the patient's health changes from one chemo dose to another. Such machine learning algorithms train more and more performing program that make their way into our society to, e.g. identify digits on images [10], control tokamak fusion [4], or write the abstract of a scientific article [5].

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

What is Interpretability?

Originally, the etymology of "interpretability" is the Latin "interpretabilis" meaning "that can be understood and explained". According to the Oxford

English dictionary, the first recorded use of the english word "interpretability" dates back to 1854 when the british logician George Boole (Figure 2) described the addition of concepts :

> I would remark in the first place that the generality of a method in Logic must very much depend upon the generality of its elementary processes and laws. We have, for instance, in the previous sections of this work investigated, among other things, the laws of that logical process of addition which is symbolized by the sign +. Now those laws have been determined from the study of instances, in all of which it has been a necessary condition, that the classes or things added together in thought should be mutually exclusive. The expression x + y seems indeed uninterpretable, unless it be assumed that the things represented by x and the things represented by y are entirely separate ; that they embrace no individuals in common. And conditions analogous to this have been involved in those acts of conception from the study of which the laws of the other symbolical operations have been ascertained. The question then arises, whether it is necessary to restrict the application of these symbolical laws and processes by the same conditions of interpretability under which the knowledge of them was obtained. If such restriction is necessary, it is manifest that no such thing as a general method in Logic is possible. On the other hand, if such restriction is unnecessary, in what light are we to contemplate processes which appear to be uninterpretable in that sphere of thought which they are designed to aid ? [2, p. 48]

What is remarkable is that the supposedly first recorded occurrence of "interpretability" was in the context of computer science. Boole asked : *when can we meaningfully apply formal mathematical operations beyond the specific conditions under which we understand them ?* In Boole's era, the concern was whether logical operations like addition could be applied outside their original interpretable contexts—where symbols and their sum represent concepts that humans can understand, e.g. red + apples = red apples. Today, we face an analogous dilemma with machine learning algorithms : black-box programs like neural

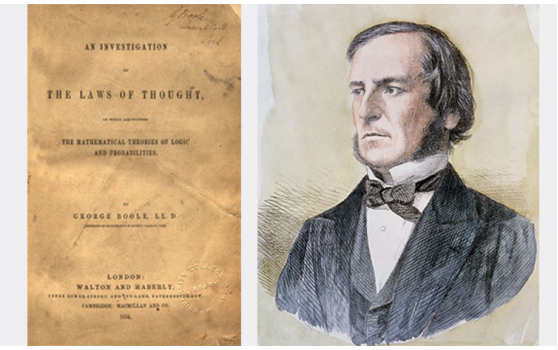
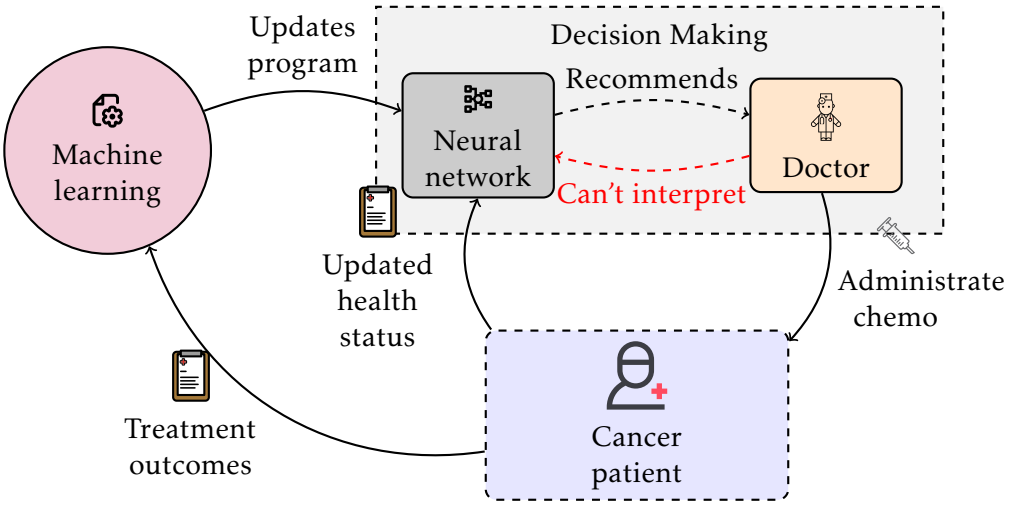


FIGURE 2 – British logician and philosopher George Boole (1815-1864) next to its book *The Laws of Thoughts* (1854) that is the oldest known record of the word "interpretability".

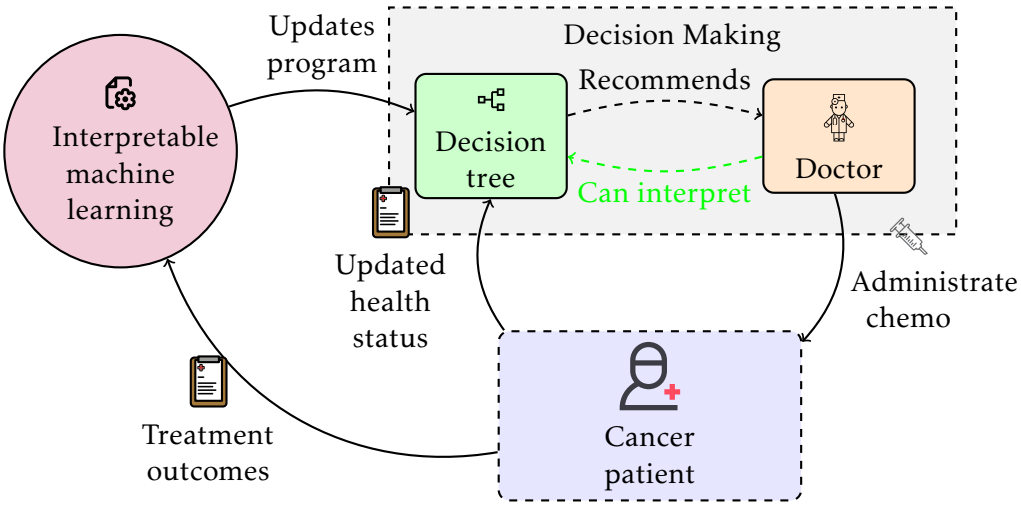
netowrks [16], that learn complex unintelligible combinations of inputs (representations), are often deployed in contexts where computations should be understood by humans, e.g., in medicine [18].

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 recommended such dosage ? In Figure 3b, we illustrate how machine learning *should* be used in practice. We would ideally want doctors to have access to computer programs that can recommend "good" treatments and which recommendations are interpretable.

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



(a) Black-box approach using neural networks



(b) Interpretable approach using decision trees

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

What are existing approaches for learning interpretable programs?

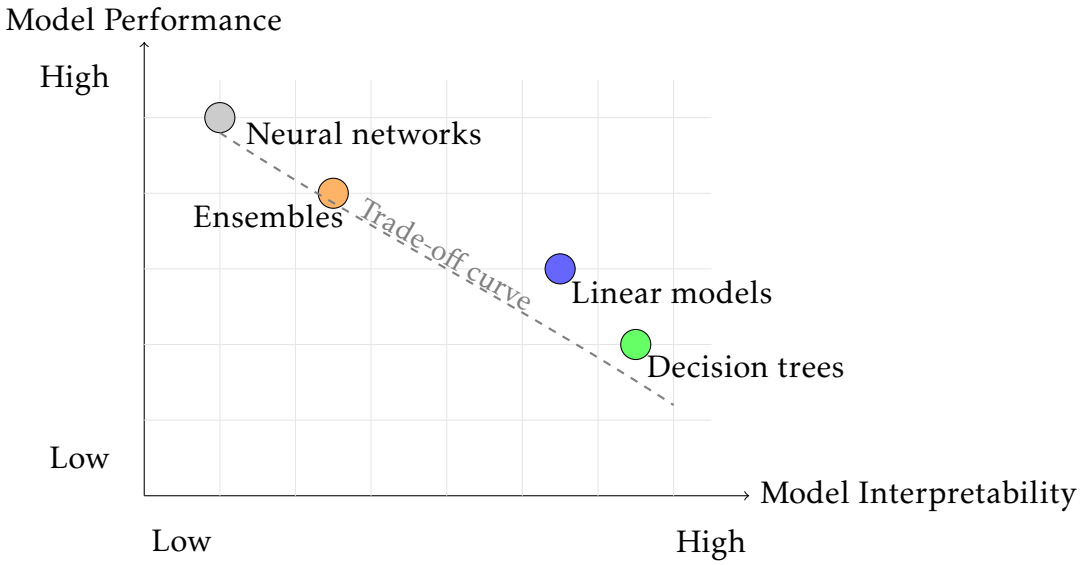


FIGURE 4 – The interpretability-performance trade-off in machine learning. Different program classes are positioned according to their typical interpretability and performance characteristics. The dashed line illustrates the general trade-off between these two properties.

Interpretable machine learning provides either local or global explanations [8]. Global methods output a program which all recommendations can be interpreted without additional computations, e.g. a decision tree [3]. On the other hand, local methods require additional computations but are agnostic to the program class : they can give an *approximate* interpretation of e.g. neural networks recommendations. In Figure 4 we present the popular trade-off between interpretability and performance of different program classes.

The most famous local explanation algorithm is LIME (Local Interpretable Model-agnostic Explanations) [15]. Given a program class, LIME works by perturbing the input and learning a simple interpretable model locally to explain that particular prediction. For each individual prediction, LIME provides explanations by identifying which features were most important for that specific decision. Hence, as stated above, LIME needs to learn a whole program per

recommendation that needs to be interpreted ; this is a lot of computations.

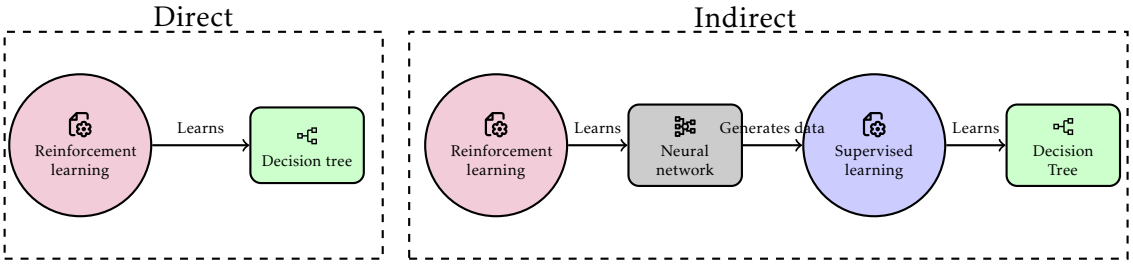


Figure 5 – Comparison of direct and indirect approaches for learning interpretable policies in sequential decision making

Global approaches are either direct or indirect [13]. Direct algorithms, such as decision tree induction [3], are algorithms that directly search a space of interpretable programs (see Figure 4). One of the key challenges that motivates this thesis is that decision tree induction is only defined for supervised learning but not for reinforcement learning. It means that to directly learn computer programs for interpretable sequential decision making, one has to design completely new algorithms. What most existing research have focused on so far, is to work around this confinement of decision tree induction to supervised learning and develop indirect methods. Indirect methods for interpretable sequential decision making–sometimes called post-hoc–start with the reinforcement learning of a non-interpretable computer program, e.g., deep reinforcement learning of a neural network, and then use supervised learning of an interpretable model with the objective to emulate the non-interpretable program. This approach is called behavior cloning or imitation learning [14, 17] and many, if not all, work on interpretable sequential decision making use this indirect approach[1, 23]. Figure 5 illustrates the key difference between these two approaches.

Researchers just recently started to study the advantage of direct over indirect learning of interpretable programs [20, 21]. In short, the motivation behind developing direct methods is to have the interpretable program optimized to solve your actual goal, e.g. patient treatment, while indirect methods learn an interpretable program that is optimized to match the behaviour of a non-interpretable model that was itself optimized to solve your goal. There is no guarantee that optimizing this indirect objective yields the “best” interpretability-performance

trade-offs. Hence, the ideal solution to interpretable sequential decision making would be to have global direct algorithms.

Next, we present the outline of this thesis.

Outline of the Thesis

In this thesis we take a stab at the difficult task of designing global direct algorithms for interpretable sequential decision making. In the first part of the manuscript we will present a mathematical formalism for the reinforcement learning of decision trees for sequential decision making. In particular we will show that indeed the direct approach can yield better trees than the indirect one. However the unfortunate key result from this opening part is that a good direct method cannot find decision trees for sequential decision making that are not *very* easy. Fortunately for us, in the second part of this manuscript, we show that some of these easy instances of interpretable sequential decision making tasks can be made *non*-sequential giving rise to a whole new decision tree induction algorithm for supervised learning. In particular, we thoroughly benchmark our new decision tree induction algorithm and claim the state-of-the-art for decision tree induction. Finally, after heavily studying decision trees and direct methods, we will leverage the diversity and simplicity of *indirect* methods to compare other model classes of programs and show that in some cases neural networks can be considered more interpretable than trees and there exist problems for which there is no need to trade-off performance for interpretability. We summarize the outline of the manuscript in Figure 6

Technical Preliminaries

What are decision trees?

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

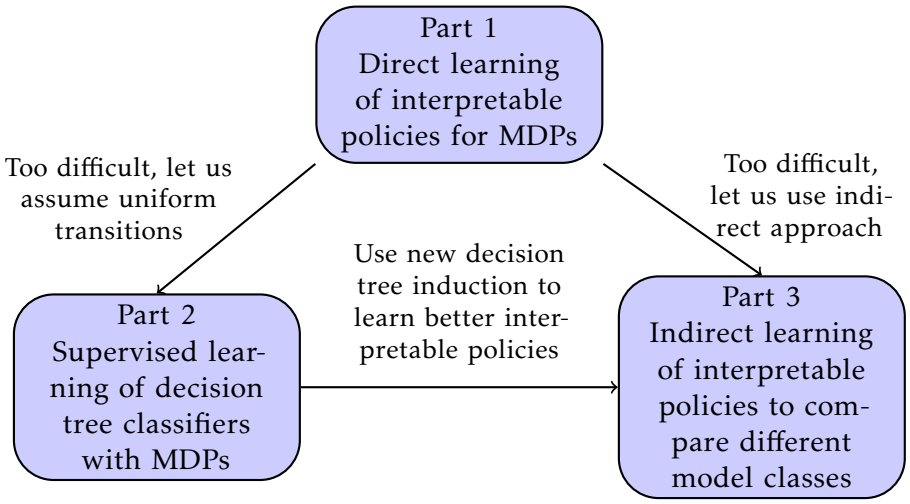


FIGURE 6 – Thesis structure showing the progression from direct reinforcement learning of decision tree policies (Chapter 1) to simplified approaches : supervised learning with uniform transitions (Chapter 2) and indirect learning methods (Chapter 3).

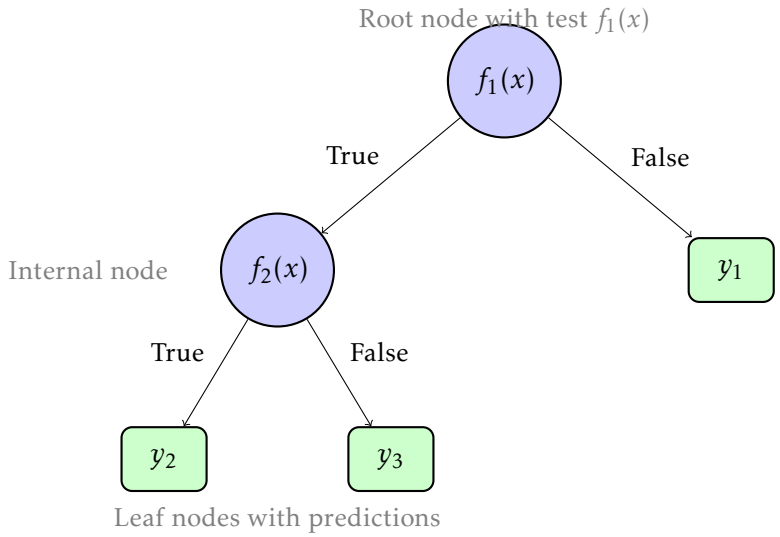


FIGURE 7 – A generic decision tree structure. Internal nodes contain test functions $f_v(x) : \mathcal{X} \to \{0, 1\}$ that map input features to boolean values. Edges represent the outcomes of these tests (True/False), and leaf nodes contain predictions $y_\ell \in \mathcal{Y}$. For any input x, the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

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

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

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

How to learn decision trees ?

The Classification and Regression Trees (CART) algorithm, developed by Leo Breiman and colleagues (Figure 8), is one of the most widely used methods for learning decision trees from supervised data. CART builds binary decision trees through a greedy, top-down approach that recursively partitions the feature space. At each internal node, the algorithm selects the feature and threshold that best splits the data according to a purity criterion such as the Gini impurity for classification or mean squared error for regression.

CART uses threshold-based test functions of the form $f_v(x) = \mathbb{I}[x[feature] \leq threshold]$ where $\mathbb{I}[\cdot]$ is the indicator function, consistent with the general de-

cision tree definition above. The key idea is to find splits that maximize the homogeneity of the resulting subsets. For classification, this means finding test functions that separate different classes as cleanly as possible. The algorithm continues splitting until a stopping criterion is met, such as reaching a minimum number of samples per leaf or achieving sufficient purity. The complete CART procedure is detailed in Algorithm 1.

Markov decision processes/problems

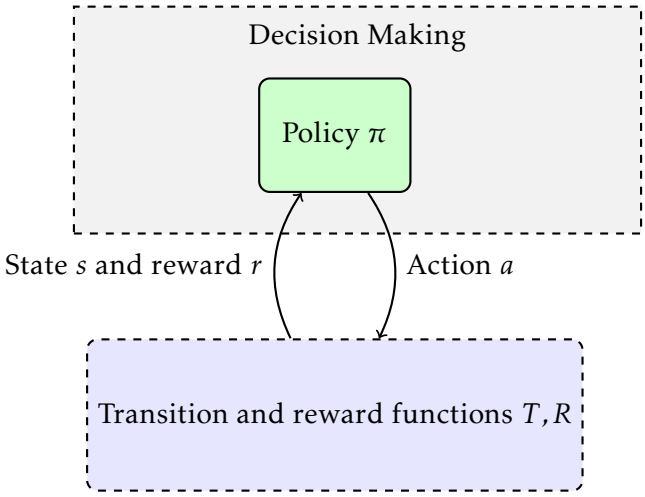


Figure 9 – Markov decision process

Markov decision processes (MDPs) were first introduced in the 1950s by Richard Bellman (cite). Informally, an MDP models how an agent acts over time to achieve its goal. At every timestep, the agent observes its current state, e.g. a patient weight and tumor size, and takes an action, e.g. injects a certain amount of chemotherapy. When doing a certain action in a certain state, the agent gets a reward that helps it evaluate the quality of its action with respect to its goal, e.g., the tumor size decrease when the agent has to cure cancer. Finally, the agent is provided with a new state, e.g. the updated patient state, and repeats this process over time. Following Martin L. Puterman's book on MDPs (cite), we formally define as follows.

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

Algorithme 1 : CART Algorithm for Decision Tree Learning

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

where :

- *S is a finite set of states $s \in \mathbb{R}^n$ representing all possible configurations of the environment.*
- *A is a finite set of actions $a \in \mathbb{Z}^m$ available to the agent.*
- *$R : S \times A \to \mathbb{R}$ is the reward function that assigns a real-valued reward to each state-action pair.*
- *$T : S \times A \to \Delta(S)$ is the transition function that maps state-action pairs to probability distributions over next states, where $\Delta(S)$ denotes the probability simplex over S.*
- *$T_0 \in \Delta(S)$ is the initial distribution over states.*

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

Definition 3 (Markov decision problem)**.** *Given an MDP $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$, the goal of an agent following policy $\pi : S \to A$ is to maximize the expected discounted sum of rewards :*

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

where $\gamma \in (0, 1)$ is the discount factor that controls the trade-off between immediate and future rewards.

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

Exact solutions for Markov decision problems

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

Definition 4 (Value of a state)**.** *The value of a state $s \in S$ under policy π is the expected discounted sum of rewards starting from state s and following policy π :*

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

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

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

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

Definition 5 (Optimal value of a state)**.** *The optimal value of a state $s \in S$, $V^\star(s)$, is the value of state s when following the optimal policy : $V^{\pi^\star}(s)$.*

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

Definition 6 (Optimal value of a state-action pair)**.** *The optimal value of a state-action pair $(s, a) \in S \times A$, $Q^\star(s, a)$, is the value of state s when taking action a and then following the optimal policy : $V^{\pi^\star}(s)$.*

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

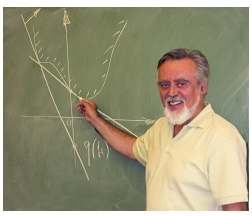
Hence, the algorithms we study in the thesis can also be seen as solving the problem : $\pi^\star = \underset{\pi}{\text{argmax}} \mathbb{E}[V^\pi(s_0) | s_0 \sim T_0]$. The well-known Value Iteration algorithm 2 solves this problem exactly (cite).

More realistically, neither the transition kernel T nor the reward function R of the MDP are known, e.g., the doctor can't **know** how the tumor and the patient

Algorithme 2 : Value Iteration

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

health will change after a dose of chemotherapy, it can only **observe** the change. This distinction between the information available to the agent is paralleled with the distinction between dynamic programming and reinforcement learning (RL) that we describe next.


(a) D. Bertsekas


(b) M.L. Puterman


(c) A. Barto


(d) R. Sutton

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

Reinforcement learning of approximate solutions to MDPs

Reinforcement learning algorithms popularized by Richard Sutton (Figure 10) (cite) don't **compute** an optimal policy but rather **learn** an approximate one based on sequences of observations $(s_t, a_t, r_t, s_{t+1})_t$. RL algorithms usually fall into two categories : value-based (cite) and policy gradient (cite). The first group of RL algorithms computes an approximation of $V^\star$ using temporal difference learning, while the second class leverages the policy gradient theorem to approximate $\pi^\star$. Examples of these approaches are shown in Algorithms 3 and 4.

Algorithme 3 : Value-based RL (Q-Learning)

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

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

Algorithme 4 : Policy Gradient RL (REINFORCE)

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

Première partie

A difficult problem : Learning Decision Trees for MDP

I have not failed. I've just found
10.000 ways that won't work.

Thomas A. Edison

Chapitre 1

A failing direct interpretable reinforcement learning algorithm

1.1 Iterative Bounding Markov Decision Processes

We start by considering a very basic Markov decision process (def) for which we would like decision tree (def) policies.

In 2021, Topin et. al. introduced Iterative Bounding Markov decision process (IBMPD) (cite). It is, to the best of our knowledge, the only work in which decision tree policies are *grown* using the RL object (def). We insist here on *growing* to refer to trees whose nodes and leaves structure is not knows *a priori* as opposed to parametric trees–trees which structure is fixed– and can be optimized with the policy gradient theorem (cite). In supervised learning a similar distinction exists too (cite).

The policy gradient theorem or random search can also be used in to learn linear policies (cite) that should be more interpretable than deep neural netwok. In this seminal work Topin et. al., formulate learning a decision tree policy for an MDP as solving an augmented MDP where some actions are adding decision nodes to a tree structure or taking base actions. (figure) More interestingly, IBMDP can generalize to any sort of tree-like policies where nodes need only to be binary functions of features.

1.2 Litterature and results on POMDPs

Without mentionning explicitely Topin et.al. propose to learning in IBMDPs by solving a Partially Observable MDP (cite). POMDPs are notorously more difficult to solve than MDPs (cite). In particular, not all policies for $\mathcal{M}_I B$ are decision tree policies for $\mathcal{M}$. Only policies from partial observation to actions are trees. Hence learning a decision tree policy for an MDP can be done by learning a *deterministic* and *reactive* policy for a POMDP. In general, this problem cannot be solved exactly with dynamic programming because of intrinsic POMDP limitations described in the work of Michael Littman (cite).

We will show that three of those limitations arise in direct interpretable RL :

1. The optimal policy in a POMDP can be stochastic
2. The optimal policy in a POMDP does not necessarily maximizes all state values simultaneously
3. Q-Learning like algorithms do not necessirily converge to the optimal Q-values.

In addition to those limitaions that are inherited from POMDPs, IBMDPs bring their own intrinsic challenges :

1. The action space might need to be state dependent
2. It is hard to align RL agents for IBMDPs

1.3 IBMDP and Didactic example

Given an MDP $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$, an Iterative Bounding Markov decision process is defined as follows.

Definition 7. *Given an MDP $\mathcal{M} = \langle S, A, R, T, T_0 \rangle$, an Iterative Bounding Markov decision process $\mathcal{M}_{IB}$ is a tuble $\langle S, O, A, A_{IG}, R, \zeta, T, T_0, P \rangle$. The IBMDP is an augmented version of an MDP. The states in an IBMPD are concatenations of $s \in S \subset [L_1, U_1] \times \cdots \times [L_n, U_n] \subsetneq \mathbb{R}^n$ and bounding values $o = (L_0) \in O$*

1.3.1 An MDP for which we want a decision tree policy

(figure) Do figure of POMDP graphical models with grid world. We consider a 2×2 grid world Markov Decision Process (MDP) defined as follows :

- **States** : Four cells labeled S_0, S_1, S_2, and G (goal state) arranged in a 2×2 grid.
- **Actions** : At each state, the agent can move right (→) or down (↓) up (↑) or left (←).
- **Transitions** : Movements are deterministic, following the direction of the chosen action. Actions that would lead outside the grid leave the agent in the same state.
- **Rewards** : All transitions yield a reward of 0, except for any action taken from the goal state G, which yields a reward of 1.
- **Objective** : Maximize the expected discounted cumulative reward.

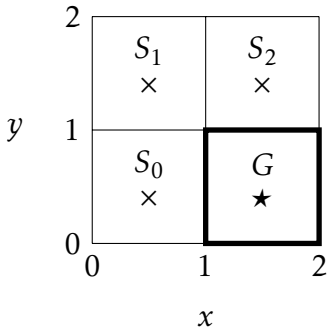


FIGURE 1.1 – The 2×2 grid world environment with states S_0, S_1, S_2, and goal state G.

In the next chapter we show the limitation of both direct and indirect approaches for this problem.

Chapitre 2

An attempt at Reinforcement Learning of Policies

2.1 Grid World tabular solutions

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

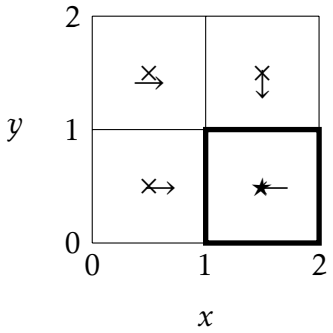


FIGURE 2.1 – An optimal tabular policy for the grid world.

For this grid world, there exist *optimal* decision tree policies. In particular

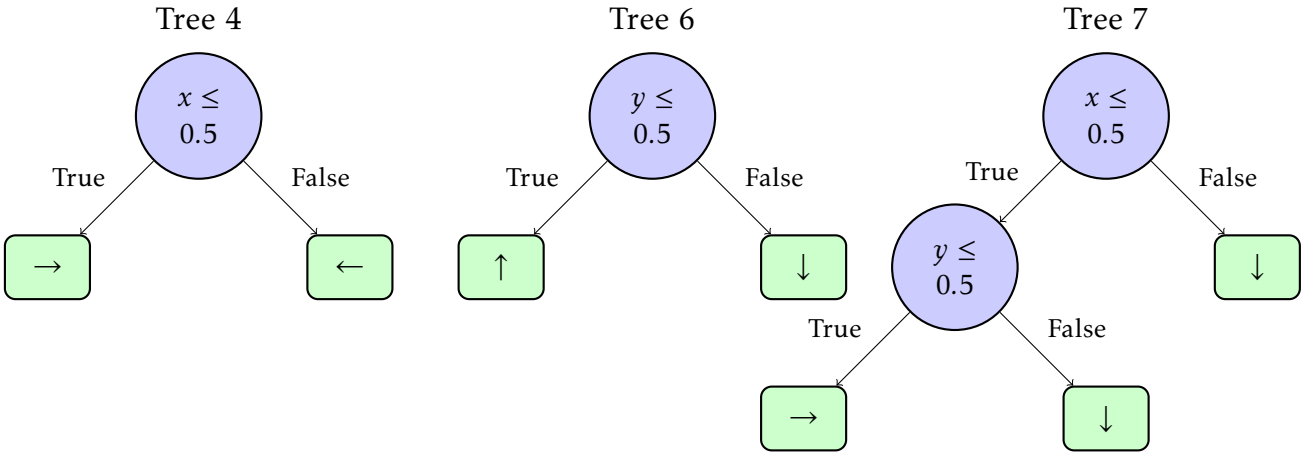


FIGURE 2.2 – Some optimal decision tree policies.

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

2.2 Method

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

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

$$\begin{aligned} V_g^{\mathcal{T}_0} &= 1 + \gamma + \gamma **2 \\ &= \frac{1}{1 - gamma} \end{aligned}$$

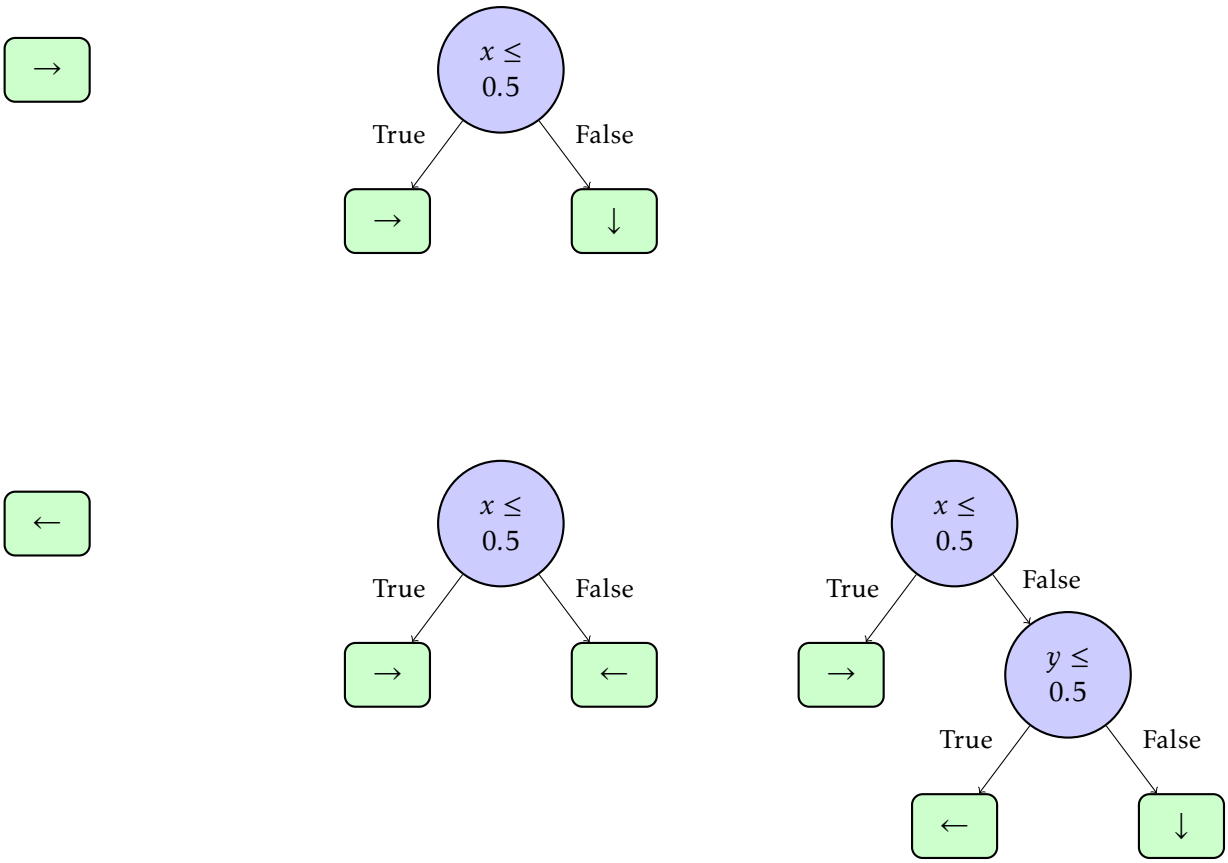


FIGURE 2.3 – Examples of decision trees for different policies in the grid world. Each tree represents a different policy, from simple single-action policies to more complex conditional policies.

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

$$V^{\mathcal{T}_0}_{(0,0)} = 0 + \gamma V^{\mathcal{T}_0}_{g} \qquad\qquad = \gamma V^{\mathcal{T}_0}_{g}$$

In the other states the policy never gets positive rewards. Hence :

$$J(\mathcal{T}_0) = \frac{1}{4} V^{\mathcal{T}_0}_{g} + \frac{1}{4} V^{\mathcal{T}_0}_{(0,0)}$$

2.3 Results

We first start by plotting the learning curves of three classical RL algorithms described in the preliminaries. There was a series of work in the 1990's(cite) that studied the performances of RL algos when naively applied to he partially observable setting, i.e, naively setting $s \leftarrow o$. The key observations are :

1. The learning finishes, i.e, learning curves flatten after some iterations.
2. There is a shift between the policies learned between the depth-0 tree and the depth>1 trees when $\zeta \geq 0$.

This is problematic because we know that *theoretically*, for $0 < \zeta < 1$, the optimal policy for the IBMDP objective had depth = 1.

Chapitre 3

Conclusion

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

Deuxième partie

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

Chapitre 4

DPDT-intro

Chapitre 5

DPDT-paper

Chapitre 6

Conclusion

Troisième partie

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

Conclusion générale

Bibliographie

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

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

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

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

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

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

[7] 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.

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

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

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

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

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

[13] 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.

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

[15] 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.

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

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

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

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

[20] 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.

[21] 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.

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

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

Annexe A

Programmes informatiques

Les listings suivants sont au cœur de notre travail.

Listing A.1 – Il est l'heure

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

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

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

```
25 ␣␣printf("\n");
26 }
```

Listing A.2 – Factorielle

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

Table des matières