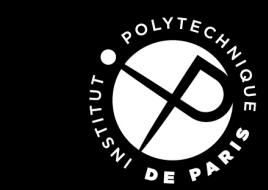


INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2024IPPAX109

Thèse de doctorat

Optimal Decision Trees via Search: A Reinforcement Learning framework

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Mathématiques et informatique

Thèse présentée et soutenue à Palaiseau, le 29 Novembre 2024, par

AYMAN CHAOUKI

Composition du Jury :

Nicolas Chopin Professeur, ENSAE	Président
Pierre Schaus Professeur, UCLouvain (ICTeam)	Rapporteur
Mathijs M. de Weerdt Professeur, TU Delft (Algorithmics Group)	Rapporteur
Emilie Kaufmann Chargée de recherche, Univ. Lille, Inria (CRIStAL)	Examinatrice
Jesse Read Professeur, École Polytechnique (ORAILIX)	Directeur de thèse
Albert Bifet Professeur, University of Waikato (Te Ipu o te Mahara AI Institute)	Co-directeur de thèse

Acknowledgements

I want to start by expressing my deepest gratitude to my thesis directors Jesse Read and Albert Bifet. These years have been very challenging intellectually and emotionally, but your continuous support, availability, research insight and confidence in my abilities helped me find inspiration throughout this journey. Thank you Jesse and Albert for the several insightful discussions we had. I always felt excited about presenting new ideas to you because, whenever I did, you always expressed great interest, excitement and encouragement. This was extremely valuable especially during moments of self doubt. I sincerely thank you for this.

I would like to thank the people I met at the AI Institute in the University of Waikato, and with whom I shared beautiful moments in New Zealand. I especially thank Guilherme Weigert Cassales, Nick Lim, Léa Cassé, Filippo, Heitor Murilo Gomes, Justin Liu, Paul Schlumbom, Nuwan Gunasekara, Yibin Sun and professors Bernhard Pfahringer and Eibe Frank.

I am extremely grateful to professor Emilie Kaufmann for welcoming me at the Scool team in Inria Lille. Emilie is the embodiment of mathematical rigour and I have learned a lot from her. Thank you Emilie for taking the time to analyse some of my theoretical work, your insight and caution about the use of conditional probabilities and your advice to employ union bounds instead whenever possible has affected my subsequent work substantially.

Thanks to professor Mathijs de Weerdt and professor Pierre Schaus for reviewing my thesis. Your feedback has been very valuable and helped me improve my work significantly especially from the empirical side. Special thanks to professor Nicolas Chopin for being president of the committee, for encouraging me and for being the first one to call me Doctor. I also thank you Nicolas for welcoming me in your class on Sequential Monte Carlo in the first year, a class that I found extremely interesting, well structured and rigorous.

Lastly, I cannot express the limits of my gratitude to my parents and my sister for their unwavering support and unconditional love. They helped me tremendously during difficult times and gave me enough strength to push forward, thank you infinitely. I also extend a huge thanks to my friend Otmane Sakhi for the numerous intellectual discussions that go beyond our thesis subjects and for all the laughter we shared. Last but certainly not least, a truly deep thank you to my partner Alice for her immense and continuous support, it was extremely helpful, especially at the end of my PhD, thank you from the bottom of my heart.

Abstract

Abstract. Decision Trees are one of the most popular models in Machine Learning because of their interpretability, which is especially noticeable for Decision Trees with few splits (or decision rules), a property that is called sparsity. Interpretable models are highly valued in domains where decisions carry substantial ramifications such as health-care and the criminal justice system. For this reason, seeking optimal sparse Decision Trees constitutes a fundamental problem in Interpretable Machine Learning, and a large research effort has been devoted to solving this problem.

Due to the NP-Hard difficulty of the problem, greedy heuristic methods such as C4.5 and CART have been favoured historically, and while fast and scalable, these methods remain unsatisfactory because they forgo the sparsity portion of the problem. In fact, they often lead to suboptimal and overly complex Decision Trees. Nevertheless, many algorithms have been developed in the literature to address the sparsity problem, with recent breakthroughs leveraging Dynamic Programming (DP) and Branch & Bound (B&B) techniques, and causing a paradigm shift from traditional Mathematical Programming to search algorithms. However, most of the proposed DP and B&B methods follow a Depth-First-Search (DFS) strategy, which, while attractive due to its storage-economy capacity, is inefficient because of its uninformative nature. Moreover, DFS necessitates the definition of a maximum depth hyperparameter a priori, and its inefficiency becomes a limiting factor when considering large maximum depths. These concerns were partially addressed with Best-First-Search (BFS) strategies. BFS is appealing because it prunes the search space more aggressively without losing the optimality guarantee, thus enabling it to find the optimal solution faster than DFS. Moreover, BFS does not necessitate fixing a maximum depth hyperparameter, it can run on infinite depth, which is an important advantage that allows practitioners to focus tuning efforts on other hyperparameters. However, these advantages come at the expense of higher memory consumption than DFS. To alleviate this issue, the search strategies of BFS methods have to be formulated as efficiently as possible, in other words, they need to find the optimal solution as quickly as possible. The reason being to tip the balance of their advantages-drawbacks more towards their advantages, outweighing their high memory consumption issue.

The current BFS methods can be improved substantially by reconsidering the sparsity problem under a framework that benefits from its structural properties. Our main contribution in this thesis addresses this concern within a Reinforcement Learning framework. Our BFS algorithm, Branches, benefits from an AO*-type search strategy, which, coupled with our pruning bound we call *Purification Bound* and several other heuristic considerations, focuses the search on relevant regions of the search space while maintaining the optimality guarantee. This enables Branches to find optimal sparse Decision Trees faster than the state of the art, and as a result

alleviate the memory burden that weighs on BFS approaches. We analyse the computational complexity of Branches theoretically by deriving an upper bound on the number of branch evaluations it performs before terminating, and we show this result to be superior to similar analyses from the literature. These findings are further validated through a series of extensive experiments showing Branches to significantly advance the state of the art in terms of number of iterations, runtime, anytime behaviour and scalability potential. This work lays the foundations for future work that can further improve the scalability of these methods.

In addition to this work, we also study the online setting where a data stream is observed rather than a fixed dataset. Unlike the batch setting, the literature on optimal sparse Decision Trees in is scarce for Online Learning. Most approaches adapt greedy heuristics to handling data streams through statistical estimates, hence inheriting their suboptimality issues. On this front, we develop three Monte Carlo Tree Search (MCTS) algorithms with asymptotic optimality guarantees, and we show that they can successfully retrieve the optimal solution in situations where the state of the art methods are practically guaranteed to fail. Furthermore, we adapt Branches to the online setting inducing Online-Branches, an algorithm that satisfies finite-time optimality guarantees rather than asymptotic ones. Our experiments show Online-Branches to outperform our previously developed MCTS methods. Our work on the online setting has the potential to draw more research attention to the problem of seeking optimal sparse Decision Trees when given a data stream. It also lays a solid foundation for these future works through several discussions about the potential issues that arise due to the hypotheses of Online Learning and our proposed solutions.

Résumé. Les arbres de décision figurent parmi les modèles les plus prisés en apprentissage automatique en raison de leur interprétabilité, une caractéristique particulièrement satisfaite par les arbres de décision comportant peu de divisions (ou règles de décision), cette propriété est connue sous le nom de parcimonie. Les modèles interprétables sont très valorisés dans des domaines où les décisions ont des répercussions significatives, tels que le domaine de la santé et le système judiciaire. C'est pour cette raison que la recherche d'arbres de décision optimaux et parcimonieux constitue un problème fondamental en apprentissage automatique interprétable, suscitant un effort de recherche considérable.

En raison de la complexité NP-Hard du problème, les méthodes heuristiques gloutonnes telles que C4.5 et CART ont historiquement été privilégiées. Malgré leur rapidité, ces méthodes restent insatisfaisantes, car elles négligent l'aspect parcimonieux du problème. En effet, elles conduisent souvent à des arbres de décision sous-optimaux et inutilement complexes. Néanmoins, de nombreux algorithmes ont été développés dans la littérature pour pallier ce problème, avec des avancées récentes exploitant la programmation dynamique (DP) et les techniques de Branch & Bound (B&B), provoquant ainsi un changement de paradigme de la programmation mathématique traditionnelle aux algorithmes de recherche. Cependant, la plupart des méthodes DP et B&B proposées suivent une stratégie Depth-First Search (DFS) qui, bien qu'ayant le mérite d'une faible consommation de mémoire, se révèle inefficace en raison de son manque d'information. De plus, DFS requiert la définition a priori d'un hyperparamètre de profondeur maximale, et son inefficacité devient un facteur limitant pour les grandes valeurs de cet hyperparamètre. Ces préoccupations ont été partiellement adressées par les stratégies dites Best-First Search (BFS). BFS est particulièrement attrayant, car il réduit plus efficacement l'espace de recherche sans compromettre la garantie d'optimalité, lui permettant ainsi de trouver la solution optimale plus rapidement que DFS. De plus, BFS ne nécessite nullement un hyperparamtètre de profondeur maximale, et peut ainsi fonctionner à profondeur infinie. Ceci représente un

avantage crucial qui permet aux praticiens de concentrer leurs efforts d'ajustement sur d'autres hyperparamètres. Cependant, ces avantages s'accompagnent d'une consommation de mémoire plus élevée que DFS. Pour atténuer ce problème, les stratégies de recherche des méthodes BFS doivent être formulées de manière aussi efficace que possible, autrement dit, elles doivent trouver la solution optimale aussi rapidement que possible. L'objectif est d'équilibrer davantage leurs avantages et inconvénients, afin que leurs atouts l'emportent sur leur consommation élevée de mémoire.

Les méthodes BFS actuelles peuvent être considérablement améliorées en réexaminant le problème de la parcimonie sous un cadre exploitant ses propriétés structurelles. La principale contribution de cette thèse étudie cette question dans un cadre d'apprentissage par renforcement. Notre algorithme BFS, Branches, s'appuie sur une stratégie de recherche de type AO*, qui, associée à une borne d'élagage que nous appelons Purification Bound ainsi qu'à plusieurs autres heuristiques, oriente la recherche vers les régions pertinentes de l'espace de recherche tout en conservant la garantie d'optimalité. Cela permet à Branches de trouver des arbres de décision optimaux et parcimonieux plus rapidement que l'état de l'art, tout en atténuant la contrainte mémoire pesant sur les approches BFS. Nous analysons la complexité computationnelle de Branches d'un point de vue théorique en établissant une borne supérieure sur le nombre d'évaluations de branches effectuées avant la terminaison, et nous démontrons la supériorité de ce résultat pa rapport à des analyses antécédentes du même genre. Ces conclusions sont ensuite validées par une série d'expériences approfondies démontrant que Branches améliore significativement l'état de l'art en termes de nombre d'itérations, de temps d'exécution, de comportement anytime et de potentiel de passage à l'échelle. Ce travail pose ainsi les bases de recherches futures visant à améliorer encore plus le passage à échelle de ces méthodes.

Nous étudions également le cadre en ligne, où un flux de données est observé plutôt qu'un ensemble fixe de données. Contrairement au cadre batch, la littérature sur les arbres de décision optimaux et parcimonieux est encore peu développée en apprentissage en ligne. La plupart des approches adaptent des heuristiques gloutonnes pour traiter les flux de données via des estimations statistiques, héritant ainsi de leurs limites en termes de sous-optimalité. Sur ce front, nous développons trois algorithmes de Monte Carlo Tree Search (MCTS) bénéficiant de garanties d'optimalité asymptotiques et nous montrons qu'ils parviennent à retrouver la solution optimale dans des situations où les méthodes actuelles échouent quasi systématiquement. Par la suite, nous adaptons Branches au cadre en ligne, à travares l'algorithme Online-Branches. Cet algorithme satisfait des garanties d'optimalité en temps fini plutôt que des résultats asymptotiques. Nos expériences montrent qu'Online-Branches surpasse nos méthodes MCTS précédentes. Notre travail sur le cadre en ligne a le potentiel d'attirer davantage l'attention de la communauté scientifique sur le problème de la recherche d'arbres de décision optimaux et parcimonieux en présence de flux de données. Il jette également des bases solides pour les recherches futures en proposant plusieurs discussions sur les défis liés aux hypothèses de l'apprentissage en ligne ainsi que nos solutions suggérées.

Contents

1 Introduction **8**
- 1.1 Overview of Decision Tree models . . . 8
- 1.2 Limitations of the greedy approaches . . . 10
- 1.3 The case for Data Stream Mining . . . 12
- 1.4 Reinforcement Learning (RL) . . . 13
- 1.5 Motivation and Objectives . . . 14
- 1.6 Outline of the Thesis . . . 15

2 Background and Related Work **18**
- 2.1 Classification Problem . . . 18
- 2.2 Greedy Decision Trees . . . 19
- 2.3 Optimal Decision Trees . . . 21
- 2.4 Online Decision Trees . . . 35
- 2.5 Reinforcement Learning (RL) . . . 40

I Optimal Batch Decision Trees **47**

3 Seeking Optimal Sparse Decision Trees in the Space of Decision Trees **48**
- 3.1 Introduction . . . 49
- 3.2 Preliminaries . . . 49
- 3.3 The Algorithm: DT-Miner . . . 53
- 3.4 Experiments . . . 56
- 3.5 Conclusion . . . 58
- 3.6 Appendix: Table of Notation . . . 59
- 3.7 Appendix: Proofs . . . 60

4 Mining Optimal Sparse Decision Trees from the Space of Branches **64**
- 4.1 Introduction . . . 65
- 4.2 Preliminaries . . . 66
- 4.3 The Algorithm: Branches . . . 69
- 4.4 Theoretical Analysis . . . 74
- 4.5 Implementation Details . . . 76
- 4.6 Branches vs GOSDT . . . 80
- 4.7 Experiments . . . 82
- 4.8 Drawbacks of Binary (One-Hot) Encoding . . . 94
- 4.9 Conclusion . . . 98

4.10 Appendix: Table of Notation . . . 101
4.11 Appendix: Proofs . . . 102
4.12 Appendix: Auxiliary Procedures . . . 113
4.13 Appendix: Experiments on depth analysis . . . 115

II Optimal Online Decision Trees 127

5 Online Learning of Decision Trees with UCB and Epsilon-Greedy 128
5.1 Introduction . . . 129
5.2 Preliminaries . . . 130
5.3 The Algorithms: UCDT and EGDT . . . 134
5.4 Sample Efficiency and Weights Degeneracy . . . 137
5.5 Experiments . . . 140
5.6 Conclusion . . . 142
5.7 Appendix: Proofs . . . 146
5.8 Appendix: Pseudo-code . . . 151

6 Online Learning of Decision Trees with Thompson Sampling 152
6.1 Introduction . . . 153
6.2 The Algorithm: TSDT . . . 153
6.3 Experiments . . . 160
6.4 Conclusion . . . 161
6.5 Appendix: Table of Notation . . . 164
6.6 Appendix: Proofs . . . 165
6.7 Appendix: Pseudo-code . . . 173

7 Online Decision Trees with Finite-Time Optimality guarantees 174
7.1 Introduction . . . 175
7.2 Preliminaries . . . 175
7.3 The Algorithm: Online-Branches . . . 176
7.4 Theoretical Analysis . . . 177
7.5 Experiments . . . 180
7.6 Conclusion . . . 182
7.7 Appendix: Proofs . . . 188

8 Conclusion 198

Chapter 1

Introduction

The abundance of data in the modern world constitutes a valuable resource from which crucial insights can be inferred, from genomics and drug discovery to image recognition and automated locomotion tasks and much more. This propelled Machine Learning to the forefront of key modern disciplines as its models can extract important patterns in an automated fashion. Indeed, Machine Learning models are applied in a multitude of domains, from classification and regression tasks, to clustering, and sequential decision making to name a few. Perhaps, the most notorious sub-field of Machine Learning is Deep Learning (Goodfellow et al., 2016; LeCun et al., 2015). With the advancements of hardware, Neural Network models became powerful and revolutionised the world of Learning due to their unparalleled capacity of learning very complex functions from unstructured data. Among their applications we find accurate classification of digitized images and speech, as well as attaining super-human level performance in strategy games with extremely large search spaces such as Chess, Go and Atari games (Silver et al., 2016, 2017b,a; Mnih et al., 2013, 2015).

The successful Deep Learning models are not to be employed carelessly in every domain. Despite their outstanding performances, they present the caveat of being black-box models. This property can lead to unpredictable behaviour adding a layer of uncontrolled uncertainty to the performance of these models. Such intriguing behaviour has been analysed extensively in a multitude of papers, among which (Szegedy et al., 2013; Morris et al., 2020; Huang et al., 2017). In domains where wrong decisions have mild consequences, this ought not to be an issue, however, it becomes a decisive concern in decision-sensitive domains. In healthcare for instance, misdiagnoses can delay crucial treatments and lead to severe outcomes for patients. Likewise, in the criminal justice system, black-box models can obscure biases related to features such as race and gender potentially resulting in discriminatory rulings. Such risks highlight the necessity of adopting interpretable models instead of black-box models in sensitive domains. The most prolific interpretable models are Decision Trees, and they will be our models of interest throughout this thesis as we will develop methods for seeking optimal interpretable Decision Tree classifiers.

1.1 Overview of Decision Tree models

The most prominent models of Interpretable Machine Learning (Molnar, 2020; Rudin et al., 2022) are Decision Trees (DTs for short); these are flowchart-like models consisting of a series of simple rules that end up with a decision such as classifying an input datum described by a set of features. For example, suppose that we have an input datum $X = \left(X^{(1)}, X^{(2)}\right) = (1, 4)$

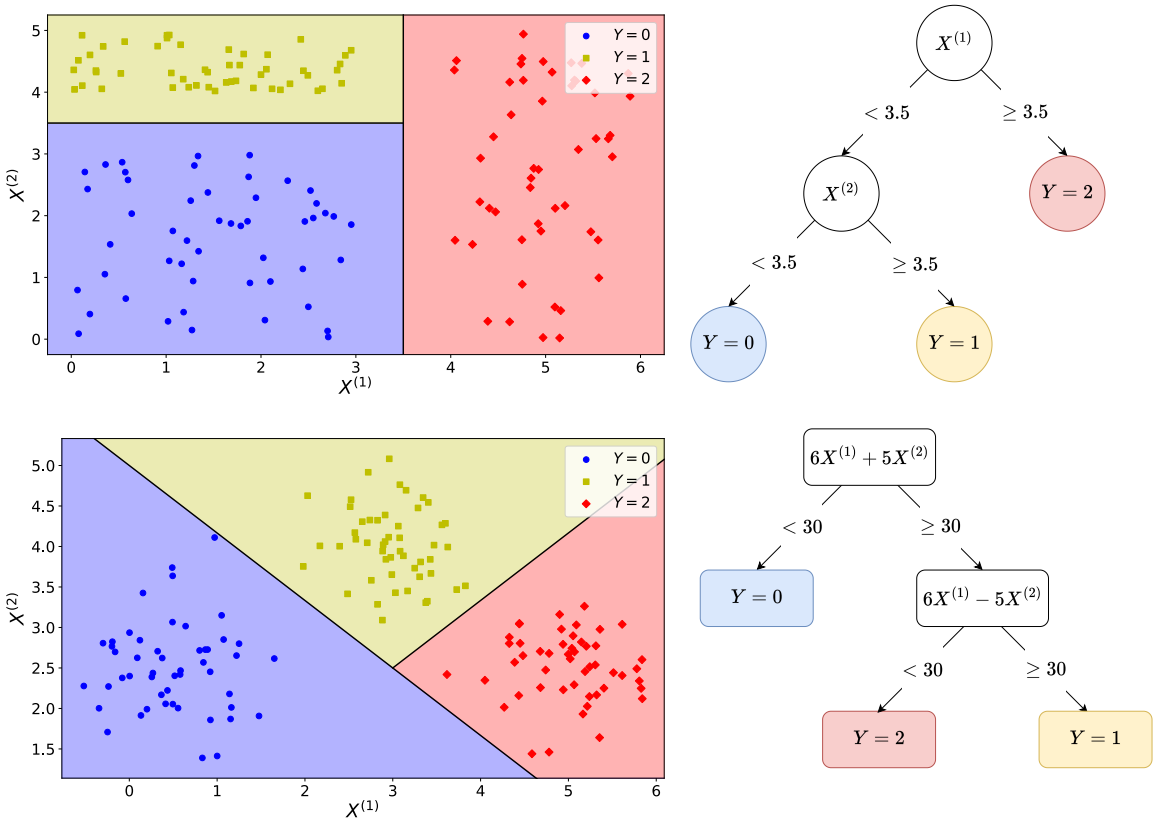


Figure 1.1: Classification problem with 3 classes on the input space $\mathcal{X} = \mathbb{R}^2$. The figure at the top depicts an orthogonal (univariate) Decision Tree classifier, the name orthogonal is due to the hyperplanes of the rules being orthogonal to the features axes. The bottom figure depicts an oblique (multivariate) Decision Tree classifier, the name oblique is due to the oblique nature of the hyperplanes of the rules since they linearly involve multiple features.

that we would like to classify according to the DT in the top-right corner of Fig. 1.1. The first rule tests whether $X^{(1)} < 3.5$ or $X^{(1)} \geq 3.5$; since $X^{(1)} = 1 < 3.5$, then we move to the second rule and test whether $X^{(2)} < 3.5$ or $X^{(2)} \geq 3.5$. Since $X^{(2)} = 4 \geq 3.5$, then the DT classifies X with $Y = 1$. The example we described is that of a DT over the input space $\mathbb{R}^2$, but DTs can be generalised to any number of dimensions $n \in \mathbb{N}$. These simple rules are what make DTs interpretable, as long as they are few in number.

There are two notable types of DTs, univariate (or orthogonal) trees, where each split (rule) involves only one feature, and multivariate (or oblique) trees, where multiple features are considered in each split. Fig. 1.1 illustrates both types of Decision Trees. While oblique DTs may provide a better classification performance, they also come at the cost of being less interpretable than the orthogonal DTs. For this reason, this thesis is solely focused on univariate Decision Trees. Unfortunately, as pointed out by Laurent and Rivest (1976), finding the optimal DT, with respect to some metric like accuracy, while simultaneously minimising its complexity (depth or number of splits) is NP-complete. This difficulty prompted the early approaches to greedily construct DTs in a top-down fashion, starting from the root, by choosing splits that optimise a local gain metric, generally based on the Gini impurity or Entropy, until the leaves satisfy one of the following termination criteria:

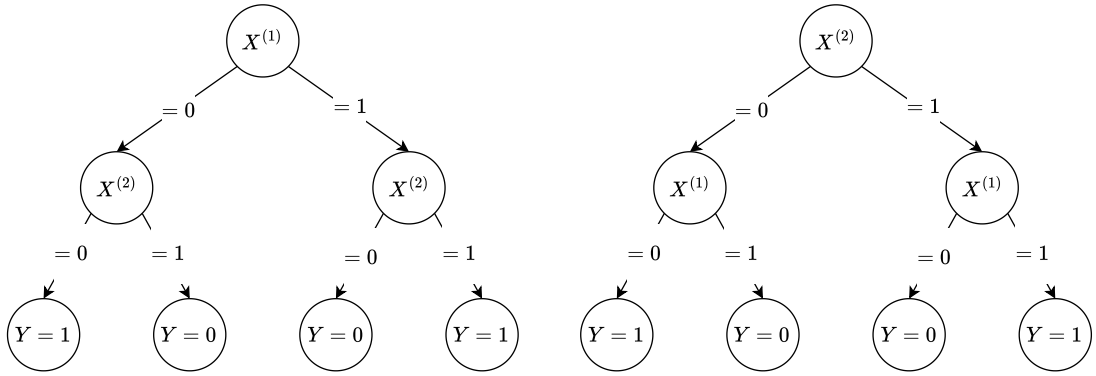


Figure 1.2: The two equivalent optimal sparse DTs representing $Y = \neg\left(X^{(1)} \oplus X^{(2)}\right)$.

- Containing less data inputs than a minimum allowed threshold.
- No split induces a positive gain in impurity.

This principled approach led to the development of the widely popular algorithms ID3 (Quinlan, 1986), C4.5 (Quinlan, 2014) and CART (Breiman et al., 1984).

1.2 Limitations of the greedy approaches

Despite the remarkable popularity of C4.5 and CART, the greedy nature of these algorithms offers no guarantee of global optimality. In fact, they often lead to overly complex DTs with poor out-of-sample accuracy. Hence why these methods are very often used within more powerful ensemble models like Random Forests (Breiman, 2001) and Gradient Boosting Machines (Friedman, 2001), but such ensemble approaches forgo interpretability.

The source of this suboptimality is deemed to be the greedy sequential construction of DTs, this is because local splits are myopic to their overall impact on the performance of the DT. Indeed, as pointed out by Bertsimas and Dunn (2017), it could happen that strong (desirable) splits are hidden behind weak splits in earlier stages of the DT construction. To understand this phenomenon, let us consider a simple binary classification problem with q binary i.i.d. uniform features $X^{(1)}, \ldots, X^{(q)}$ and predicted class $Y \in \{0, 1\}$ such that $Y = 1$ if and only if $X^{(1)} = X^{(2)}$[1]. $X^{(1)}$ and $X^{(2)}$ are the only relevant features, therefore, any DT that includes splits with respect to any other feature is suboptimal in the sense that it would include an irrelevant split. Fig. 1.2 illustrates the two possible optimal sparse DTs (sparse here refers to the inclusion of relevant splits only). The marginal distribution of Y is a uniform Bernoulli, and since Y is independent of features $X^{(3)}, \ldots, X^{(q)}$, splitting the root with respect to any of these irrelevant features yields leaves where Y is still (conditionally) distributed as a uniform Bernoulli. On the other hand, despite being relevant, splitting the root with respect to either $X^{(1)}$ or $X^{(2)}$ still yields a uniform Bernoulli distribution of Y at the level of the leaves, as evident by Fig. 1.3. Therefore, all splits at the root yield the same gain measure regardless of the employed impurity metric. As a consequence, a greedy algorithm such as CART either never splits the root, or arbitrarily chooses a feature for the split, which means that CART is very unlikely to retrieve an optimal sparse DT since it will only succeed if, by chance, it splits the root with respect to $X^{(1)}$ or $X^{(2)}$. The probability of retrieving an optimal DT in this case is therefore

[1] This is equivalent to the negated XOR function $Y = \neg\left(X^{(1)} \oplus X^{(2)}\right)$.

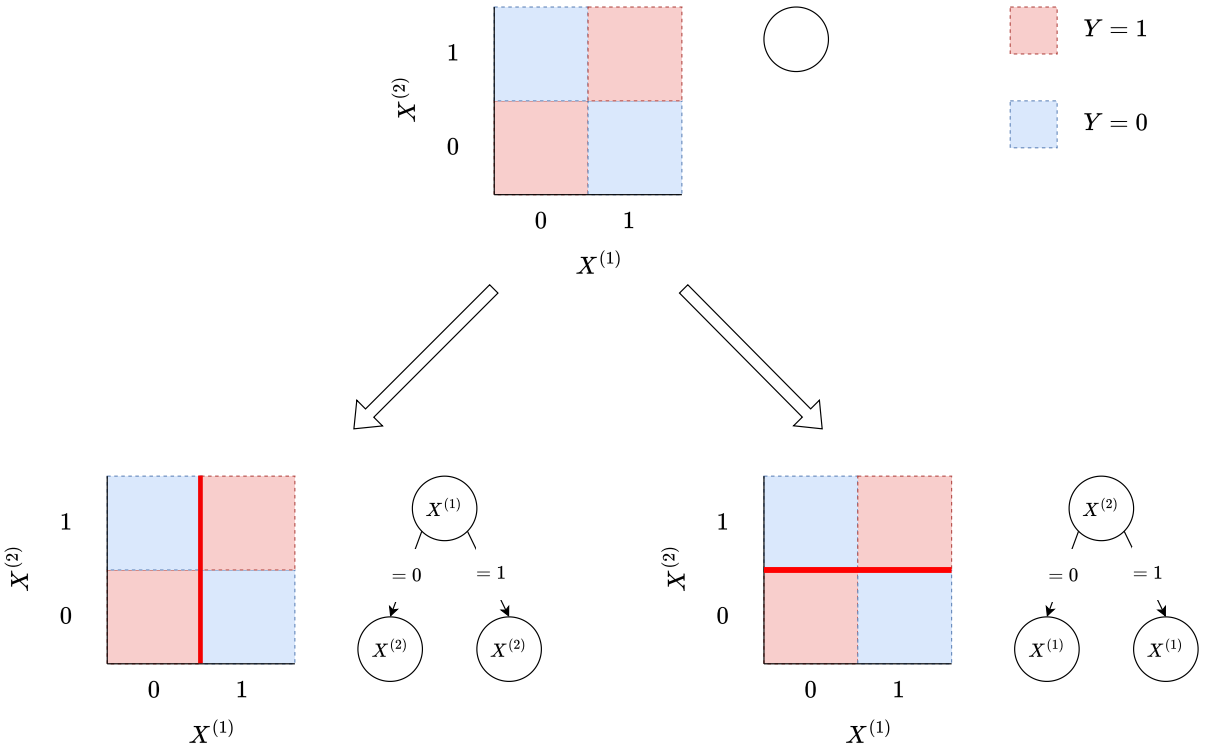


Figure 1.3: Even when splitting the root with respect to one of the relevant features $X^{(1)}$ and $X^{(2)}$, the induced (conditional) distribution of Y at the level of each leaf is still a uniform Bernoulli.

$\frac{2}{q+1}$ ($q+1$ is the number of possible splits plus the no splitting option), it decreases linearly with q.

The suboptimality issue of greedy DTs has been the subject of a plethora of research papers. To address the issue of myopic splits, some approaches relied on look-ahead heuristics such as IDX (Norton, 1989), LSID3 and ID3-k (Esmeir and Markovitch, 2007). However, Murthy and Salzberg (1995) empirically showed that the simple greedy methods consistently produce equally performing DTs to the more computationally expensive look-ahead heuristics. Nevertheless, it still seems that the natural way of addressing the myopic splits issue is through designing methods that construct the whole DT in one step, as this would allow the optimisation of all splits concurrently. This observation motivated many researchers to adopt a Mathematical Programming approach, with early approaches focusing on continuous optimisation. Bennett (1994); Bennett and Blue (1996) formulate a Multi-Linear program with a non-linear and non-convex objective function to optimise over a polyhedral region. Norouzi et al. (2015) derive a smooth convex-concave upper bound on the ill-behaved empirical loss, this upper bound is considered as a surrogate objective that is amenable to minimisation via Stochastic Gradient Descent (SGD). In a more recent attempt, Blanquero et al. (2021) introduce soft (randomized) decision rules at the level of the internal nodes, and formulate another Non-Linear Continuous Optimisation problem in which the objective is to minimise the expected misclassification cost. A different line of research emerged recently with Bertsimas and Dunn (2017), who observed that the discrete nature of the problem underlying the construction of optimal DTs is best described within a Mixed-Integer Optimisation (MIO) framework, and that the aversion to such formulation is due to the belief that MIO problems are intractable, even for small to medium sized instances. Bertsimas and Dunn (2017) argue that such belief may no longer be justified due

to the stellar progress of both modern MIO solvers and hardware in the recent decades. The authors then propose Optimal Classification Trees (OCT), an algorithm for retrieving the optimal DT based on a MIO formulation. On the same vein, Verwer and Zhang (2017, 2019) propose DTIP and BinOCT in order to decrease the number of variables involved in the program for scalability purposes. Other works also considered Satisfiability (SAT) and Constrained Programming (CP) (Bessiere et al., 2009; Narodytska et al., 2018; Avellaneda, 2020; Hu et al., 2020; Verhaeghe et al., 2020; Boutilier et al., 2022; Aghaei et al., 2024). These Mathematical Programming methods suffer from a serious issue of poor scalability. This is due to the dependence of the number of involved variables in the Mathematical Program on the training data size, the number of features, and the depth of the considered DTs. Furthermore, and even more importantly in our opinion, these techniques often fix the topology (structure) of the DT and only optimise its internal splits with no regard for sparsity.

To alleviate these issues, a new line of research emerged in the last five years exploring Dynamic Programming (DP) (Bellman and Dreyfus, 2015) and Branch & Bound (B&B) (Lawler and Wood, 1966) techniques, among which (Hu et al., 2019; Lin et al., 2020; Aglin et al., 2020; Demirović et al., 2022; McTavish et al., 2022; van der Linden et al., 2024). The allure of these methods is that they provide bounds, specifically tailored for DTs, that prune large parts of the search space. Moreover, some of these works (Hu et al., 2019; Lin et al., 2020; Demirović et al., 2022; van der Linden et al., 2024) optimise jointly for accuracy and sparsity (complexity). These DP and B&B approaches seem to be the most promising methods for seeking optimal sparse DTs. Yet, they also suffer from scalability issues. Our own work in this thesis is within this framework, we design novel DP and B&B algorithms within a Reinforcement Learning (RL) framework. The richness of RL provides clear intuition into the design of our bounds and procedures, as well as formal proofs for our theoretical complexity results.

1.3 The case for Data Stream Mining

The DT algorithms we discussed earlier are Batch Learning methods, i.e. they require the provision of a fixed training dataset. Batch Learning is not the only learning paradigm that exists. In fact, the continuous storage of data requires even larger memories to keep up, which is unsustainable. Besides, many modern applications naturally have a continuous supply of data, which renders most batch algorithms inapplicable since updating them requires a whole retraining process on the new complete dataset formed of the old and newly arrived data. Bifet and Kirkby (2009) provide a detailed explanation of these issues and the need for a new learning paradigm called Data Stream Mining (or Online Learning).

In Online Learning, the motivation behind using Decision Trees is similar to Batch Learning. Due to their popularity and simple formulations, greedy DTs were considered for Online Learning, however, they necessitate calculating the impurity metrics on the whole batch of data, and as a consequence, they cannot be directly applied to handling data streams. In response to this problem, Domingos and Hulten (2000) introduced the Very Fast Decision Trees (VFDT) algorithm, which adapts the greedy DT construction to the online setting, yielding the so-called Hoeffding Trees. In each leaf of the current DT, VFDT keeps running statistics to calculate statistical estimates of the gains induced by each feature as we observe more data from the stream. A concentration inequality, based on Hoeffding's bound, is then used to decide with high confidence whether to split the leaf or not, and which feature to use in case of splitting.

This approach yielded a principled algorithm and laid the foundation for subsequent develop-

ments (Hulten et al., 2001; Bifet and Gavalda, 2009; Manapragada et al., 2018). A decade later, Rutkowski et al. (2012) noticed that Hoeffding's bound is inapplicable to the usual impurity metrics Entropy and Gini. This is because they cannot be written as a sum of independent bounded random variables. Nonetheless, these impurity metrics have bounded differences, as shown by Rutkowski et al. (2012), and as such, they satisfy McDiarmid's concentration inequality. The new resulting bounds produce the McDiarmid Tree algorithm. Yet despite the theoretical argument presented by Rutkowski, popular Data Stream Mining libraries, such as scikit-multiflow (Montiel et al., 2018a) and River (Montiel et al., 2021), still rather implement Hoeffding Trees. The reason pertains to the more conservative nature of the McDiarmid-based bounds, especially with regard to Entropy, which make McDiarmid Trees prohibitively slower than Hoeffding Trees. To make these bounds tighter, Jin and Agrawal (2003); Rutkowski et al. (2013) proposed methods based on the Gaussian approximation and the multivariate Delta method.

While there is a large corpus of literature investigating ways of seeking optimal sparse DTs in the batch setting, attention is more scarcely allocated to this problem within the online setting. Besides, extending the existing batch methods to handling streams of data is no trivial task. The Mathematical Programming methods are incompatible with data streams since the variables constituting their mathematical programs are rigid with respect to a fixed training set. DP and B&B methods might be more malleable, however, it is still unclear how to extend them to data streams, nor how such extensions would perform under the online hypothesis.

In this thesis, our objective is to develop new algorithms for seeking optimal sparse DTs within the batch and online settings. Constructing DTs can be viewed as a sequential decision making problem. On the other hand, Reinforcement Learning (RL) is a very promising framework for modeling such problems. As such, our approach consists of formulating DT optimisation within a RL framework and then developing the adequate methods to solve the problem. In the next section, we provide a brief overview of RL.

1.4 Reinforcement Learning (RL)

Sequential decision making problems are best described within a RL framework (Barto et al., 1995). Within this setting, an agent is tasked to take a series of successive actions, for which it receives rewards, with the aim of maximising the sum of these accumulated rewards. If the rewards depend only on the actions, then we are within the confines of a Multi-Armed Bandit (MAB) setting (Lattimore and Szepesvári, 2020). Otherwise, the rewards depend on both the performed action and the agent's state. The state can change when the agent takes an action, hence rendering the interaction more complicated than in the MAB setting. This interaction is generally described within the hypotheses of a Markov Decision Process (MDP) (Barto et al., 1995), which is the setting we will consider throughout our work.

A MDP consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition probability function p such that $p\left(s'|s,a\right)$ is the probability of transitioning to state s' from state s upon taking action a, and a reward function r, where $r\left(s,a,s'\right)$ is the reward of taking action a in state s and ending up in state s'. The agent's decision making is dictated by a policy, which is a function mapping the state space to the action space $\pi : \mathcal{S} \mapsto \mathcal{A}$. The objective is therefore to find the optimal policy, which takes the best action in each state accumulating the maximum amount of rewards while the agent is navigating the environment.

Throughout the years, many algorithms were developed to search for these optimal policies.

From SARSA and Q-Learning (Watkins and Dayan, 1992) to REINFORCE (Williams, 1992) and Actor-Critic methods (Konda and Tsitsiklis, 1999; Peters and Schaal, 2008; Schulman et al., 2015, 2017) to name a few. When coupled with neural network function approximators, within what is called Deep RL, these methods produced great breakthroughs in very complex domains, for which solutions were not expected for many more decades. Such impressive feats include attaining super-human performance level in highly complex strategic games such as Go and Chess (Silver et al., 2016, 2017b,a), video games such as Atari (Mnih et al., 2013, 2015), locomotion tasks (Lillicrap et al., 2015) and much more. These success stories highlight RL as a promising avenue to investigate whenever a problem can be framed as a sequential decision making task. This applies to DT construction as well.

The idea of employing RL for DT optimisation is not new, however the attention devoted to this topic is rather scarce in the literature. Bonet and Geffner (1998) proposed a Partially Observable Markov Decision Process (POMDP) formulation of the problem. However, POMDPs are known to be significantly harder to solve than MDPs, hence why subsequent works rather favour MDP formulations. Preda (2007) defines the state space as the set of all possible partitions of a given training set, and the action space as the set of rules partitioning a subset of the training set. Since the state space can be very large, Preda (2007) adopts a Linear Q-Learning algorithm to solve the MDP rather than tabular Q-Learning (Watkins and Dayan, 1992). Some later works also considered Deep RL and Monte Carlo Tree Search approaches (Xiong et al., 2017; Wen and Wu, 2021; Nunes et al., 2018). In addition to these batch methods, RL was also considered for online DT construction albeit more rarely. Garlapati et al. (2015) formulated an MDP that is naturally suited for learning from data streams, then Blake and Ntoutsi (2018) built upon it to propose a way of tackling the concept drift issue that may arise, i.e. when the underlying joint probability distribution of the features and class variable is not stationary. Garlapati et al. (2015) and Blake and Ntoutsi (2018) use tabular Q-Learning, which can be intractable for large state spaces in the absence of strategies constraining the search space. As such, these methods are not scalable and are difficult to apply real-world settings.

1.5 Motivation and Objectives

As we have discussed, finding accurate DTs of small size is a central problem for Interpretable Machine Learning. It is a significantly challenging problem that drew the attention of a large corpus of the literature for multiple decades. Currently, DP and B&B approaches appear to be the most promising methods for solving this problem in a practical fashion. Yet, their proposed search strategies are still insufficiently efficient to scale with larger datasets. Moreover, it is unclear how to generalise such methods to the online setting.

Given the propensity of DT construction to be modeled as a sequential decision making problem, and the recent successful applications of RL, we naturally seek to answer the following questions:

- How can we model DT optimisation as a MDP?
- Can we solve this MDP with a combination of DP and B&B?
- Can we improve the search strategies of the proposed DP and B&B methods from the literature?
- Can we guarantee the optimality of our approach once the algorithm terminates?

- What would be the computational complexity of our derived methods?
- How can we generalise DT optimisation to handling data streams?
- What theoretical guarantees, in terms of optimality and computational complexity, do our methods satisfy?
- How do our methods, both batch and online, compare with the state of the art theoretically and empirically?

1.6 Outline of the Thesis

To answer the questions raised in Section 1.5, we split the thesis into two parts. The first one is devoted to batch DTs while the second is devoted to online DTs. In the following, we provide the structure of the thesis:

Chapter 2. This is a Background and Related Work chapter that introduces the following:

- The classification problem and a preliminary definition of Decision Trees, along with the main notation we use.
- A survey of greedy DTs and a discussion of their limitations.
- A survey of the main approaches to constructing optimal DTs. From Mathematical Programming, to Satisfiability and DP and B&B methods. An in-depth discussion of their benefits and drawbacks is provided.
- A survey of the literature on online DTs. This literature mainly contains greedy heuristic approaches. We discuss how these inherit the limitations of their batch counterparts.
- An introduction of the main notions of RL, along with a survey on DT construction methods that employ RL.

Chapter 3. This chapter presents our first attempt at defining an adequate MDP for the DT optimisation problem. In this MDP, the state space is the space of DTs, and each action either splits a DT or terminates the episode (i.e. it transitions to an absorbing state). We seek the optimal policy via a Tree Search approach that is augmented with a DP procedure and a structured B&B search. We further provide a new bound, called Purification bound, for pruning the search space efficiently. The induced algorithm, called DT-Miner, is implemented in Python and is, unfortunately, outperformed by the state of the art C++ algorithms such as GOSDT. Nevertheless, its approach inspires the one we follow in the subsequent chapter, resulting in the main contribution of this paper.

Chapter 4. We refine DT-Miner's MDP by modifying the state space into the space of sub-DTs, a notion we will define therein. The action space and reward function are also modified accordingly. This refinement improves the information reuse of DP dramatically. Moreover, we also adapt the Purification bound, introduced in Chapter 3, to this new setting. The result is a novel algorithm called Branches that leverages an AO*-like search strategy (Nilsson, 2014; Martelli and Montanari, 1978). We prove the following theoretical guarantees for Branches:

- Whenever Branches terminates, the returned solution is the true optimal sparse DT with respect to our objective function that penalises DTs with many splits.

- We provide a complexity analysis of BRANCHES in the form of an upper bound on the number of branch evaluations that the algorithm performs. To our knowledge, such complexity analysis is rarely discussed in the literature. We are only aware of (Hu et al., 2019, Theorem E.2), which provides such result for the algorithm OSDT. Naturally, we ask: **How does the complexity analysis of Branches compare to OSDT's?** Due to the difficulty of comparing the two bounds analytically, we rather provide a numerical comparison on several settings, where we show that our bound is significantly smaller.

We compare BRANCHES with the state of the art through a series of experiments. Despite being implemented in Python, BRANCHES outperforms its C++ competitors on a number of metrics, such as runtime, number of iterations before termination and anytime behaviour. This chapter employs an equivalent formalism to the one we present in the following paper:

- Chaouki, A., Read, J., and Bifet, A. (2025). Branches: Efficiently Seeking Optimal Sparse Decision Trees via AO*.

Chapter 5. Here we move to the online setting and we reconsider the MDP introduced in Chapter 3. The approach we follow in this setting is best viewed from a Monte Carlo Tree Search (MCTS) lens. We devise two algorithms, UPPER CONFIDENCE BOUND DECISION TREES (UCDT) and EPSILON-GREEDY DECISION TREES (EGDT), which explore the search space with the UCB and Epsilon-greedy policies respectively. We show that, asymptotically, UCDT and EGDT converge to the optimal DT. However, as of today, we are still incapable of deriving finite-time guarantees of these methods, in the form of rates of convergence for example. Nevertheless, from an empirical stand-point, we show that our methods exhibit better prequential accuracy and final accuracy compared to the state of the art online DTs. This chapter will be compiled with the next chapter into a journal paper.

Chapter 6. This chapter naturally follows Chapter 5 by investigating the use of another popular policy, Thompson Sampling. This policy is of Bayesian nature, and as such it necessitates the definition of prior distributions along with their posterior updates. To achieve this, a series of approximations are used, thus yielding the THOMPSON SAMPLING DECISION TREES (TSDT) algorithm. In similar fashion to UCDT and EGDT, TSDT is asymptotically optimal. Furthermore, TSDT outperforms the state of the art in terms of prequential and final accuracies. We note that finite-time guarantees are also very challenging to derive for TSDT. This chapter is based on the following publication:

- Chaouki, A., Read, J., and Bifet, A. (2024). Online learning of decision trees with Thompson sampling. In Dasgupta, S., Mandt, S., and Li, Y., editors, *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 2944–2952. PMLR.

This paper was the recipient of the "Outstanding Student Paper Highlights" award.

Chapter 7. Since BRANCHES constitutes our main contribution in the batch setting, naturally, we considered generalising it to handling data streams. The induced algorithm, called Online-BRANCHES, differs in essence from UCDT, EGDT and TSDT. It does not run forever but rather includes a termination condition. As a consequence it does not satisfy the asymptotic optimal convergence results of UCDT, EGDT and TSDT. Indeed, when Online-BRANCHES terminates, there is always a small but non-zero probability that it could return a suboptimal DT. Nevertheless, this formulation allows us to derive the desired finite-time guarantee in the form

of a high-probability optimal termination after the observation of a certain number of data samples. We validate our theoretical findings with an empirical evaluation, in which we show that Online-Branches outperforms UCDT, EGDT and TSDT. A paper based on this chapter is to be submitted to The 28th International Conference on Artificial Intelligence and Statistics in October.

Chapter 2

Background and Related Work

2.1 Classification Problem

Throughout the thesis, we consider classification problems with categorical features only $X = \left(X^{(1)}, \dots, X^{(q)}\right)$ and class variable $Y \in \{1, \dots, K\}$ such that:

$$\forall i \in \{1, \dots q\} : X^{(i)} \in \{1, \dots, C_i\}, C_i \geq 2$$

where $q \geq 2$ and $K \geq 2$. We denote $\mathcal{X} = \prod_{i=1}^{q} \{1, \dots, C_i\}$ the input (or features) space.

In Part I, we consider the batch setting where we are provided with $\mathcal{D} = \{(X_m, Y_m)\}_{m=1}^{n}$ a dataset of $n \geq 1$ examples. In Part II, we rather observe data (X_t, Y_t) at each time t through a continuous stream, we assume that each sample (X_t, Y_t) is drawn from a joint probability distribution $\mathbb{P}_{X,Y}$. We further assume this joint probability distribution to be stationary and that the samples are observed in an i.i.d. fashion.

The classifier models we consider throughout this thesis are univariate Decision Trees (DTs). In this section we provide a brief definition of these models, we reserve the detailed definition for the main chapters of the thesis. A DT T is a Directed Acyclic Graph (DAG) where each node has one unique parent, except one node called the root, which has no parent. The nodes of T from which there are no outgoing edges are called leaves while the remaining nodes are called internal nodes. Each node in T represents a subset of $\mathcal{X}$, with the root representing the whole input space $\mathcal{X}$. Let $\mathcal{N}$ be a node in T with $\mathcal{X}_\mathcal{N}$ the subset of $\mathcal{X}$ represented by $\mathcal{N}$. Then the children of $\mathcal{N}$ represent a partition of $\mathcal{X}_\mathcal{N}$ based on only one feature, i.e. there exists $1 \leq i \leq q$ such that $\mathcal{N}_j$, for $j \in \{1, \dots, C_i\}$, are the children of $\mathcal{N}$ in T representing the subsets $\mathcal{X}_{\mathcal{N}_j} = \{X \in \mathcal{X}_\mathcal{N} : X^{(i)} = j\}$ respectively. As such, since the root represents $\mathcal{X}$, the leaves of a DT represent a partition of $\mathcal{X}$ and any input $X \in \mathcal{X}$ is contained within one unique leaf.

The purpose of a DT T is not only to partition the input space $\mathcal{X}$ but also to classify any example $X \in \mathcal{X}$. To achieve this, T labels each leaf with a class in $\{1, \dots, K\}$ and classifies all inputs in a leaf with the corresponding assigned class. These class assignments are often performed by choosing the majority class of a leaf given a training dataset. This classification strategy makes DTs interpretable as classifying each input amounts to first sorting it into its corresponding leaf, then classifying it with the leaf's label. Thus, granted these leaves are not too deep and are not too numerous, the graphical representation of DTs makes them prominent interpretable models. In this thesis, we distinguish two important notions:

- Optimal DTs: they are the most accurate DTs that do not exceed a hard constraint such

as maximum depth or maximum number of nodes, or other types of constraints.

- Optimal Sparse DTs: this is a subset of Optimal DTs that actively minimises some criterion on complexity such as the number of leaves or the number of splits. These solutions are usually fetched by optimising an objective that includes a soft constraint on the complexity criterion, or by solving multiple Optimal DT problems while gradually increasing the complexity criterion of the considered DTs. Our work considers the former approach.

Optimal Sparse DTs have the additional benefit, over Optimal DTs, of being the solutions of lowest complexity that we can hope for at their achieved accuracy. Upon finding an Optimal Sparse DT of accuracy $0 < \alpha \leq 1$ and number of splits $s \in \mathbb{N}$, we are certain that there exists no other DT of lower number of splits $< s$ that is at least as accurate as the found solution. In other words, Optimal Sparse DTs belong to the pareto front of the joint optimisation problem that maximises accuracy and minimises complexity. Optimal DTs do not necessarily satisfy this property.

Solving for sparsity: We say that a method solves for sparsity if it seeks Optimal Sparse DTs.

2.2 Greedy Decision Trees

Due to the NP-completeness of DT optimisation, greedy heuristic methods, such as ID3 (Quinlan, 1986), C4.5 (Quinlan, 2014) and CART (Breiman et al., 1984), have been favoured historically and are still very popular amongst practitioners. These methods construct DTs iteratively in a top-down fashion. In this section, we describe their principled approach and we indicate the differences between them.

We consider the classification problem introduced in Section 2.1, more specifically, we consider the batch setting where we are provided with a dataset $\mathcal{D}$. Starting from the root, the greedy DT heuristics iteratively split the leaves as follows. Let l be a leaf and $\mathcal{D}_l \subset \mathcal{D}$ the subset of data in l. Ideally, leaf l is desirable if it is pure, i.e. if all the data in $\mathcal{D}_l$ share the same class. We evaluate the impurity of l with a metric $\mathcal{G}(l)$, the higher $\mathcal{G}(l)$ is, the less desirable l becomes, and the more incentive we have to split l seeking purer leaves. In general the impurity of l is evaluated with either the Entropy or the Gini impurity:

- **Entropy:** $\mathcal{G}(l) = -\sum_{k=1}^{K} \frac{n_k(l)}{n(l)} \log\left(\frac{n_k(l)}{n(l)}\right)$
- **Gini impurity:** $\mathcal{G}(l) = 1 - \sum_{k=1}^{K} \left(\frac{n_k(l)}{n(l)}\right)^2$

where $n(l)$ is the number of data in $\mathcal{D}_l$ and $n_k(l)$ is the number of data in $\mathcal{D}_l$ that are of class k. Without loss of generality, suppose we split l with respect to feature $X^{(1)}$, if $X^{(1)}$ is discrete and takes values in $\{1, \ldots, C_1\}$, then the resulting children $l_1, \ldots, l_{C_1}$ contain subsets $\mathcal{D}_{l_j} = \{X \in \mathcal{D}_l : X^{(1)} = j\} \subset \mathcal{D}_l$ for $1 \leq j \leq C_1$. On the other hand, if $X^{(1)}$ is continuous, then a split point s is chosen in the interior of the support of $X^{(1)}$, which results in two children $l_{\leq s}$ and $l_{>s}$ containing subsets $\mathcal{D}_{\leq s} = \{X \in \mathcal{D} : X^{(1)} \leq s\}$ and $\mathcal{D}_{>s} = \{X \in \mathcal{D} : X^{(1)} > s\}$ respectively. In any case, let $l_1, \ldots, l_{C_1}$ be the children leaves that result from splitting l with respect to feature $X^{(1)}$, this notation also encompasses the continuous case where we would have $C_1 = 2$ and $l_1 = l_{\leq s}$ and $l_2 = l_{>s}$. The gain measure of this split is defined as the average decrease in impurity:

$$\text{Gain}\left(l, X^{(1)}\right) = \mathcal{G}(l) - \sum_{j=1}^{c_1} \frac{n(l_j)}{n(l)} \mathcal{G}(l_j)$$

When $X^{(1)}$ is continuous, the split point s is chosen to maximise Gain $\left(l, X^{(1)}\right)$. With this in mind, the heuristic chooses the feature $\text{att}^*(l)$ maximising the gain in leaf l:

$$\text{att}^*(l) = \text{Argmax}_{1 \leq m \leq q} \text{Gain}\left(l, X^{(m)}\right)$$

This splitting process proceeds iteratively until the induced leaves satisfy one of the following criteria:

- The leaf contains less data than a predefined threshold.
- No split induces a positive gain for the leaf.

While it seems odd to define the gain based on impurity metrics rather than the misclassifiaction error (since our ultimate objective is to minimise the misclassifiaction error), Breiman et al. (1984) motivate this choice by a continuous growth of the DT. It appears that using the misclassification error stunts the growth of DTs by reaching the second termination condition prematurely. Nonetheless, the misclassification error is still employed in the pruning phase, where the objective is to shrink the DT by discarding the splits that do not induce a significant gain in misclassification error. Formally, a small complexity parameter $0 \leq \alpha < 1$ is defined, α is the smallest gain in misclassification error tolerated. Then following a bottom-up approach from the leaves to the root, splits that induce less gain in misclassification error than α are discarded, i.e. the children $l_1, \ldots, l_C$ of a leaf l are pruned if they satisfy:

$$\mathcal{H}(l) - \sum_{j=1}^{C} \frac{n(l_j)}{n(l)} \mathcal{H}(l_j) < \alpha$$

where $\mathcal{H}(l)$ is the misclassification error of leaf l defined as:

$$\mathcal{H}(l) = \frac{1}{n(l)} \sum_{(X,Y) \in \mathcal{D}_l} \mathbb{1}\left\{Y \neq \text{Argmax}_{1 \leq k \leq K}\{n_k(l)\}\right\}$$

ID3, C4.5 and CART follow this general DT construction scheme, but they differ in some regards:

- ID3 uses Entropy and is specifically tailored for discrete (categorical) features. Moreover, it does not implement a pruning strategy.
- C4.5 is an improved version of ID3. It also employs Entropy, but can handle both discrete and continuous features along with missing values. Unlike ID3, C4.5 employs a pruning strategy.
- CART is perhaps the most popular of these algorithms, it uses Gini impurity, can handle both discrete and continuous features along with missing values, and employs a pruning strategy. Furthermore, CART builds binary Decision Trees.

While these heuristics are fast and scalable, their greedy nature often leads to suboptimal and overly complex DTs, detracting from their interpretability. This undermines the primary motivation behind using DTs. In the next section, we survey the literature of Optimal (Sparse) DTs.

2.3 Optimal Decision Trees

The suboptimality issue of greedy Decision Trees has been thoroughly in the literature, mainly through a *Mathematical Programming* framework ranging from *Continuous Optimisation* to *Mixed Integer Programming (MIP)* and *Satisfiability (SAT)*. More recently, breakthroughs in scalability have emerged from a combination of Dynamic Programming and Branch & Bound search approaches. In this section, we survey these methods and discuss their respective benefits and drawbacks.

2.3.1 Continuous Optimisation

In the early 1990's, Bennett (1992) proposed a Linear Programming method for constructing binary oblique Decision Trees called Multi-Surface Method Tree (MSMT). We recall that, unlike univariate (orthogonal) DTs, multivariate (oblique) DTs can use multiple features per decision rule at the level of an internal node. MSMT is a greedy heuristic that follows a similar top-down construction to CART, ID3 and C4.5. But rather than seeking the feature (and split point in the continuous case) maximising the gain measure, MSMT's objective is to find a separating hyperplane minimising a weighted misclassification error. Concretely, let l be a leaf we want to split, MSMT considers binary classification problems, as such, let $A(l)$ be the $n_+(l) \times q$ matrix where each row is an input $X \in l$ labelled with $Y = 0$, and $B(l)$ the $n_-(l) \times q$ matrix where each row is an input $X \in l$ labelled with $Y = 1$. $n_+(l)$ and $n_-(l)$ are the number of inputs in l with labels $Y = 0$ and $Y = 1$ respectively. Let e be a vector of ones in a real space of arbitrary dimension (depending on the context). Then, MSMT seeks to solve:

$$(w(l), \gamma(l)) = \underset{w \in \mathbb{R}^q, \gamma \in \mathbb{R}}{\text{Argmin}} \left\{ \frac{1}{n_+(l)} \left\| (-A(l)\, w + e\gamma + e)_+ \right\|_1 + \frac{1}{n_-(l)} \left\| (B(l)\, w - e\gamma + e)_+ \right\|_1 \right\}$$

Two children leaves are then created, they correspond to the subsets $\{X \in l | w(l)\, X > \gamma(l) + 1\}$ and $\{X \in l | w(l)\, X < \gamma(l) - 1\}$, the remaining points, i.e. $\{X \in l | \gamma(l) - 1 \leq w(l)\, X \leq \gamma(l) + 1\}$, can be put in either child arbitrarily. Bennett (1992) showed that this problem is equivalent to the following Linear Program:

$$\min_{w,\gamma,y,z} \left\{ \frac{1}{n_+(l)} ey + \frac{1}{n_-(l)} ez \;\middle|\; y \geq -A(l)\, w + e\gamma + e, z \geq B(l)\, w - e\gamma + e, y \geq 0, z \geq 0 \right\}$$

The Linear Program is then solved with a solver such as Simplex. Although MSMT is greedy, we gave an overview of this method because Linear Programming will be employed by Bennett in subsequent works seeking Optimal Decision Trees.

Bennett (1994) devises the Global Tree Optimisation (GTO) algorithm for constructing optimal binary oblique DTs. Given a fixed DT structure, in terms of the number of internal decision nodes, the tree depth and the classification labels at the leaves, GTO represents the DT as a set of disjunctive linear inequalities under which it seeks to minimise the misclassification error. As a result, Bennett (1994) formulates a MultiLinear Program (MLP) aiming at minimising a non-convex objective function over a polyhedral set defining the constraints. Frank-Wolfe algorithm is then applied to solve the MLP in practice. In a subsequent work, Bennett and Blue (1996) note that Frank-Wolfe might not be the best solver for this MLP, indeed Frank-Wolfe assumes the objective function to be convex and differentiable, while the error function in this MLP is non-convex. Furthermore, Bennett and Blue (1996) consider two other objective functions (Minimum Error and Count Error) which are not even continuous,

and as such, their induced MLPs are not compatible with Frank-Wolfe. Bennett and Blue (1996) show that for all the proposed objective functions, it suffices to look for a solution among the extreme points. Hence, to go beyond the limitations of Frank-Wolfe to differentiable convex functions, the authors developed a global search heuristic called Extreme Point Tabu Search (EPTS), which is essentially a Tabu Search over the extreme points of the MLP.

While GTO is an elegant formulation of the Decision Tree optimisation problem, it comes with a substantial computational cost that limits its application to small DT structures with few internal nodes. To alleviate this issue, Norouzi et al. (2015) follow a different approach based on structured prediction with latent variables where the authors formulate a convex-concave upper bound on the empirical loss (log-loss for classification and squared loss for regression). Norouzi et al. (2015) consider a classification problem with maximal binary DTs with a fixed number m of internal nodes, and thus $m+1$ leaf nodes. Each DT is parameterised with (W, Θ) where $W = (w_1, \dots, w_m)^T \in \mathbb{R}^{m \times q}, \Theta = (\theta_1, \dots, \theta_{m+1})^T \in \mathbb{R}^{(m+1) \times K}$, q is the number of features and K the number of classes. Internal nodes are indexed with $1 \leq i \leq m$ so that $w_i^T X = 0$ defines the splitting hyperplane of node i, i.e. any point $X \in i$ is directed to the left child of i if it satisfies $\text{sign}\left(w_i^T X\right) = -1$, and to the right child otherwise. On the other hand, leaves are indexed with $1 \leq j \leq m+1$ so that the prediction of the DT at the level of leaf j is prescribed by the following softmax function:

$$p\left(Y = k | X \in j\right) = \frac{\exp\left(\theta_j^{(k)}\right)}{\sum_{l=1}^{K} \exp\left(\theta_j^{(l)}\right)}; \; \theta_j = \left(\theta_j^{(1)}, \dots, \theta_j^{(K)}\right)$$

For this formulation, the log-loss $l\left(\theta, Y\right) = -\theta^{(Y)} + \log\left(\sum_{l=1}^{K} \exp\left(\theta^{(l)}\right)\right)$ is naturally a suitable choice. This leads to the global empirical loss:

$$\mathcal{L}\left(W, \Theta; \mathcal{D}\right) = \sum_{(X,Y) \in \mathcal{D}} l\left(\Theta^T f\left(\text{sign}\left(WX\right)\right), Y\right)$$

where we recall that $\mathcal{D}$ is the batch dataset. sign (.) is applied element-wise to the vectors WX, and f maps a vector in $\mathbb{H}^m = \{-1, 1\}^m$ to a vector in $\mathbb{R}^{m+1}$ where all the elements are 0 except one that is equal to 1. This non-zero element corresponds to the leaf where an input X is sorted with parameter W, hence why f is called the navigation function. Direct minimisation of this objective is very challenging because it is discontinuous and piece-wise constant with respect to W, moreover, it is unclear how to account for the navigation function f. Given these challenges, Norouzi et al. (2015) rather derive an upper bound which has the form of a structured prediction problem with latent variables:

$$l\left(\Theta^T f\left(\text{sign}\left(WX\right)\right), Y\right) \leq \max_{g \in \mathbb{H}^m} \left[g^T WX + l\left(\Theta^T f\left(g\right), Y\right)\right] - \max_{h \in \mathbb{H}^m} \left[h^T WX\right]$$

This upper bound is then used as a surrogate objective for the following regularised minimisation problem:

$$\min_{W, \Theta} \sum_{(X,Y) \in \mathcal{D}} \left[\max_{g \in \mathbb{H}^m} \left\{g^T WX + l\left(\Theta^T f\left(g\right), Y\right)\right\} - \max_{h \in \mathbb{H}^m} \left\{h^T WX\right\}\right]; \; \forall 1 \leq i \leq m : \|w_i\|^2 \leq \nu$$

This surrogate objective is piece-wise convex-concave with respect to W. In addition, the constraints on W define a convex set. For these reasons, Norouzi et al. (2015) solve this optimisation problem with Stochastic Gradient Descent (SGD). While Norouzi et al. (2015)'s

approach scales better with larger data sets and more complex DTs than GTO, the use of SGD only guarantees convergence to a local minimum. Besides, even a global optimal convergence for the surrogate objective does not necessarily entail an optimal convergence for the empirical loss. An equivalence result, between optimising the surrogate objective and the true empirical loss, is lacking. Such result is provided in another approach credited to Blanquero et al. (2021), which we discuss next.

In a more recent work, Blanquero et al. (2021) follow a different path where the decision rules at the level of internal nodes are no longer hard rules but rather soft ones, i.e. data points are sorted to the left or right child randomly according to a Bernoulli probability distribution, which yields the Optimal Randomized Classification Tree (ORCT) algorithm. The motivation behind this formulation is to smooth out the discontinuities related to the hard decision rules. Similarly to Norouzi et al. (2015), Blanquero et al. (2021) also consider binary DTs with m internal nodes and $m+1$ leaves. Consider the classification problem in Section 2.1, but with continuous features $\forall j \in \{1,\dots,q\} : X^{(j)} \in [0,1]$. Let F be a cumulative distribution function and $W = (w_1,\dots,w_m) \in \mathbb{R}^{q\times m}$ a matrix of coefficients defining the decision rules of the DT. For any internal node $1 \leq u \leq m$, the probability of sorting data point X_i in the left child of u is modeled as:

$$p_{iu}(w_u) = F\left(w_u^T X_i\right)$$

The next step now is to derive the probability that a data point lands on a leaf. To this end, for any leaf $1 \leq l \leq m+1$, Blanquero et al. (2021) define $N_L(l)$ the set of ancestor nodes of l for which the path to l traverses the left child, $N_R(l)$ is defined similarly but for the right child. Then the probability that X_i lands on leaf l is:

$$P_{il}(W) = \mathbb{P}[X_i \in l] = \prod_{u\in N_L(l)} p_{iu}(w_u) \prod_{u\in N_R(l)} [1 - p_{iu}(w_u)]$$

The remaining ingredient for formulating the objective function is to define the decision variables at the level of the leaves. These are defined for any leaf $1 \leq l \leq m+1$ as:

$$C_{kl} = \mathbb{1}\{l \text{ is labelled with class } k\}$$

Since each leaf is assigned a unique class, then the decision variables C_{kl} satisfy:

$$\forall 1 \leq l \leq m+1 : \sum_{k=1}^{K} C_{kl} = 1$$

In addition to C_{kl}, cost variables $V_{Y_i k} \geq 0$ are defined to model the cost of misclassifying X_i, and as such $V_{Y_i k} = 0$ if $k = Y_i$. Naturally, the objective is to minimise the expected misclassification cost, as a result, the following Mixed Integer Non-Linear Optimisation (MINLO) problem is formulated:

$$\begin{aligned}
\min_{W,C_{kl}} \quad & \frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{m+1} P_{il}(W) \sum_{k=1}^{K} V_{Y_i k} C_{kl} \\
\text{s.t:} \quad & \sum_{k=1}^{K} C_{kl} = 1, \quad l \in \{1,\dots,m+1\} \\
& \sum_{l=1}^{L} C_{kl} \geq 1, \quad k \in \{1,\dots,K\} \\
& w_u \in [-1,1]^q, \quad u \in \{1,\dots,m\} \\
& C_{kl} \in \{0,1\}, \quad k \in \{1,\dots,K\}, l \in \{1,\dots,m+1\}
\end{aligned}$$

The second constraint $\sum_{l=1}^{L} C_{kl} \geq 1$ was added to avoid the case of never predicting the minority class. It is interesting to note that this formulation circumvents the issue of the navigation function that plagued Norouzi et al. (2015)'s empirical loss. Nevertheless, solving the MINLO problem is still challenging, hence why Blanquero et al. (2021) relax it into the following Non-Linear Continuous Optimisation (NLCO) problem:

$$\begin{aligned}
\min_{W, C_{kl}} \quad & \frac{1}{n} \sum_{i=1}^{n} \sum_{l=1}^{m+1} P_{il}(W) \sum_{k=1}^{K} V_{Y_i k} C_{kl} \\
\text{s.t:} \quad & \sum_{k=1}^{K} C_{kl} = 1, \quad l \in \{1, \ldots, m+1\} \\
& \sum_{l=1}^{L} C_{kl} \geq 1, \quad k \in \{1, \ldots, K\} \\
& w_u \in [-1, 1]^q, \quad u \in \{1, \ldots, m\} \\
& C_{kl} \geq 0, \quad k \in \{1, \ldots, K\}, l \in \{1, \ldots, m+1\}
\end{aligned}$$

It is tempting to consider the NLCO problem as it is easier to deal with than the MINLO problem. However, we should first make sure that both problems are equivalent. Fortunately, (Blanquero et al., 2021, Theorem 1) proves this result.

> There exists an optimal solution of the NLCO such that:
>
> $$\forall k \in \{1, \ldots, K\}, l \in \{1, \ldots, m+1\} : C_{kl} \in \{0, 1\}$$

By virtue of this Theorem, we can look for solutions of the NLCO problem for which all decision variables at the leaves satisfy $C_{kl} \in \{0, 1\}$. Such solution coincides with the optimal solution of the MINLO problem. To this end, Blanquero et al. (2021) employ the IPOPT solver (Wächter and Biegler, 2006) because it is tailored for non-linear optimisation problems. However, IPOPT is a local optimiser, it does not guarantee finding the global optimal solution for the NLCO problem. ORCT also suffers from other limitations. While its soft decision rules are justified for continuous features because they introduce fuzzy regions, the same reasoning does not hold for categorical features. Thus ORCT is more tailored for classification problems with continuous features than those with categorical features. Furthermore, despite the fact that the number of constraints only depends on the number of classes K and the number of internal nodes m, the experimental results seem to indicate that ORCT suffers from scalability issues. For the MGT dataset, to converge, ORCT takes $551s$ for a maximum DT depth of $D = 2$, $t = 2018s$ for $D = 3$ and $t = 5603s$ for $D = 4$, where s refers to seconds. This computation time is significantly higher than those reported for the other datasets, and we note that MGT is a binary classification problem with only 10 features (the second lowest among all the considered datasets), but with significantly more data points (19020) than the other datasets. This seems to indicate that ORCT still suffers from scalability issues that depend on the size of the dataset.

Continuous optimisation is a very attractive approach for deriving optimal Decision Trees, it offers problem formulations that can be readily solved with off the shelf open-source solvers. On the other hand, these methods suffer from the following limitations:

- They fix the structure of the DT and only seek its optimal internal splits and class labels of the leaves (In (Bennett, 1994; Bennett and Blue, 1996), even the class labels are fixed as part of the chosen DT structure). As such, these methods do not solve for sparsity.

- Local optimisers are often used despite the non-convexity of the objective function. This point holds for all the cited methods except GTO (Bennett and Blue, 1996).
- Continuous optimisation does not offer enough flexibility for deriving optimal orthogonal Decision Trees. This is due to the inherent discrete nature of the internal nodes decision variables in these models. As discussed in Section 1.1, orthogonal DTs are more interpretable than oblique DTs, hence why there is a need to derive optimal DTs of this nature as well. Mixed-Integer Programming (MIP) offers an appropriate framework for this problem, which we review in the next section.

2.3.2 Mixed-Integer Programming (MIP)

Bertsimas and Dunn (2017) observed that the discrete nature of the problem underlying the construction of optimal DTs is best described within a Mixed-Integer Programming (MIP) framework. Bertsimas and Dunn (2017) also remarked that, while the aversion to MIP in the Machine Learning and Statistics literature is due to the belief that MIP problems are intractable even for small to medium sized instances, such belief might no longer be justified due to the stellar progress of both modern MIP solvers and hardware. This observation stirred up a surge in interest around MIP-based approaches.

Bertsimas and Dunn (2017) introduce the Optimal Classification Trees (OCT) algorithm for orthogonal DTs and the Optimal Classification Trees with Hyperplanes (OCT-H) for oblique DTs. In this review, we only focus on OCT. OCT is tailored for maximal binary DTs with a fixed number m of internal nodes (indexed with $1 \leq u \leq m$) and $m+1$ leaves (indexed with $m+1 \leq l \leq 2m+1$). Let $W = (w_1, \ldots, w_m) = \left(w_u^{(j)}\right) \in \mathbb{R}^{q \times m}$ and $B = (b_1, \ldots, b_m) \in \mathbb{R}^{1 \times m}$ be the coefficients defining the decision rules of the DT. For any data point X_i in an internal node $u \in \{1, \ldots, m\}$, $\text{sign}\left(w_u^T X_i - b_u\right) = -1$ decides that X_i is sorted into the left child of u, while $\text{sign}\left(w_u^T X_i - b_u\right) = 1$ decides that X_i is sorted into the right child of u. Since OCT considers orthogonal DTs, it imposes the following constraints:

$$\begin{aligned}
&\sum_{j=1}^{q} w_u^{(j)} = d_u, \ \forall u \in \{1, \ldots, m\} \\
&0 \leq b_u \leq d_u, \ \forall u \in \{1, \ldots, m\} \\
&w_u^{(j)} \in \{0, 1\}, \ \forall j \in \{1, \ldots, q\}, u \in \{1, \ldots, m\}
\end{aligned}$$

where $d_u = \mathbb{1}\{\text{node } u \text{ applies a split}\}$. When $d_u = 0$ then we have both $\forall j \in \{1, \ldots, q\} : w_u^{(j)} = 0$ and $b_u = 0$. This is interpreted as node u being a leaf. In this sense, OCT can solve for sparsity unlike the previously considered continuous optimisation methods, i.e. OCT optimises the DT structure as well as the misclassification error. The OCT model further defines the data assignment variables $z_{iu} = \mathbb{1}\{X_i \in v\}$ for any node $1 \leq v \leq 2m+1$ and any data point $1 \leq i \leq n$. To minimise the misclassification error, OCT introduces the cost variables $\forall i \in \{1, \ldots, n\}, k \in \{1, \ldots, K\} : Y_{ik} = \mathbb{1}\{Y_i = k\}$. The misclassification error of a leaf $l \in \{m+1, \ldots, 2m+1\}$ is then:

$$L_l = \sum_{i=1}^{n} z_{il} - \max_{1 \leq k \leq K} \left\{ \sum_{i=1}^{n} Y_{ik} z_{il} \right\} = \min_{1 \leq k \leq K} \sum_{i=1}^{n} \left(1 - Y_{ik}\right) z_{il}$$

Unlike the continuous optimisation methods reviewed in Section 2.3.1, OCT seeks to minimise both the misclassification error and the sparsity of the DT solution. The sparsity (or complexity)

of a DT is defined as the number of its internal splits $\sum_{u=1}^{m} d_u$. A complexity parameter $\alpha \geq 0$ is set, upon which OCT seeks to minimise the normalised misclassification error constrained with a penalty on sparsity:

$$\frac{1}{\widehat{L}} \sum_{l=m+1}^{2m+1} L_l + \alpha \sum_{u=1}^{m} d_u$$

where $\widehat{L}$ is the misclassification error that is obtained by predicting the most common class in the entire dataset. There are many constraints that define the final MIP OCT seeks to solve, instead of re-transcribing them here, we refer the reader to (Bertsimas and Dunn, 2017, Formula (24)) for the full MIP formulation. The authors use Gurobi 6.5 to solve this MIP. Unfortunately, while OCT offers the possibility of seeking optimal orthogonal sparse DTs, it suffers from serious scalability issues. In the authors own words (Bertsimas and Dunn, 2017, Page 12):

The difficulty of the model is primarily determined by the number of binary variables z_{il} , which is $n2^D$. Empirically we observe that we can find high-quality solutions in minutes for depths up to 4 for datasets with thousands of points. Beyond this depth or dataset size, the rate of finding solutions is slower, and more time is required.

D in this quote refers to the maximum depth, so that $2^D = m + 1$ leaves. Thus, the scalability issue of OCT pertains to the exponential dependence of the number of z_{il} variables on depth D and its linear dependence on the training set size n. Moreover, despite the claim *"we can find high-quality solutions in minutes for depths up to 4 for datasets with thousands of points."*, a closer look at the experimental results for the MONK1 dataset indicates that the scalability issue also plagues a depth of 4 with just 124 data points. Indeed, MONK1 has only $n = 124$ data points, $q = 11$ features and $K = 2$ classes, and an optimal solution of depth $D = 4$ that perfectly separates the dataset (100% accuracy), this ground-truth solution is provided with the description of the dataset in the UCI repository (we will also find it later with our own proposed algorithms and illustrate it in Fig. 4.15). Yet despite the 30 minutes of allocated time, OCT with $D = 4$ fails to achieve the optimal solution (Bertsimas and Dunn, 2017, Table 16), even though we recall and emphasize that MONK1 only contains 124 data points. In fact, even OCT-H, which is tailored for retrieving the more powerful oblique DTs, fails at this task (Bertsimas and Dunn, 2017, Table 19). In Chapter 3 and especially in Chapter 4, we will see that the algorithms we introduce retrieve the optimal solution for MONK1 in significantly shorter computational times than 30 minutes, in fact, they retrieve it in less than one second.

In the same year, Verwer and Zhang (2017) considered a different MIP formulation and derived another algorithm called Decision Trees as Integer Programs (DTIP). The main difference with OCT is that DTIP encodes the decision variables at the level of the leaves in binary rather than unary form, as a consequence, the number of these variables drops from $n2^D$ (for OCT) to $2nD$ (for DTIP). Moreover, accuracy is not the only objective that is considered for DTIP, other objectives such as a discrimination-aware classification and an improved learning from imbalanced data are also considered. Nonetheless, DTIP still suffers from the following limitations:

- The number of decision variables at the level of the leaves still increases linearly with the size of the training set.
- the number of constraints defining the MIP is still exponential in the depth.
- Unlike OCT, DTIP does not solve for sparsity, but rather seeks the optimal maximal binary DT of a prespecified depth D.

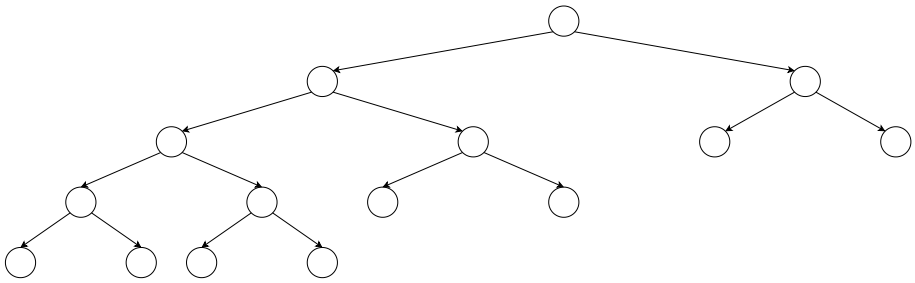


Figure 2.1: The Decision Tree topology that Günlük et al. (2021) considered for MONK1 to achieve the optimal solution.

Two years later, Verwer and Zhang (2019) addressed the first limitation with their Binary encoding for constructing Optimal Classification Trees (BinOCT) algorithm. This algorithm uses a binary encoding of both integer and continuous features in what the authors call Integer Decision Thresholds. This scheme leads to a number of decision variables at the level of the leaves that is independent of the training set size, and rather depends logarithmically on the number of unique feature values. However, the remaining two limitations of DTIP still hold for BinOCT. Furthermore, while BinOCT achieves better out-of-sample accuracy on MONK1 than OCT, it still fails at retrieving the true optimal sparse DT even with an allocated time of 10 minutes.

Other MIP formulations were also proposed in the literature. The SVM1-ODT algorithm, introduced by Zhu et al. (2020), seeks optimal oblique DTs for which a 1-norm SVM model is trained at the level of each leaf with the aim of achieving a better out-of-sample accuracy than models with majority class predictors at the leaves. In addition, Zhu et al. (2020) also address the scalability issue of MIP-based models for large training sets. The proposed solution is a data selection method that aims at extracting a small subset maximising the information captured from the whole training set. However, as we have discussed earlier, we are more interested in orthogonal DTs because they provide better interpretability than oblique DTs. In fact, this interpretability issue might be accentuated for SVM1-ODT because of the SVM predictive model at the level of the leaves. Moreover, it is worth noting that SVM1-ODT also fails at deriving the optimal solution for MONK1, albeit the authors only considered it with maximum depth $D = 2$ while the optimal DT is of depth 4. Nonetheless, the depth of 4 pertains to the optimal orthogonal DT with majority class predictors at the leaves, thus perhaps $D = 2$ might be sufficient for oblique DTs, especially as their leaf predictors are augmented with SVM.

More recently, another interesting MIP was put forward by Günlük et al. (2021), yielding the Optimal Decision Trees (ODT) algorithm. In this MIP, the authors aim at deriving optimal orthogonal DTs for binary classification problems with categorical features. Unlike Verwer and Zhang (2017, 2019) and Zhu et al. (2020) who only consider maximal binary DTs of a certain fixed depth, Günlük et al. (2021) allow any kind of binary DT topology as long as it it prespecified in the MIP formulation. ODT exploits the categorical nature of the considered features to define internal decisions (splits) with respect to a subset of categorical values and its complement. Concretely, suppose that the j^{th} feature satisfies $X^{(j)} \in \mathcal{C}^{(j)} = \{1, \ldots, C_j\}$ and let $\mathcal{C}_L^{(j)} \subset \mathcal{C}^{(j)}, \mathcal{C}_R^{(j)} = \mathcal{C}^{(j)} \backslash \mathcal{C}_L^{(j)}$, then splitting with respect to $X^{(j)}$ is of the decision form $X^{(j)} \in \mathcal{C}_L^{(j)}$ for branching at the left child and $X^{(j)} \in \mathcal{C}_R^{(j)}$ for branching at the right child. As such, we already observe that the number of possible splits for the $j^{(th)}$ feature alone is $2^{C^{(j)}}$, which grows exponentially with the number of categories that $X^{(j)}$ can take. Moreover, ODT's formulation,

which can be found in (Günlük et al., 2021, (8a), (8c), (8c)), also involves a number of decision variables at the level of the leaves that grows linearly with the size of the training set. There is one other remark that is worth noting here. Unlike the previous methods, ODT achieves the true optimal sparse DT solution for MONK1, however, a closer look indicates that this is only achieved when the topology (structure) in Fig. 2.1 is considered[1]. This topology with 8 leaves is in fact the topology of the true optimal sparse DT for MONK1. As such, prespecifying it for ODT makes finding the optimal DT significantly less challenging, but it assumes prior knowledge of this topology before running the algorithm. Furthermore, this topology is not a trivial one to guess, thus the recommendation in (Günlük et al., 2021, section 4.2) would only be valid if we happened to prespecify the correct topology among the candidates, with the most promising being chosen via cross-validation.

While MIP-based formulations provide enough flexibility to seek optimal (sparse) orthogonal DTs, the dependence of their number of decision variables on the size of the training set is a recurring theme that limits their applicability to small datasets. More importantly, the MIP methods we reviewed do not solve for sparsity, except (Bertsimas and Dunn, 2017), and they fail at retrieving the true optimal sparse DT for MONK1, an important illustrative baseline. In the next section, we explore other approaches based on Satisfiability (SAT) (Biere et al., 2009).

2.3.3 Satisfiability (SAT)

Given a fixed DT topology, Bessiere et al. (2009) formulate a SAT problem for seeking the DT that perfectly classifies a dataset $\mathcal{D} = \{(X_i, Y_i)\}_{i=1}^{n}$. More specifically, the authors consider a binary classification problem with binary features. Let $\mathcal{E} = \{X_t\}_{t=1}^{n}$ be the set of examples in $\mathcal{D}$, with $\mathcal{E}^-$ and $\mathcal{E}^+ = \mathcal{E} \setminus \mathcal{E}^-$ the subsets of negatively and positively labeled examples respectively. Each example X_i is a vector of Boolean valuations of features from the set of features $\mathcal{F} = \{1, \dots, q\}$. For any $f \in \mathcal{F}$, $X[f]$ denotes the valuation of feature f at example X. Bessiere et al. (2009) then define a DT structure as a tuple $T = (\mathcal{N}, \mathcal{U}, r)$ of:

- $\mathcal{N}$ the set of nodes constituting the structure.
- $\mathcal{U}$ the set of edges, $(u, v) \in \mathcal{U}$ denotes the edge from the parent node u to its child v.
- $r \in \mathcal{N}$ the root node.
- $\mathcal{L} \subset \mathcal{N}$ is the set of leaves of the structure.

Note that we have modified some of the authors' notation to make it harmonious with ours.

A concrete DT based on the structure T is then defined via labeling each internal node $u \in \mathcal{N} \setminus \mathcal{L}$ with a feature $f(u) \in \mathcal{F}$. For any edge $(u, v) \in \mathcal{U}$, let $g(u, v) = \mathbb{1}\{v \text{ is the right child of } u\}$. This function helps assigning each example $X \in \mathcal{E}$ to a unique leaf $l(X) \in \mathcal{L}$ as follows: every edge (u, v) in the path $p(l(X))$, from r to $l(X)$, satisfies $X[f(u)] = g(u, v)$. In their SAT formulation, Bessiere et al. (2009) look for the DT, of structure T, that classifies dataset $\mathcal{E}$, i.e. such that:

$$\forall (X_i, X_j) \in \mathcal{E}^+ \times \mathcal{E}^- : l(X_i) \neq l(X_j)$$

To achieve this, Bessiere et al. (2009) introduce the literals:

$$\forall u \in \mathcal{N} \setminus \mathcal{L}, f \in \mathcal{F} : t_{uf} = \mathbb{1}\{f(u) = f\}$$

[1] This topology is taken from (Günlük et al., 2021, Fig. 3)

Then the classification of $\mathcal{E}$ condition translates to:

$$\forall (X_i, X_j) \in \mathcal{E}^+ \times \mathcal{E}^-, \forall l \in \mathcal{L} : \left\{ \bigvee_{(u,v)\in p(l), f\in \text{eq}(X_i,X_j) | g(u,v)\neq X_i[f]} t_{uf} \right\} \vee \left\{ \bigvee_{(u,v)\in p(l), f\in \mathcal{F}\setminus \text{eq}(X_i,X_j)} t_{uf} \right\}$$

$$\forall u \in \mathcal{N} \setminus \mathcal{L}, \forall f, f' \in \mathcal{F} : \left\{ \neg t_{uf} \vee \neg t_{uf'} \right\}$$

$$\forall l \in \mathcal{L}, \forall f \in \mathcal{F} : \bigwedge_{(u,v)\in p(l),(u',v')\in p(l), u\neq u'} \left\{ \neg t_{uf} \vee \neg t_{u'f} \right\}$$

where $\text{eq}(X_i, X_j) = \{f \in \mathcal{F} | X_i[f] = X_j[f]\}$ is the set of features where examples X_i and X_j agree. For any $(X_i, X_j) \in \mathcal{E}^+ \times \mathcal{E}^-$, the first clause prohibits any leaf from containing both X_i and X_j. Indeed, the left side of the first clause describes leaves that do not include X_i nor X_j, while the right side describes leaves l for which the path includes a node u labelled with a feature $f(u) \in \mathcal{F} \setminus \text{eq}(X_i, X_j)$ where X_i and X_j disagree, and therefore l cannot include both X_i and X_j. The second clause ensures that each node is labelled with at most one feature, and the third clause avoids the case where two nodes on the same path share the same feature. The main takeaway from this formulation is that, as noted by Bessiere et al. (2009) themselves, the space complexity (number of literals) of this SAT formulation is $\mathcal{O}(qN^2n^2 + Nq^2 + qN^3)$ where $|\mathcal{N}| = N, |\mathcal{F}| = q, |\mathcal{E}| = n$, which is very large and unfeasible in most real world experiments. In fact, the authors only test the SAT approach, which they solve with the SAT4J solver (Le Berre and Parrain, 2010), on the small datasets "Weather" and "Mouse" with less than 100 examples and for which DTs of only 9 and 15 total nodes (respectively) are fetched. In addition to the scalability issues, this SAT formulation does not address the main objective of the authors, it does not seek the smallest DT classifying $\mathcal{E}$, but rather a DT classifying $\mathcal{E}$ given a prespecified structure $T = (\mathcal{N}, \mathcal{U}, r)$. To alleviate these issues, Bessiere et al. (2009) also formulate a Constraint Program (CP), which we do not re-transcribe here, we refer the reader to (Bessiere et al., 2009, Sections 4 and 5) for further details. However, this CP model rather provides approximate solutions.

Narodytska et al. (2018) noticed the existence of structural properties of the DT learning problem that can be exploited to define a tighter SAT model. The objective set by the authors is to verify the existence of a DT, with a fixed number of internal nodes N, that classifies the training set $\mathcal{E}$. The difference with Bessiere et al. (2009) is that this approach only fixes the size N while allowing for any topology as long as it is consistent with a binary tree structure. Narodytska et al. (2018)'s approach consists of two steps:

- Given a fixed size N, guess a valid binary tree topology.
- For each of the proposed topologies, verify whether we can classify $\mathcal{E}$ or not.

The full SAT formulation can be found in (Narodytska et al., 2018, Sections 3.1 and 3.2). In addition to allowing for more flexible topologies, the main contribution of Narodytska et al. (2018) is to encode the constraint describing the classification of $\mathcal{E}$ in a way that does not necessitate a clause for each tuple $(X_i, X_j) \in \mathcal{E}^+ \times \mathcal{E}^-$. This provided a much tighter SAT model than Bessiere et al. (2009)'s, with a space complexity of only $\mathcal{O}(nNq + qN^2)$ instead of $\mathcal{O}(qN^2n^2 + Nq^2 + qN^3)$. In practice, lowering the dependence on the training set size n from a quadratic dependence to a linear one makes this new SAT model applicable to larger datasets, as is also confirmed empirically in (Narodytska et al., 2018, Table 3). Yet despite this significant contribution, the linear dependence on the training set size n is still a cause for serious scalability issues, as we saw earlier with Continuous Optimisation and MIP methods. Moreover, a major drawback of the SAT approach in general is its aim at finding the DT that perfectly classifies

the training set. In many cases, such objective leads to DTs overfitting the training set, and as a consequence, suffering from poor out-of-sample performance.

To avoid the overfitting issue while keeping a SAT-like formulation of the problem, Hu et al. (2020) propose a modification of Narodytska et al. (2018)'s model, where rather than seeking the DT of size N that perfectly classifies $\mathcal{E}$, the objective is is to find the DT of size N that maximises the number of correctly classified examples. While the two objectives coincide when there exists a DT that classifies $\mathcal{E}$ perfectly, the new objective allows DTs of smaller sizes that do not classify $\mathcal{E}$ perfectly to be considered. Hu et al. (2020)'s approach consists of modeling the problem within a weighted partial MaxSAT framework. Instead of seeking an assignment of the Boolean variables that satisfies all the clauses (as is the case in a regular SAT problem), the MaxSAT framework rather separates the clauses into *hard* clauses, which have to be satisfied, and *soft* clauses, which can be violated. A positive weight is then assigned to each soft clause, and the MaxSAT problem is now defined as seeking the assignment of the Boolean variables that satisfies all the hard clauses while maximising the weighted sum of the satisfied soft clauses. By introducing variables $b_q = \mathbb{1}\{X_q \text{ is correctly classified}\}$, Hu et al. (2020) modify Narodytska et al. (2018)'s SAT formulation into a MaxSAT formulation while avoiding quadratic dependencies on the training set size n incurred in (Bessiere et al., 2009). Nevertheless, the linear dependence on n still constitutes a major scalability issue.

2.3.4 Dynamic Programming and Branch & Bound

We have seen that the Mathematical Programming methods, from Continuous Optimisation to MIP and SAT, suffer from severe scalability issues that limit their applicability to relatively small datasets and shallow DTs. Even more importantly in our opinion, most of these methods do not solve for sparsity (as per the definition in Section 2.1), they rather fix a topology and optimise its internal splits and leaf predictions. Exceptions such as (Bertsimas and Dunn, 2017; Narodytska et al., 2018; Hu et al., 2020) do not completely address this limitation. We have seen that OCT (Bertsimas and Dunn, 2017) fails to retrieve the optimal sparse DT for MONK1. As for (Narodytska et al., 2018; Hu et al., 2020), despite allowing more flexibility when it comes to the DT topology, the problem they seek to solve is predicated upon fixing the total number of nodes of the DT.

To alleviate these issues, recent approaches based on Dynamic Programming (DP) and Branch & Bound (B&B) have emerged. These methods offered the first practical algorithms for retrieving optimal (and sometimes sparse) DTs in reasonable times, they are becoming more and more popular and have the potential to trigger a paradigm shift for DT optimisation. We review these methods in this section.

We first review a fully DP-based algorithm called Decision Trees from Lattices (DL8) that was developed and analysed by Nijssen and Fromont (2007, 2010). DL8 is an exact algorithm that can be used to find optimal sparse DTs satisfying a certain set of constraints, its main idea is to relate DT learning to the domain of itemset mining (Agrawal et al., 1996; Han et al., 2000; Zaki et al., 1997). We consider again the classification problem introduced in Section 2.1. DL8 operates on binary features, thus:

$$\forall 1 \leq i \leq n, 1 \leq j \leq q : X_i^{(j)} \in \{0, 1\}$$

The approach consists of transforming the set of inputs (examples) $\mathcal{E} = \{X_i\}_{i=1}^n$ into a transac-

A	B	C	Y
0	1	1	0
1	0	1	1
0	0	1	1
0	1	0	0
1	0	0	1
0	0	0	0
0	0	1	0
0	1	0	0
1	1	0	1
0	0	0	1
0	0	1	0
0	0	0	1

Itemset	Y
$\neg a, b, c$	0
$a, \neg b, c$	1
$\neg a, \neg b, c$	1
$\neg a, b, \neg c$	0
$a, \neg b, \neg c$	1
$\neg a, \neg b, \neg c$	0
$\neg a, \neg b, c$	0
$\neg a, b, \neg c$	0
$a, b, \neg c$	1
$\neg a, \neg b, \neg c$	1
$\neg a, \neg b, c$	0
$\neg a, \neg b, \neg c$	1

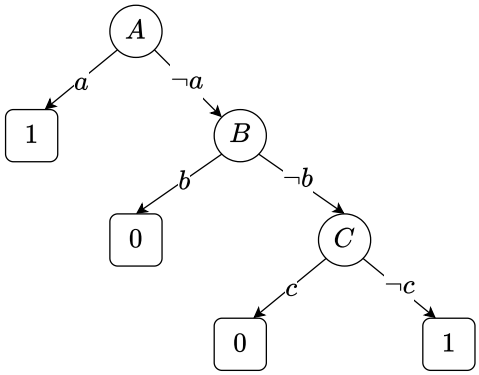


Figure 2.2: The left-most table is a set of binary inputs $\mathcal{E}$ that is transformed into the transactional database $\mathcal{T}$ in the middle. The DT drawn here is the optimal DT for this example viewed from the transactional lens.

tional database $\mathcal{T} = \{t_i\}_{i=1}^{n}$ such that:

$$\forall 1 \leq i \leq n : t_i = \{j |\ 1 \leq j \leq q, X_i^{(j)} = 1\} \cup \{\neg j |\ 1 \leq j \leq q, X_i^{(j)} = 0\}$$

Fig. 2.2 illustrates an example of this transformation[2]. In this example, features A, B and C yield the set of items $\mathcal{I} = \{a, \neg a, b, \neg b, c, \neg c\}$ that are used to construct $\mathcal{T}$. The set of all possible itemsets, i.e. the set of all subsets of $\mathcal{I}$, forms a lattice that we illustrate in Fig. 2.3[3]. Every possible DT is encoded within the lattice as a subgraph, granted it satisfies the following properties:

- The subgraph is rooted at the root of the lattice, the empty node.
- Every path I (that starts from the root) in the subgraph is either contained in no other path, or is contained in exactly two paths $I \cup \{i\}$ and $I \cup \{\neg i\}$ where $i \notin I$ and $\neg i \notin I$.

Therefore the optimal DT satisfying some form of constraints (or no constraints at all) is also encoded within the lattice. The core idea of DL8 is to traverse the lattice in a Depth-First-Search (DFS) (Tarjan, 1971) fashion looking for the optimal DT. This recursive exploration strategy is feasible because the problem can be decomposed into subproblems at the level of each node. Indeed, for the optimal DT, each subtree rooted at one of its internal nodes t is itself the optimal subtree among all subtrees rooted at t regardless of the other branches[4]. The DP component of DL8 resides in its symmetry-aware representation of itemsets, for example, itemset $\{a, \neg b\}$ is represented with the same node in the lattice regardless of whether we reach it as a child of node $\{a\}$ or node $\{\neg b\}$. This symmetry-awareness allows the reuse of information during the DFS traversal of the lattice. However, DL8 can be computationally very costly as the size of the lattice (its number of nodes) grows exponentially with the number of features. If q is the number of features, then $2q$ is the number of items $|\mathcal{I}|$, and since the size of the lattice is the number of all possible itemsets that can be constructed from $\mathcal{I}$, then the size of the lattice is 2^{2q}. This high complexity renders DL8 impractical for many real world applications.

[2]This example is taken from (Aglin et al., 2020).
[3]This same lattice can be found in (Nijssen and Fromont, 2007, 2010; Aglin et al., 2020).
[4]This is true for additive objective functions like accuracy.

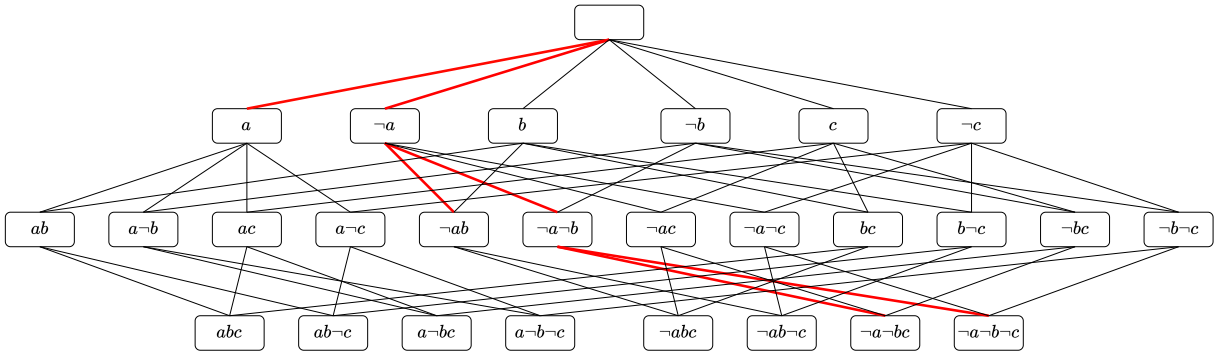


Figure 2.3: Lattice corresponding to the set of items $\{a, \neg a, b, \neg b, c, \neg c\}$. The optimal DT for the example in Fig. 2.2 is represented in red.

A decade later, Aglin et al. (2020) reconsidered the idea of mining optimal DTs from itemset lattices, and designed an improved version of DL8 called DL8.5. In addition to the DP component, this new algorithm adds a B&B constituent to the DFS strategy, which in many cases permits the pruning of large areas from the lattice. Concretely, once the first subproblem has been solved, i.e. once we hit a leaf of the lattice or we reach a node at which we prune the lattice, the evaluation of the node, e.g. its misclassification error, serves as an upper bound to prune the lattice by disqualifying unpromising nodes (in other regions of the lattice) from being explored. Moreover, this upper bound is updated as we find better solutions. DL8.5 displays practical runtimes on a broader range of datasets compared to DL8 while simultaneously keeping the possibility of defining constraints on the depths of DTs and the minimum support size (number of data per leaf). Nonetheless, DL8.5 has its own limitations. While DL8 offers enough flexibility to solve for a form of sparsity, as evident by the lexicographical objective $\text{Argmin}_T\{\text{error}(T), \text{size}(T)\}$ defined in (Nijssen and Fromont, 2010, Section 2.3), DL8.5 on the other hand does not solve for sparsity. When an upper bound has been set by the solution of a subproblem, it can prune another solution with the same misclassification error even if it is of smaller size, and there is no trivial way around this phenomenon since we do not know the size of this potential solution beforehand, and DL8.5 prunes it pre-emptively as part of its B&B strategy. In fact, this is evident from the failure of DL8.5 at retrieving the optimal sparse DT for MONK1, albeit the authors only ran it with a maximum depth constraint of 2. Nevertheless, even when setting the optimal depth to 4, DL8.5 can only retrieve a solution with 100% accuracy, it fails at finding the optimal sparse DT with minimum number of nodes.

A breakthrough that incorporates solving for sparsity was achieved with the Optimal Sparse Decision Trees (OSDT) algorithm Hu et al. (2019). OSDT also employs a combination of DP and B&B, but seeks to minimise a regularised empirical risk function that penalises DTs with many leaves. Moreover, OSDT's B&B strategy is more efficient than DL8.5's as it employs a series of specialised analytical bounds that prune the search space without the need for solving subproblems first. OSDT operates on classification problems with binary features as well:

$$\forall 1 \leq i \leq n, 1 \leq j \leq q : X_i^{(j)} \in \{0, 1\}$$

Hu et al. (2019) model a DT of length $H > 0$ as a tuple of leaves $d = (l_1, \ldots, l_H)$. We chose to describe each leaf l_k as a subset of the input (features) space $\mathcal{X} = \{0, 1\}^q$, so that $\mathbb{1}\{X_i \in l_k\}$ is a Boolean variable that valuates to 1 if and only if X_i is in the subset of $\mathcal{X}$ represented by l_k. Each leaf l_k is assigned a label $\widehat{Y}_k^{(\text{leaf})}$. Furthermore, Hu et al. (2019) complete the definition of

DT d via partitioning its leaves as follows $d = (d_{\text{un}}, d_{\text{split}}, K, H)$[5] where $d_{\text{un}} = (l_1, \ldots, l_K)$ is the tuple of unchanged leaves and $d_{\text{split}} = (l_{K+1}, \ldots, l_H)$ is the set of leaves that will be split. The misclassification error of DT d is then:

$$\mathcal{L}(d, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{H} \mathbb{1}\{X_i \in l_k\} \mathbb{1}\{\widehat{Y}_k^{(\text{leaf})} = Y_k\}$$

The objective is not to minimise $\mathcal{L}(d, \mathcal{D})$ but rather a regularised version that the authors call the regularised empirical risk, where the number of leaves is penalised:

$$\mathcal{R}(d, \mathcal{D}) = \mathcal{L}(d, \mathcal{D}) + \lambda H$$

$\lambda > 0$ is a penalty parameter. For binary trees, the number of leaves is always equal to the number of internal splits +1, hence the solution to optimising $\mathcal{R}$ coincides with the solution to optimising Bertsimas and Dunn (2017)'s loss function. A hierarchical structure between DTs is also defined, where $d' = \left(d'_{\text{un}}, d'_{\text{split}}, K', H'\right)$ is a child of $d = (d_{\text{un}}, d_{\text{split}}, K, H)$ if and only if $H' = H + 1, d \subset d', d_{\text{un}} \subset d'_{\text{un}}$, this structure is the first step towards employing B&B. The second step involves finding a lower bound on the regularised empirical risk of all the descendants of a DT. Such lower bound is provided in (Hu et al., 2019, Theorem 3.1) where it is called the Hierarchical objective lower bound:

Let $d = (d_{\text{un}}, d_{\text{split}}, K, H)$ be a DT and define the lower bound:

$$b(d_{\text{un}}, \mathcal{D}) = \underbrace{\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{X_i \in l_k\} \mathbb{1}\{\widehat{y}_k^{(\text{leaf})} = Y_k\}}_{\text{Misclassification error due to } d_{\text{un}}} + \lambda H$$

Then for any descendent $d' = \left(d'_{\text{un}}, d'_{\text{split}}, K', H'\right)$ of d, i.e. $H' \geq H, d \subset d', d_{\text{un}} \subset d'_{\text{un}}$:

$$b(d_{\text{un}}, \mathcal{D}) \leq \mathcal{R}(d', \mathcal{D})$$

Further auxiliary bounds are provided by Hu et al. (2019). The allure of this approach compared to DL8.5 is that it does not necessitate solving subproblems first, its analytical bounds can potentially prune more parts of the search space, and it solves for sparsity. In fact, OSDT is the first algorithm we came across that finds the true optimal sparse DT for MONK1 (Hu et al., 2019, Figure 6). On the other hand, OSDT displays the following drawbacks:

- For a set of leaves $d = (l_1, \ldots, l_H)$, the DT $(\emptyset, d, 0, H)$ always has the smallest bound among all trees that share the same leaves d. This rigidity makes a scheduling policy based on the lower bound $b(d_{\text{un}}, \mathcal{D})$ always choose $(\emptyset, d, 0, H)$ before the other DTs sharing leaves d, even if the algorithm's resources could be better diverted towards exploring one of those other DTs. This might be why the authors notice a better empirical performance when using the curiosity scheduling policy from (Angelino et al., 2018).
- The bound on the total number of evaluated DTs, derived in (Hu et al., 2019, Corollary D.2), does not reflect the complexity of the algorithm as the number of DT evaluations could be higher than the total number leaf evaluations. Indeed, let $d^1 = \left(d^1_{\text{un}}, d^1_{\text{split}}, K_1, H\right)$

[5] We opted for a more compact notation than $d = (d_{\text{un}}, \delta_{\text{un}}, d_{\text{split}}, \delta_{\text{split}}, K, H)$, employed by the authors, as we will not provide OSDT's pseudocode.

and $d^1 = \left(d^2_{\text{un}}, d^2_{\text{split}}, K_2, H\right)$ be two different DTs sharing the same leaves, i.e.:

$$d = d^1_{\text{un}} \cup d^1_{\text{split}} = d^2_{\text{un}} \cup d^2_{\text{split}};\ d^1_{\text{un}} \subset d^2_{\text{un}};\ K_1 < K_2$$

These DTs are created at the same time according to (Hu et al., 2019, Algorithm 1). Even though they share the same leaves and thus the same regularised empirical risk $\mathcal{R}(d, \mathcal{D})$, their bounds $b\left(d^1_{\text{un}}, \mathcal{D}\right)$ and $b\left(d^2_{\text{un}}, \mathcal{D}\right)$ are calculated differently. Thus in order to assess OSDT's complexity, it does not suffice to bound the number of possible leaves that will not be pruned by the algorithm because for the same leaves, OSDT considers multiple different DTs, and they will be extracted from the queue and treated at different iterations.

- OSDT is significantly slower than DL8.5 on many settings.
- OSDT's search space is the space of DTs. The corresponding DP procedure does not allow for fine-grained information reuse at the same level as the lattice of itemsets allows.

OSDT's approach marks a breakthrough in DT learning, as such, it was revisited by Lin et al. (2020) and generalised to handling a broad range of objective functions including weighted accuracy, balanced accuracy, F-score, AUC and partial area under the ROC convex hull. The resulting algorithm, called Generalized and Scalable Optimal Sparse Decision Trees (GOSDT), employs a different leaf representation from OSDT and DL8.5. While the latter algorithms represent leaves as conjunctions of assertions (on features), GOSDT rather represents each leaf with the subset of the training set $\mathcal{D}$ it captures, which is called the support set. This allows further reuse of information via Dynamic Programming (DP).

More improvements to these methods have been considered recently, with Demirović et al. (2022)'s MurTree algorithm. MurTree employs a DFS strategy similarly to DL8.5 and contrary to OSDT and GOSDT, which are BFS (Best-First-Search) algorithms. MurTree generalises DL8.5's constraints to handling both the maximum depth and the maximum number of nodes, while also improving it computationally via a more efficient technique for handling DTs of depth at most 2 and introducing a new similarity-based lower bound to further prune the search space. This leads MurTree to display better execution times than DL8.5. While MurTree can be conceptually extended to solving for sparsity as explained by the authors in (Demirović et al., 2022, sections 4.8.3 and 4.8.4), our experiments with the implementation in `https://github.com/jurra/pymurtree.git` will showcase a suboptimal behaviour when handling the regularised objective. Moreover, it is unclear how much of the search space is pruned with MurTree's techniques and what MurTree's overall complexity is. In fact, similarly to DL8.5, MurTree needs to first solve a subproblem before its bounds could start pruning the search space. As such, in a worst-case scenario, the algorithm may only prune small parts of the search space and behave closer to an exhaustive search.

Additional recent advancements in the field include (McTavish et al., 2022), which introduce a guessing strategy to navigate the search space, seeking solutions with performance akin to a reference ensemble model. Demirović et al. (2023) improve on DL8.5 via a different search strategy that prioritizes expanding non-terminal branches before optimising grown branches, thus inducing an anytime algorithm called Blossom. van der Linden et al. (2024) investigate separable objectives and constraints and introduce a generalised DP framework called STreeD which solves for sparsity.

Before concluding this section, we categorise optimal DT methods in the DP and B&B literature into two categories. Depth-First-Search (DFS) algorithms like DL8.5, MurTree and

STreeD, and Best-First-Search (BFS) algorithms like OSDT and GOSDT. We summarise below the benefits and drawbacks of these methods:

- BFS methods are specifically designed to solve for sparsity. As such, if sparsity is of no concern in the considered problem, DFS methods perform well and fast, and the benefits of employing BFS over DFS are unclear.
- When solving for sparsity, BFS methods employ analytical bounds in their B&B strategy, which prune the search space more aggressively than DFS methods. Moreover, BFS is an informed way of exploring the search space by prioritising promising regions, this generally leads to faster execution.
- The informed strategy of BFS comes at the cost of higher memory consumption compared to DFS methods. In a worst-case scenario with a large dataset and many features, BFS displays a higher risk to run out of memory than DFS. However, in such challenging cases, DFS algorithms are very likely to run out of time. This is a global scalability issue that plagues optimal sparse DT methods.
- BFS methods have a straightforward anytime behaviour. This means that even if they run out of time before achieving optimality, they can still propose a solution, albeit with no guarantee on quality. This is less clear in the DFS context. For example, STreeD (as of v1.3.1) does not enjoy the anytime behaviour.

For an excellent discussion on the differences between DFS and BFS, refer to (Pearl, 1984, Chapter 2), a brief summary is provided in (Pearl, 1984, Section 2.5). Because of the aggressive pruning and the informed search strategy of BFS, we think that this approach has a more promising potential in the context of solving for sparsity. In Chapter 4 we will introduce our new BFS algorithm called Branches, which is the main contribution of this thesis.

2.4 Online Decision Trees

As we explained in Section 2.1, in Online Learning, a fixed dataset $\mathcal{D}$ is no longer available. Instead, we observe data incrementally as they arrive through a stream. As a consequence, the algorithms we reviewed in Section 2.2 and Section 2.3 are no longer applicable due to the following points:

- The greedy heuristics calculate the gain of each split based on the subset of $\mathcal{D}$ contained in the leaf we consider for splitting.
- The Mathematical Programming methods have rigid formulations of their mathematical programs with respect to $\mathcal{D}$. It is unclear how to extend such formulations to handling data streams.
- The DP and B&B methods are also rigid with respect to $\mathcal{D}$, some more than others. For example, GOSDT caches branches based on their support sets (the subset of data they contain). This is incompatible with data streams because these support sets are not fixed but change with time. Moreover, the pruning bounds of the B&B strategies are calculated on $\mathcal{D}$.

Nevertheless, attempts have been made to adapt DT algorithms to handling data streams, albeit, to the best of our knowledge, most of these attempts focus on greedy heuristics rather than optimal sparse DTs. The most popular algorithm of this sort is VFDT (Domingos and Hulten, 2000), it constructs online DTs called Hoeffding Trees in a top-down fashion, similar

to ID3, C4.5, and CART. VFDT stores estimates of the gains of the features in each leaf and chooses the feature that maximizes the expected gain with high probability. Consider the online classification problem introduced in Section 2.2, where (X_t, Y_t) is the observed data point at time t. At each leaf l of a Hoeffding Tree, VFDT stores the following running statistics that depend on the current time t:

$$\forall 1 \leq i \leq q, 1 \leq j \leq C_i, 1 \leq k \leq K : n_{ijk}(l) = \sum_{t'=1}^{t} \mathbb{1}\{X_{t'} \in l\}\mathbb{1}\{X_{t'}^{(i)} = j\}\mathbb{1}\{Y_{t'} = k\}$$

$$n_k(l) = \sum_{i=1}^{q}\sum_{j=1}^{C_i} n_{ijk}(l)$$

$$n(l) = \sum_{k=1}^{K} n_k(l)$$

$n_{ijk}(l)$ is the number of observed data points in l of class k such that the i^{th} feature is equal to j. The gain of splitting l with respect to feature $X^{(i)}$ is estimated with:

$$\widehat{\text{Gain}}\left(l, X^{(i)}\right) = \widehat{\mathcal{G}}(l) - \sum_{j=1}^{C_i} \frac{n(l_j)}{n(l)} \widehat{\mathcal{G}}(l_j)$$

where $\widehat{\mathcal{G}}(l)$ is the current estimated impurity (Entropy or Gini) using the $n_{ijk}(l)$ statistics. Without loss of generality, let $X^{(1)}$ and $X^{(2)}$ be the features with highest and second highest estimated gains respectively and define the gain difference between them as:

$$\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right) = \widehat{\text{Gain}}\left(l, X^{(1)}\right) - \widehat{\text{Gain}}\left(l, X^{(2)}\right) \geq 0$$

Domingos and Hulten (2000) claim that $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)$ satisfies the following Hoeffding inequality (Hoeffding, 1963) for any $0 < \delta < 1$:

$$\begin{cases} \mathbb{P}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right) - \mathbb{E}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)\right] \leq \epsilon(\delta, n(l))\right] \geq 1 - \delta \\ \epsilon(\delta, n(l)) = C\sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2n(l)}} \quad \text{where } C > 0 \end{cases}$$

According to this claim, if at some point we get $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right) > \epsilon(\delta, n(l))$, it would mean that we are at least $1-\delta$ confident that $\mathbb{E}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)\right] > 0$, thus motivating us to split l with respect to $X^{(1)}$.

VFDT gained immense popularity within the Data Stream Mining community and laid the foundation for subsequent developments (Hulten et al., 2001; Bifet and Gavalda, 2009; Manapragada et al., 2018; Gouk et al., 2019). Yet despite this success, a careful examination by Rutkowski et al. (2012) of the employed Hoeffding's inequality reveals an inadequacy. Indeed, $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)$ is not a sum of independent bounded variables, and as such, it violates the assumptions of Hoeffding's inequality. This is not to say that $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)$ satisfies no concentration inequalities whatsoever, but rather that such a concentration result should be carefully re-examined in light of the properties of $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)$. This led Rutkowski et al. (2012) to discover that $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)$ has bounded differences, and as a consequence, it

satisfies McDiarmid's inequality (McDiarmid et al., 1989). (Rutkowski et al., 2012, Theorem 1) retrieves the McDiarmid-based concentration inequality that is satisfied by the Entropy:

$$\begin{cases} \mathbb{P}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right) - \mathbb{E}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)\right] \leq \epsilon\left(\delta, n\left(l\right)\right)\right] \geq 1-\delta \\ \epsilon\left(\delta, n\left(l\right)\right) = \left[6\left(K\log\left(en\left(l\right)+\log\left(2n\left(l\right)\right)\right)\right)+2\log K\right]\sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2n(l)}} \end{cases}$$

where e is Euler's number. A similar concentration result was also derived for Gini in (Rutkowski et al., 2012, Theorem 2):

$$\begin{cases} \mathbb{P}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right) - \mathbb{E}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)\right] \leq \epsilon\left(\delta, n\left(l\right)\right)\right] \geq 1-\delta \\ \epsilon\left(\delta, n\left(l\right)\right) = 8\sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2n(l)}} \end{cases}$$

While the McDiarmid and Hoeffding bounds $\epsilon\left(\delta, n\left(l\right)\right)$ are similar for Gini, in the sense that they are both $\mathcal{O}\left(\sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2n(l)}}\right)$, this is clearly not the case for Entropy, where the McDiarmid bound is $\mathcal{O}\left(\log\left(n\left(l\right)\right)\sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2n(l)}}\right)$ and thus larger.

Despite shedding light on the inconsistencies of Hoeffding Trees, the McDiarmid-based approach remains very conservative because of its loose bounds, especially for Entropy. The looseness of these bounds is due to the generality of McDiarmid's inequality, which assumes nothing about the probability distributions of the involved variables except that they are independent and that $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)$ has bounded differences. Furthermore, there is an additional issue with these concentration inequalities that does not seem to be discussed in the literature to the best of our knowledge. It pertains to the fact that they consider the concentration of $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)$ around its expectation. However, the main objective of a split is rather to seek the feature with maximum gain with respect to the true impurity:

$$\text{Gain}\left(l, X^{(1)}\right) = \mathcal{G}\left(l\right) - \sum_{j=1}^{C_1} p\left(l_j\right)\mathcal{G}\left(l_j\right)$$

where $p\left(l\right) = \mathbb{P}\left[X \in l\right]$ is the probability of $X \in l$ and $\mathcal{G}\left(l\right)$ is the true Entropy or Gini, i.e.:

- **Entropy:** $\mathcal{G}\left(l\right) = -\sum_{k=1}^{K} p_k\left(l\right)\log\left(p_k\left(l\right)\right)$
- **Gini impurity:** $\mathcal{G}\left(l\right) = 1 - \sum_{k=1}^{K}\left(p_k\left(l\right)\right)^2$

where $p_k\left(l\right) = \mathbb{P}\left[Y = k | X \in l\right]$ is the probability that a data point in l is of class k. On the other hand, the expected estimated gain satisfies the following:

$$\mathbb{E}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)\right] = \mathbb{E}\left[\sum_{j=1}^{C_2}\frac{n\left(l_j\right)}{n\left(l\right)}\widehat{\mathcal{G}}\left(l_j\right)\right] - \mathbb{E}\left[\sum_{j=1}^{C_1}\frac{n\left(l_j\right)}{n\left(l\right)}\widehat{\mathcal{G}}\left(l_j\right)\right]$$

However, Gini and Entropy are concave, and thus by virtue of Jensen's inequality, $\widehat{\mathcal{G}}\left(l\right)$ is biased and so is $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)$, see (Basharin, 1959; Harris, 1975) for an analysis of the Entropy bias. As a consequence, a feature maximising the expected gain does not necessarily maximise

the gain with respect to the true Entropy or Gini impurity.

As discussed earlier, the bounds induced by both Hoeffding and McDiarmid make no assumptions about the probability distribution of the gain function, as such these bounds can be very conservative. Thus, the question of whether we can find tighter bounds naturally arose, leading Jin and Agrawal (2003) to propose a Gaussian approximation of the gain based on the Multivariate Delta Method. Jin and Agrawal (2003) consider a binary classification problem with binary DTs and rewrite the gain as the following function:

$$\begin{aligned}\text{Gain}\left(l, X^{(1)}\right) &= \text{Gain}\left(p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}\right) \\ &= \mathcal{G}\left(p_{1L}^{(1)} p_L^{(1)} + p_{1R}^{(1)}\left(1 - p_L^{(1)}\right)\right) - p_L^{(1)} \mathcal{G}\left(p_{1L}^{(1)}\right) - \left(1 - p_L^{(1)}\right) \mathcal{G}\left(p_{1R}^{(1)}\right)\end{aligned}$$

- $p_L^{(1)} = \mathbb{P}\left[X^{(1)} = 1 \middle| X \in l\right]$ is the probability that a data point in leaf l goes to the left child corresponding to $X^{(1)} = 1$.
- $1 - p_L^{(1)} = \mathbb{P}\left[X^{(1)} = 2 \middle| X \in l\right]$ is the probability that a data point in leaf l goes to the right child corresponding to $X^{(1)} = 2$.
- $p_{1L}^{(1)} = \mathbb{P}\left[Y = 1 | X \in l, X^{(1)} = 1\right]$ is the probability that a data point in the left child has class $Y = 1$.
- $p_{1R}^{(1)} = \mathbb{P}\left[Y = 1 | X \in l, X^{(1)} = 2\right]$ is the probability that a data point in the right child has class $Y = 1$.
- **Entropy:** $\mathcal{G}\left(p_{1L}^{(1)}\right) = p_{1L}^{(1)} \log\left(p_{1L}^{(1)}\right) + \left(1 - p_{1L}^{(1)}\right) \log\left(1 - p_{1L}^{(1)}\right)$
- **Gini:** $\mathcal{G}\left(p_{1L}^{(1)}\right) = 1 - \left(p_{1L}^{(1)}\right)^2 - \left(1 - p_{1L}^{(1)}\right)^2$

(Jin and Agrawal, 2003, Lemma 2) uses the Multivariate Delta Method to derive the following result:

$$\begin{cases} \widehat{\text{Gain}}\left(l, X^{(1)}\right) = \text{Gain}\left(\widehat{p}_L^{(1)}, \widehat{p}_{1L}^{(1)}, \widehat{p}_{1R}^{(1)}\right) \longrightarrow \mathcal{N}\left(\text{Gain}\left(l, X^{(1)}\right), \frac{\left(\tau^{(1)}\right)^2}{n(l)}\right) \\ \left(\tau^{(1)}\right)^2 = \left(\frac{\partial \text{Gain}}{\partial p_L^{(1)}}\right)^2 p_L^{(1)}\left(1 - p_L^{(1)}\right) + \left(\frac{\partial \text{Gain}}{\partial p_{1L}^{(1)}}\right)^2 p_{1L}^{(1)}\left(1 - p_{1L}^{(1)}\right) + \left(\frac{\partial \text{Gain}}{\partial p_{1R}^{(1)}}\right)^2 p_{1R}^{(1)}\left(1 - p_{1R}^{(1)}\right) \end{cases}$$

where $\widehat{p}_L^{(1)}, \widehat{p}_{1L}^{(1)}, \widehat{p}_{1R}^{(1)}$ are the empirical average estimates of $p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}$ respectively, the derivatives are evaluated at $\left(p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}\right)$, and the convergence is in distribution as $n(l)$ grows to infinity.

This result is then used to provide a concentration inequality for $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)$ via a Gaussian approximation as follows:

$$\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right) \longrightarrow \mathcal{N}\left(\Delta\text{Gain}\left(X^{(1)}, X^{(2)}\right), \frac{\left(\tau^{(1)}\right)^2 + \left(\tau^{(2)}\right)^2}{n(l)}\right)$$

$$\implies \begin{cases} \mathbb{P}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right) - \mathbb{E}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)\right] \leq \epsilon\left(\delta, n(l)\right)\right] \geq 1 - \delta \\ \epsilon\left(\delta, n(l)\right) = z_{1-\delta}\sqrt{\frac{\left(\tau^{(1)}\right)^2 + \left(\tau^{(2)}\right)^2}{n(l)}} \end{cases}$$

where $z_{1-\delta}$ is the $(1-\delta)^{\text{th}}$ quantile of the standard Normal distribution. In (Jin and Agrawal, 2003, Theorem 4), the authors claim that their induced bound is tighter than Hoeffding-style bounds. However, to our knowledge, no proof was provided.

Perhaps the most interesting result of this paper is that it provides a theoretically sound approach to deriving concentration inequalities on the gain measure. Moreover, the induced bound for Entropy is $\mathcal{O}\left(z_{1-\delta}\sqrt{\frac{1}{n(l)}}\right)$, which is better than the McDiarmid induced bound that is $\mathcal{O}\left(\log\left(n\left(l\right)\right)\sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{n(l)}}\right)$. The contribution of this work goes beyond bound comparisons. Indeed, upon a closer look, we notice that the derived concentration result of $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)$ is with respect to the true gain $\Delta\text{Gain}\left(X^{(1)}, X^{(2)}\right)$ rather than the expected estimated gain $\mathbb{E}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)\right]$. Nonetheless, this approach suffers from the following limitations, $\tau^{(1)}$ depends on the true unknown probabilities $p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}$ and there is no trivial way of calculating $\tau^{(1)}$. In addition, the gain estimate $\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right) = \text{Gain}\left(\widehat{p}_L^{(1)}, \widehat{p}_{1L}^{(1)}, \widehat{p}_{1R}^{(1)}\right)$ does not satisfy the assumptions of the Multivariate Delta Method because $\widehat{p}_{1L}^{(1)}$ and $\widehat{p}_{1R}^{(1)}$ are not calculated with $n\left(l\right)$ samples like $\widehat{p}_L^{(1)}$. Rutkowski et al. (2013) noticed these inconsistencies and proposed an amelioration of the work, where rather than directly applying the Multivariate Delta Method, a first order Taylor expansion of the gain is first performed around $p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}$ yielding (Rutkowski et al., 2013, Lemma 2):

$$\begin{cases} \widehat{\text{Gain}}\left(l, X^{(1)}\right) = \text{Gain}\left(\widehat{p}_L^{(1)}, \widehat{p}_{1L}^{(1)}, \widehat{p}_{1R}^{(1)}\right) \longrightarrow \mathcal{N}\left(\text{Gain}\left(l, X^{(1)}\right), \frac{\left(\tau^{(1)}\right)^2}{n(l)}\right) \\ \left(\tau^{(1)}\right)^2 = \left(\tau_L^{(1)}\right)^2 + \frac{\left(\tau_{1L}^{(1)}\right)^2}{\widehat{p}_L^{(1)}} + \frac{\left(\tau_{1R}^{(1)}\right)^2}{1-\widehat{p}_L^{(1)}} \\ \left(\tau_L^{(1)}\right)^2 = \left(\frac{\partial \text{Gain}}{\partial p_L^{(1)}}\right)^2 p_L^{(1)}\left(1-p_L^{(1)}\right) \\ \left(\tau_{1L}^{(1)}\right)^2 = \left(\frac{\partial \text{Gain}}{\partial p_{1L}^{(1)}}\right)^2 p_{1L}^{(1)}\left(1-p_{1L}^{(1)}\right) \\ \left(\tau_{1R}^{(1)}\right)^2 = \left(\frac{\partial \text{Gain}}{\partial p_{1R}^{(1)}}\right)^2 p_{1R}^{(1)}\left(1-p_{1R}^{(1)}\right). \end{cases}$$

Where the partial derivatives are evaluated at $\left(p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}\right)$

Since $\tau^{(1)}$ depends on the unknown probabilities $p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}$, Rutkowski et al. (2013) propose an upper bound $\left(\tau^{(1)}\right)^2 \leq Q\left(C\right)$ that depends on a predefined threshold for $p_1\left(l\right) = \mathbb{P}\left[Y=1 | X \in l\right]$ under which there is no need to split leaf l. This leads to the paper's main result (Rutkowski et al., 2013, Theorem 1):

$$\begin{cases} \mathbb{P}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right) - \mathbb{E}\left[\Delta\widehat{\text{Gain}}\left(X^{(1)}, X^{(2)}\right)\right] \leq \epsilon\left(\delta, n\left(l\right)\right)\right] \geq 1-\delta \\ \epsilon\left(\delta, n\left(l\right)\right) = z_{1-\delta}\sqrt{\frac{2Q(C)}{n(l)}} \end{cases}$$

Rutkowski et al. (2013) derive this result for Entropy and show numerically how tighter this bound is compared to their earlier McDiarmid-based bound for Entropy. The authors likely did

not provide a similar analysis with Gini since its McDiarmid bound is already $\mathcal{O}\left(\sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{n(l)}}\right)$.

While this work constitutes an improvement on the earlier work by Jin and Agrawal (2003), it also overlooks an inadequacy when deriving the result of (Rutkowski et al., 2013, Lemma 2). Indeed, to show that:

$$\text{Gain}\left(\widehat{p}_L^{(1)}, \widehat{p}_{1L}^{(1)}, \widehat{p}_{1R}^{(1)}\right) \longrightarrow \mathcal{N}\left(\text{Gain}\left(l, X^{(1)}\right), \frac{\left(\tau^{(1)}\right)^2}{n\left(l\right)}\right)$$

The authors perform the following first order Taylor expansion of $\text{Gain}\left(\widehat{p}_L^{(1)}, \widehat{p}_{1L}^{(1)}, \widehat{p}_{1R}^{(1)}\right)$ around $\left(p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}\right)$:

$$\begin{aligned}\text{Gain}\left(\widehat{p}_L^{(1)}, \widehat{p}_{1L}^{(1)}, \widehat{p}_{1R}^{(1)}\right) = \text{Gain}\left(p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}\right) &+ \frac{\partial \text{Gain}\left(p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}\right)}{\partial p_L}\left(\widehat{p}_L^{(1)} - p_L^{(1)}\right) \\ &+ \frac{\partial \text{Gain}\left(p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}\right)}{\partial p_{1L}}\left(\widehat{p}_{1L}^{(1)} - p_{1L}^{(1)}\right) \\ &+ \frac{\partial \text{Gain}\left(p_L^{(1)}, p_{1L}^{(1)}, p_{1R}^{(1)}\right)}{\partial p_{1R}}\left(\widehat{p}_{1R}^{(1)} - p_{1R}^{(1)}\right)\end{aligned}$$

Then they proceed to make the following independent Gaussian approximations based on the Central Limit theorem:

$$\begin{aligned}&\widehat{p}_L^{(1)} \sim \mathcal{N}\left(p_L^{(1)}, \frac{p_L^{(1)}\left(1 - p_L^{(1)}\right)}{n\left(l\right)}\right), \ \widehat{p}_{1L}^{(1)} \sim \mathcal{N}\left(p_{1L}^{(1)}, \frac{p_{1L}^{(1)}\left(1 - p_{1L}^{(1)}\right)}{n_L^{(1)}\left(l\right)}\right) \\ &\widehat{p}_{1R}^{(1)} \sim \mathcal{N}\left(p_{1R}^{(1)}, \frac{p_{1R}^{(1)}\left(1 - p_{1R}^{(1)}\right)}{n_R^{(1)}\left(l\right)}\right)\end{aligned}$$

where $n_L^{(1)}\left(l\right)$ and $n_R^{(1)}\left(l\right)$ are the number of observed samples (from the $n\left(l\right)$ samples in l) in the left and right children respectively. Given a fixed large $n\left(l\right)$, the approximation of $\widehat{p}_L^{(1)}$ is justified in light of the Central Limit Theorem. However, this is not the case for the second approximation because we do not observe a fixed large number $n_L^{(1)}\left(l\right)$ of samples. $n_L^{(1)}\left(l\right)$ is a random variable that depends on $n\left(l\right)$ (the same remark is valid for the third approximation). Such approximation could be justified conditionally on a large number $n_L^{(1)}\left(l\right)$, but then we would still have to integrate over all the possible values of $n_L^{(1)}\left(l\right)$, including small values for which the conditional approximation is not justified.

The main framework we will use throughout this thesis to describe and derive our methods is Reinforcement Learning (Barto et al., 1995). In the next section, we introduce this framework and present an overview of DT methods that are based upon it.

2.5 Reinforcement Learning (RL)

RL is a framework describing the sequential interaction between an agent and an environment, where the agent seeks to maximise the rewards it gathers from the environment through a series

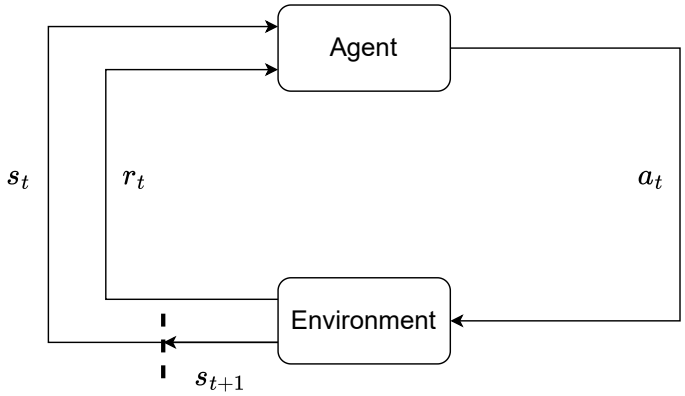


Figure 2.4: At time t, the agent is in a current state s_t, it performs an action a_t upon which it receives feedback from the environment in the form of a reward r_t and a next state s_{t+1}.

of actions. Fig. 2.4 illustrates the Agent-Environment dynamics. Formally, a RL problem is usually defined within the confines of a Markov Decision Process (MDP), which is a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ of the following notions:

- $\mathcal{S}$ is a state space. There are special types of states called absorbing states, from which we only incur a reward 0 and transition back to the same state, regardless of the action taken. Such states stall the interaction between the agent and the environment, and we can say that an episode ends once an absorbing state is reached for the first time.

- $\mathcal{A}$ is an action space. The agent is tasked to perform the best action given its current state in order to gather the maximum amount of rewards from the environment before reaching an absorbing state, the end of an episode.

- $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the transition probability function.
  $p(s'|s, a) = p(s_{t+1} = s'|s_t = s, a_t = a)$ is the probability of transitioning from state s to s' after performing action a. p is assumed to be Markovian, i.e.

  $$p(s_{t+1} = s'|s_t = s, a_t = a) = p(s_{t+1} = s'|s_t = s, a_t = a, \ldots)$$

  which means that, conditionally on a state s_t and action a_t, the next state s_{t+1} is independent of the history $\{s_0, a_0, \ldots, s_{t-1}, a_{t-1}\}$. We note that all the MDPs we will consider in this thesis have deterministic dynamics, which means that given $\{s_t = s, a_t = a\}$, there exists a unique state s' to which we can transition, and we denote:

  $$s \xrightarrow{a} s'$$

  This setting can be describe with p as a Dirac measure.

- $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function. $r(s, a)$ is the reward incurred by performing action a in state s. Usually, the reward function is rather defined as a map $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$. Our definition $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is justified by the deterministic nature of our transition dynamics.

- $0 < \gamma \leq 1$ is a discount factor. It discounts future rewards as we shall see shortly when defining the notion of return.

The agent's interaction with the environment is monitored by a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ deciding which action the agent should take in each state. The agent starts its journey in some initial state s_0, then it performs a series of actions according to its policy π and gathers the following cumulative discounted (or undiscounted if $\gamma = 1$) return:

$$\begin{cases} \mathcal{R}^{\pi}(s_0) = \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \\ \forall t \geq 1 : s_{t-1} \xrightarrow{\pi(s_{t-1})} s_t \end{cases} \tag{2.1}$$

The infinity in this sum raises concerns about the divergence of this quantity. However, in the case with bounded rewards, having $0 < \gamma < 1$ is sufficient to ensure that the return always converges because:

$$\forall \pi : \sum_{t=0}^{\infty} |\gamma^t r(s_t, \pi(s_t))| \leq \sum_{t=0}^{\infty} \gamma^t |\max_{s,a} r(s,a)| \leq \frac{|\max_{s,a} r(s,a)|}{1-\gamma}$$

Thus ensuring the absolute convergence of the series $\mathcal{R}^{\pi}(s_0) = \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t))$ for any policy π. There is also another way of ensuring this, and it corresponds to the MDPs we will define in this thesis. If an absorbing state is always guaranteed (with probability 1) to be reached in a finite number of steps, regardless of the policy π, then $\forall \pi : |\mathcal{R}^{\pi}(s_0)| < \infty$.

The return $\mathcal{R}^{\pi}(s_0)$ is in general stochastic, it depends on the stochasticity of the transition dynamics, the rewards and the policy. Therefore, we measure the quality of a policy, in general, via its expected return, also called the value function of π:

$$\mathcal{V}^{\pi}(s_0) = \mathbb{E}[\mathcal{R}^{\pi}(s_0)] \tag{2.2}$$

In a case with deterministic transition dynamics, rewards and policies, the return is also deterministic and thus $\mathcal{R}^{\pi}(s_0) = \mathcal{V}^{\pi}(s_0)$. This is the relevant case for our work, and we choose to consider $\mathcal{V}^{\pi}(s_0)$ instead of $\mathcal{R}^{\pi}(s_0)$ to refer to the same notion. The objective of the agent is to find an optimal policy with respect to the value from the initial state s_0:

$$\pi^* \in \text{Argmax}_{\pi} \mathcal{V}^{\pi}(s_0) \tag{2.3}$$

Searching directly for π^* among all possible policies is unfeasible because the search space is immense; the number of such policies is $|\mathcal{A}|^{|\mathcal{S}|}$. Fortunately, the Markovian property of the transition dynamics yields structural dependencies, in the form of Bellman equations, which can be used to look for π^* iteratively via Dynamic Programming. To formulate the Bellman equations, it is convenient to define the state-action value function of π:

$$\begin{aligned} &\mathcal{Q}^{\pi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \\ &\qquad (s,a) \mapsto \mathbb{E}\left[r(s_0, a_0) + \sum_{t=1}^{\infty} \gamma^t r(s_t, \pi(s_t)) \,\middle|\, s_0 = s, a_0 = a \right] \end{aligned} \tag{2.4}$$

$\mathcal{Q}^{\pi}(s,a)$ is the expected return gathered by starting from s, taking action a, and then following policy π. Since we consider deterministic transitions and deterministic policies, we provide the Bellman Equations we will be working with in the following proposition. For the general case, please refer to (Sutton and Barto, 2018, Chapter 4).

Proposition 1 (Bellman Equations)**.** *Let $(s,a) \in \mathcal{S} \times \mathcal{A}$ be a state-action pair and $s \xrightarrow{a} s'$. Then for any policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ we have the following:*

$$\begin{aligned} \mathcal{Q}^{\pi}(s,a) &= r(s,a) + \gamma \mathcal{V}^{\pi}(s') \\ \mathcal{V}^{\pi}(s) &= \mathcal{Q}^{\pi}(s, \pi(s)) \end{aligned}$$

Moreover, in the special case of the optimal policy π^*, we have:

$$\begin{aligned} \mathcal{Q}^{\pi^*}(s,a) &= r(s,a) + \gamma \mathcal{V}^{\pi^*}(s') \\ \mathcal{V}^{\pi^*}(s) &= \max_{a \in \mathcal{A}} \mathcal{Q}^{\pi^*}(s,a) \end{aligned}$$

Proof

$$\begin{aligned} \mathcal{Q}^{\pi}(s,a) &= \mathbb{E}\left[r(s_0,a_0) + \sum_{t=1}^{\infty} \gamma^t r(s_t, \pi(s_t)) \,\middle|\, s_0 = s, a_0 = a \right] \\ &= r(s,a) + \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t r(s_t, \pi(s_t)) \,\middle|\, s_1 = s' \right] \\ &= r(s,a) + \gamma \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_{t+1}, \pi(s_{t+1})) \,\middle|\, s_1 = s' \right] \\ &= r(s,a) + \gamma \mathcal{V}^{\pi}(s') \end{aligned}$$

On the other hand we have:

$$\begin{aligned} \mathcal{V}^{\pi}(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \,\middle|\, s_0 = s' \right] \\ &= \mathbb{E}\left[r(s_0,a_0) + \sum_{t=1}^{\infty} \gamma^t r(s_t, \pi(s_t)) \,\middle|\, s_0 = s, a_0 = \pi(s) \right] \\ &= \mathcal{Q}^{\pi}(s, \pi(s)) \end{aligned}$$

Then, for the optimal policy π^*, the first result is straightforward:

$$\mathcal{Q}^{\pi^*}(s,a) = r(s,a) + \gamma \mathcal{V}^{\pi^*}(s')$$

For the second result, if it is not satisfied, then we define the policy π' that coincides with π on all states except s where:

$$\pi'(s) = \text{Argmax}_{a \in \mathcal{A}} \mathcal{Q}^*(s,a)$$

In which case we would have:

$$\mathcal{V}^{\pi'}(s) = \max_{a \in \mathcal{A}} \mathcal{Q}^*(s,a) > \mathcal{V}^{\pi^*}(s)$$

therefore contradicting the optimality of π^*. In this reasoning, we implicitly assumed there to be a unique optimal action at each state to avoid overloading the notation. ■

RL is a very promising framework for sequential decision making problems, granted we can formulate an adequate MDP. Constructing an optimal DT fits within the scope of a sequential decision-making problem and can therefore be modeled using RL. Surprisingly, there is limited

research on optimal DTs utilizing RL. In the subsequent paragraphs, we review some of these approaches.

Bonet and Geffner (1998) undertook an approach based on Partially Observable Markov Decision Processes (POMDPs) (Sondik, 1971; Cassandra et al., 1994; Littman et al., 1995). Unfortunately, POMDPs are very hard and costly to solve. In fact (Bonet and Geffner, 1998, Table 1) shows that this method fails to retrieve the optimal DT for MONK1 even after 10000 trials. For this reason, we focus on more recent MDP-based algorithms. Among these methods, Preda (2007) defines the following undiscounted ($\gamma = 1$) MDP:

- **State space:** Given a dataset $\mathcal{D}$, the state space is defined as the set of all partitions of $\mathcal{D}$. A state is labeled as final if all of its elements (subsets of $\mathcal{D}$) are composed of data points sharing the same class.
- **Action space:** Given a state $s = \{s_1, \ldots, s_n\}$, the set of action $\mathcal{A}(s)$ is the set of pairs (att, i) where att is a feature and $i \in \{1, \ldots, n\}$. An action (att, i) partitions s_i using feature att.
- **Reward function:** Given a state-action pair (s, a), $r(s, a) = -1$ if s is not final and $r(s, a) = 0$ otherwise, regardless of action a.

The optimal policy for this MDP can be seen as a DT that partitions $\mathcal{D}$ incrementally until reaching a final state describing the leaves of the DT. From the definition of a final state, this optimal DT perfectly classifies $\mathcal{D}$, moreover, since $r(s, a) = -1$ for any non final state, the optimal policy reaches a final state by taking the smallest number of actions possible, hence the corresponding optimal DT is the smallest DT (in terms of the number of splits or equivalently internal nodes) that perfectly classifies $\mathcal{D}$. Therefore this approach is similar to the SAT-based methods we reviewed in Section 2.3.3, but presents the additional benefit of solving for sparsity instead of optimising a prespecified topology. Nonetheless, as is the case for SAT methods, the goal of perfectly classifying the training set often leads to overfitting. Furthermore, contrary to the B&B methods reviewed in Section 2.3.4, this RL algorithm does not enjoy specifically tailored bounds to prune the search space, which would make a tabular method such as Q-Learning (Watkins and Dayan, 1992) significantly more costly and less scalable than B&B. To address this scalability issue, Preda (2007) rather employs a Linear Q-Learning approach where the state-action value function is a parameterised linear function:

$$\begin{aligned} \forall s \in \mathcal{S}\ \forall a \in A(s) : Q_\theta(s, a) &= \theta \cdot \phi(s, a) \\ \phi(s, a) &= \left(\phi_1^{(s,a)}, \ldots, \phi_k^{(s,a)}\right)^T \in \mathbb{R}^k \\ \theta &= (\theta_1, \ldots, \theta_k) \in \mathbb{R}^k \end{aligned}$$

Each state-action pair (s, a) is mapped to a feature space via $\phi(s, a) = f(g(s, a))$, where $g(s, a) = (g_0(s), g_1(s, a), \ldots, g_l(s, a))$. $g_0(s)$ is the weighted average Entropy of state (partition) s, for any $1 \leq i \leq l$ $g_i(s, a)$ is the gain measure of partitioning s with action a. This gain is based on different metrics, Entropy, Gini index and discriminant power. $f : \mathbb{R}^{l+1} \mapsto R^k$ is a tile coding function with k layers of tiles. Nevertheless, it is unclear whether the linear approximation of Q_θ, along with the feature mapping of state-action pairs, keeps the optimality of the approach, and even if it is the case, it is hard to quantify the computational gain of the linear approach compared to the tabular one. Furthermore, the state space definition is very rigid with respect to the dataset $\mathcal{D}$, which prohibits the extension of this method to the online setting.

In a different work, Nunes et al. (2018) adopt a Monte Carlo Tree Search (MCTS) approach (Browne et al., 2012) with a different undiscounted MDP:

- **State space:** The space of Decision Trees.
- **Action space:** For a state (DT), the set of actions is the set of possible splits of the DT.
- **reward function:** A validation set is provided, on which DTs are evaluated.

Nunes et al. (2018) apply the Upper Confidence Bound for Tree (UCT) algorithm (Kocsis and Szepesvári, 2006; Kocsis et al., 2006) as follows. They define a rollout policy that completes the selected DT with C4.5 on the training set, this DT is then evaluated on a validation set, upon which the result is backpropagated to the selected state and its ancestors (the states that led to the selected state by following the Tree policy). This approach does not differentiate between DTs of different complexities, in fact, the authors rely on a custom definition of terminal states in terms of predefined maximum depth and number of instances, alongside C4.5's pruning strategy. Additionally, by virtue of using C4.5, a pure batch algorithm, this algorithm is not applicable for data streams. It is also worth noting that (Nunes et al., 2018, Table 1) indicates that this method is prohibitively slow, taking up hours to finish (105 hours in an instance).

So far, all the methods we have discussed pertain to the batch setting. However, there are few attempts at handling data streams. Garlapati et al. (2015) define the following MDP that is tailored for this objective:

- **State space:** Given an input $X = \left(X^{(1)}, \ldots, X^{(q)}\right)$, a state $s \in \mathcal{S}$ is a vector of size q where each entry i, is either the value of $X^{(i)}$ or describes that $X^{(i)}$ is unobserved. For example, consider $X = (1, 3, 0, 2, 5)$ and let $\emptyset$ denote an unobserved feature. The state $s = (\emptyset, 3, 0, \emptyset, \emptyset)$ describes that $X^{(2)}$ and $X^{(3)}$ are observed and their respective values are 3 and 0 while features $X^{(1)}, X^{(4)}, X^{(5)}$ are, on the other hand, unobserved. The empty state consists of $\emptyset$ only and is always the initial state, it describes the fact that we have not yet observed any feature of X.
- **Action space:** In a state s, we can either query the value of an unobserved feature or directly label the data point, these two types of actions are called query actions and report actions respectively. The action space is then defined as $\mathcal{A} = \{1, \ldots q, q+1, \ldots, q+K\}$ where $\{1, \ldots, d\}$ are the query actions and $\{d+1, \ldots, d+K\}$ the report actions, where K is the number of classes in our classification problem. When taking action a in a state s, if a is a report action, the episode ends (i.e. we transition to an absorbing state) and the data point is classified according to a, otherwise the next state observes all the already observed features at s in addition to the queried feature. For example, let $X = (1, 0, 2, 0, 3)$ be a data point, $s = (\emptyset, 0, 2, \emptyset, 3)$ the current state and a the query action of the 4^{th} feature, the next state is then $s' = (\emptyset, 0, 2, 0, 3)$ by observing the 4^{th} feature of X.
- **Reward:** The agent is rewarded positively when it reports the right class and negatively otherwise. For the policy to use a fewer number of features, a query action is rewarded negatively with $\lambda < 0$. R_+ denotes the positive reward of a right report action and R_- the negative reward of a wrong report action.
- **Episode:** An episode starts upon the arrival of a new data point, the initial state is then initialized to the empty state and the agent's policy is run until a report action is taken.

Upon the arrival of a new data point X, the agent starts an episode from the empty state, then according to its policy π, it makes a sequence of query actions followed by a report action

ending the episode. This process can be viewed as a descent down a DT T^π where each node represents a state and each query action represents a split of a node with respect to a feature, the leaves of T^π are states where π takes a report action, the prediction of T^π at a leaf l is the report action taken by π at state (leaf) l. By construction, the mapping $\pi \mapsto T^\pi$ is bijective between the set of policies and the set of DTs. Garlapati et al. (2015) then solve this MDP using tabular Q-Learning and call their algorithm Reinforcement Learning-based Decision Trees (RLDT). RLDT naturally handles data streams, but its tabular approach, coupled with the absence of a strategy to prune the search space, makes it hard to scale for large state spaces. Blake and Ntoutsi (2018) set to generalise RLDT to concept drift situations, i.e. when the underlying data distribution is non-stationary, but the main limitations of RLDT still hold.

Part I

Optimal Batch Decision Trees

Chapter 3

Seeking Optimal Sparse Decision Trees in the Space of Decision Trees

---

Contents

3.1 Introduction 49
3.2 Preliminaries 49
3.3 The Algorithm: DT-Miner 53
3.4 Experiments 56
3.5 Conclusion 58
3.6 Appendix: Table of Notation 59
3.7 Appendix: Proofs 60

3.1 Introduction

The current state of the art in Decision Tree optimisation are search algorithms that leverage DP and B&B. As we have discussed in Section 2.3.4, there are two main approaches in this literature, BFS methods such as OSDT (Hu et al., 2019) and GOSDT (Lin et al., 2020), and DFS methods such as DL8.5 (Aglin et al., 2020) MurTree (Demirović et al., 2022) and STreeD (van der Linden et al., 2024).

In this thesis, we investigate new ways of formulating and improving BFS methods. Specifically, our approach formulates (sparse) DT optimisation as a Markov Decision Process. The richness of this view will provide us with the proper intuition to infer analytical bounds, for our B&B strategy, that are efficient at pruning the search space and yield satisfying theoretical results on the computational complexity of the induced algorithms. In this chapter, we introduce DT-Miner, a BFS algorithm with a new analytical bound we call *Purification Bound*. DT-Miner solves for sparsity. However, the formulation of its state space as the space of DTs significantly hinders its runtime performance. This is due to a weak information reuse compared to DP methods that operate at the level of branches. Moreover, DT-Miner's Python implementation further prohibits it from being a serious competitor of C++ implementations such as GOSDT, as will be evident from Table 3.2. Nevertheless, DT-Miner's MDP formulation and Purification Bound offer key insights that pave the way for our follow-up algorithm Branches, the main contribution of this thesis. Branches is introduced in Chapter 4.

3.2 Preliminaries

We consider the batch classification problem introduced in Section 2.1. In the following sections, we introduce the necessary terminology and notation to formulate our optimisation problem within a MDP.

3.2.1 Branches

A branch l is a conjunction of clauses on the features of the following form:

$$l = \bigwedge_{v=1}^{\mathcal{S}(l)} \mathbb{1}\left\{X^{(i_v)} = j_v\right\}$$

such that $\forall v \in \{1, \dots, \mathcal{S}(l)\} : i_v \in \{1, \dots, q\}$, $j_v \in \{1, \dots, C_{i_v}\}$ and:

$$\forall v, v' \in \{1, \dots \mathcal{S}(l)\} : v \neq v' \implies i_v \neq i_{v'} \tag{3.1}$$

This condition ensures that no feature is used in more than one clause within l. We refer to these clauses as rules or splits. $\mathcal{S}(l)$ is the number of splits in l.

Consider a branch $l = \bigwedge_{v=1}^{\mathcal{S}(l)} \mathbb{1}\left\{X^{(i_v)} = j_v\right\}$. For any datum $X \in \mathcal{X}$, regardless of whether it is included in the dataset $\mathcal{D}$ or not, the valuation of l at X is denoted $l(X) \in \{0, 1\}$ and defined:

$$l(X) = 1 \iff \bigwedge_{v=1}^{\mathcal{S}(l)} \mathbb{1}\left\{X^{(i_v)} = j_v\right\} = 1$$

When $l(X) = 1$, we say that X is in l or that l contains X. The branch containing all possible data is called the root and denoted Ω. Since $l(X)$ remains invariant when reordering the

splits, we uniquely represent l by sorting its splits from the smallest feature index to the highest, specifically we impose the condition $1 \leq i_1 < \ldots < i_{\mathcal{S}(l)} \leq q$. This unique representation is useful for the memoisation (or caching) procedure that takes place in our Dynamic Programming procedure.

In the following, we define the notion of splitting a branch. Let $i \in \{1, \ldots, q\} \setminus \{i_1, \ldots, i_{\mathcal{S}(l)}\}$ be an unused feature in the splits of l. We define the children of l that stem from splitting l with respect to i as the set $\mathrm{Ch}(l, i) = \{l_1, \ldots, l_{C_i}\}$ where:

$$\forall j \in \{1, \ldots, C_i\} : l_j = l \wedge \mathbb{1}\{X^{(i)} = j\} \tag{3.2}$$

The dataset $\mathcal{D}$ provides an empirical probability distribution of the data. The probability that l contains a datum is:

$$\begin{cases} \mathbb{P}[l(X) = 1] = \frac{n(l)}{n} \\ n(l) = \sum_{m=1}^{n} l(X_m) \end{cases}$$

where $n(l)$ is the number of data in l. Likewise, we want to define the probability that a datum is in l and is correctly classified. For this purpose, we define the predicted class in l as the majority class:

$$\begin{cases} k^*(l) = \mathrm{Argmax}_{1 \leq k \leq K}\{n_k(l)\} \\ n_k(l) = \sum_{m=1}^{n} l(X_m)\, \mathbb{1}\{Y_m = k\} \end{cases} \tag{3.3}$$

where $n_k(l)$ is the number data in l that are of class k. Now we can deduce the probability that a datum is in l and is correctly classified with:

$$\mathcal{H}(l) = \mathbb{P}[l(X) = 1, Y = k^*(l)] = \frac{n_{k^*(l)}(l)}{n} \tag{3.4}$$

3.2.2 Decision Trees

A Decision Tree $T = \{l_1, \ldots, l_{|T|}\}$ is a collection of branches that stem from successive splits starting from the root branch Ω. $\mathcal{S}(T)$ denotes the number of these splits. Intuitively, $\mathcal{S}(T)$ is the number of *internal nodes* in T. Fig. 3.1 shows an example of such a Decision Tree. T partitions Ω in the following sense:

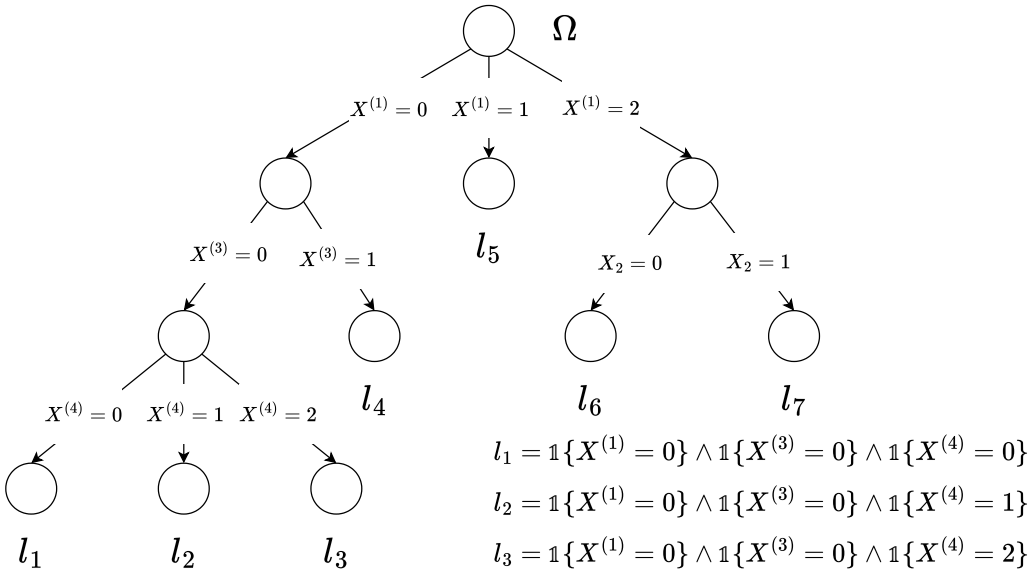
$$\begin{cases} \Omega = \bigvee_{u=1}^{|T|} l_u \\ \forall u, u' \in \{1, \ldots, |T|\} : u \neq u' \implies l_u \wedge l_{u'} = 0 \end{cases}$$

here $l_u \wedge l_{u'} = 0$ just means that $\forall X \in \mathcal{X} : l_u \wedge l_{u'}(X) = 0$. For any datum $X \in \mathcal{X}$, T predicts the majority class of the branch l_u containing X:

$$T(X) = \sum_{u=1}^{|T|} l_u(X)\, k^*(l_u) \in \{1, \ldots, K\} \tag{3.5}$$

Now we can define the accuracy of T as the probability that T makes a correct prediction:

$$\begin{aligned} \mathcal{H}(T) &= \mathbb{P}[T(X) = Y] \\ &= \sum_{u=1}^{|T|} \mathbb{P}[T(X) = Y, l_u(X) = 1] \\ &= \sum_{u=1}^{|T|} \mathbb{P}[k^*(l_u) = Y, l_u(X) = 1] = \sum_{u=1}^{|T|} \mathcal{H}(l_u) \end{aligned} \tag{3.6}$$



$$l_1 = \mathbb{1}\{X^{(1)} = 0\} \wedge \mathbb{1}\{X^{(3)} = 0\} \wedge \mathbb{1}\{X^{(4)} = 0\}$$
$$l_2 = \mathbb{1}\{X^{(1)} = 0\} \wedge \mathbb{1}\{X^{(3)} = 0\} \wedge \mathbb{1}\{X^{(4)} = 1\}$$
$$l_3 = \mathbb{1}\{X^{(1)} = 0\} \wedge \mathbb{1}\{X^{(3)} = 0\} \wedge \mathbb{1}\{X^{(4)} = 2\}$$
$$l_4 = \mathbb{1}\{X^{(1)} = 0\} \wedge \mathbb{1}\{X^{(3)} = 1\}$$
$$l_5 = \mathbb{1}\{X^{(1)} = 1\}$$
$$l_6 = \mathbb{1}\{X^{(1)} = 2\} \wedge \mathbb{1}\{X^{(2)} = 0\}$$
$$l_7 = \mathbb{1}\{X^{(1)} = 2\} \wedge \mathbb{1}\{X^{(2)} = 1\}$$

Figure 3.1: Example of a Decision Tree $T = \{l_1, \ldots, l_7\}$ on the feature space $\mathcal{X} = \{0, 1, 2\} \times \{0, 1\} \times \{0, 1\} \times \{0, 1, 2\}$. In this example we have $\mathcal{S}(T) = 4$

The additivity property in the second line is due to $\{l_1, \ldots, l_{|T|}\}$ forming a partition of Ω. The third line stems from the fact that, when $l_u(X) = 1$, then we necessarily have $T(X) = k^*(l_u)$ as per Eq. (3.5).

Maximising accuracy is not a suitable objective. It overlooks the sparsity portion of the problem and likely leads to overtraining issues. For instance, the complete DT that uses all possible splits is the most accurate solution, yet it describes a very complex DT that just corresponds to point-wise classification of all data. Such solution is of no interest. To incorporate sparsity, we rather consider the following regularised objective:

$$\mathcal{H}_\lambda(T) = -\lambda \mathcal{S}(T) + \mathcal{H}(T) \tag{3.7}$$

$\lambda \in [0, 1]$ is a penalty parameter penalising DTs with too many splits. This objective is employed by CART during the pruning phase, it was also considered by OCT (Bertsimas and Dunn, 2017). Our objective now is to find the optimal solution T^* with respect to this objective, i.e.

$$T^* = \text{Argmax}_T \mathcal{H}_\lambda(T) \tag{3.8}$$

3.2.3 Markov Decision Process (MDP)

As explained in Section 2.5, in order to formulate a Reinforcement Learning problem, we need to define an adequate MDP. In this section, we define the following undiscounted ($\gamma = 1$) MDP.

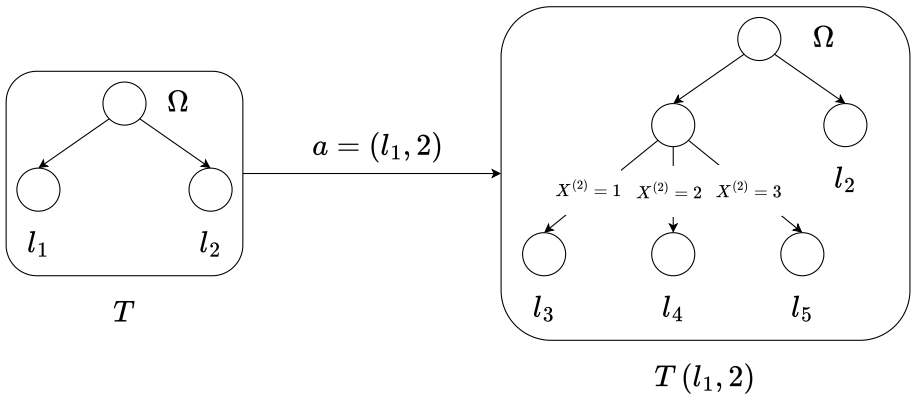


Figure 3.2: Example of a transition $T \xrightarrow{a} T(l_1, 2)$ involving the split action $a = (l_1, 2)$.

State space: The set of all possible DTs. The initial state is always the root DT Ω[1].

Action space: For each state T, $\mathcal{A}(T)$ denotes the set of permissible actions at T. There are two types of actions:

- The terminal action $\overline{a}$. It transitions from T to an absorbing state $\overline{T}$. We denote this transition with $T \xrightarrow{\overline{a}} \overline{T}$.
- Split actions that split a branch of the DT with respect to a feature. Let $T = \{l_1, \ldots, l_{|T|}\}$ be a DT. Each branch $l_u = \bigwedge_{v=1}^{\mathcal{S}(l_u)} \mathbb{1}\{X^{(i_v)} = j_v\}$ can be split with respect to an unused feature in $\mathcal{A}(l_u) = \{1, \ldots, q\} \setminus \{i_1, \ldots, i_{\mathcal{S}(l_u)}\}$, therefore the set of possible split actions at T is:

$$\mathcal{A}(T) \setminus \{\overline{a}\} = \bigcup_{u=1}^{|T|} \{l_u\} \times \mathcal{A}(l_u)$$

  Upon taking split action (l_u, i), the following transition is performed:

$$T \xrightarrow{(l_u, i)} T(l_u, i) = T \setminus \{l_u\} \cup \mathrm{Ch}(l_u, i) \tag{3.9}$$

  $T(l_u, i)$ stems from replacing l_u in T with its children branches $\mathrm{Ch}(l_u, i)$. See Fig. 3.2 for an example of a transition involving a split action.

Reward function: $r(T, a)$ denotes the reward of taking action $a \in \mathcal{A}(T)$ in state T.

- If a is a split action, then $r(T, a) = -\lambda$ regardless of T. $\lambda \in [0, 1]$ is the penalty parameter defined in Eq. (3.7).
- If $a = \overline{a}$, then $r(T, \overline{a}) = \mathcal{H}(T)$.
- If $T = \overline{T}$, i.e. T is an absorbing state, then $r(T, a) = 0$ by the definition of an absorbing state.

[1]The rigorous notation here would be $\{\Omega\}$ since we are considering a DT, however, to make the notation lighter, we opted to also denote Ω the DT that only contains the root.

A policy π maps each state T to one of its permissible actions $\pi(T) \in \mathcal{A}(T)$. The trajectory of π starting from T is the sequence of states $(T_t)_{t=0}^{\infty}$ such that $T_0 = T$ and $\forall t \geq 0 : T_t \xrightarrow{\pi(T_t)} T_{t+1}$. From Eq. (2.1) and Eq. (2.2), and the fact that π and the transition dynamics of our MDP are deterministic, we get, as discussed in Section 2.5, the following value of π at T:

$$\mathcal{V}^{\pi}(T) = \sum_{t=0}^{\infty} r(T_t, \pi(T_t))$$

Proposition 2 ensures that all policies eventually arrive in an absorbing state after a finite number of steps, which guarantees that all policies have finite values.

Proposition 2. *Let T be a non-absorbing state and π a policy with $(T_t)_{t=0}^{\infty}$ its trajectory starting from T. Then there exists a non-absorbing state $T^{\pi}(T)$ and $\tau \geq 1$ such that:*

$$\begin{cases} T_{\tau-1} = T^{\pi}(T) \\ \forall t \geq \tau : T_t = \overline{T^{\pi}(T)} \end{cases}$$

When $T = \Omega$, we call $T^{\pi}(\Omega)$ the DT of π and we abbreviate the notation $T^{\pi}(\Omega) \equiv T^{\pi}$.

The ultimate goal when formulating an MDP is to seek its optimal policy $\pi^* = \text{Argmax}_{\pi} \mathcal{V}^{\pi}(\Omega)$. However, **How does π^* relate to our DT optimisation problem** (3.8)**?** Proposition 3 links the two problems.

Proposition 3. *For any policy π and state T, the value of π from T satisfies:*

$$\mathcal{V}^{\pi}(T) = -\lambda \left[\mathcal{S}(T^{\pi}(T)) - \mathcal{S}(T)\right] + \mathcal{H}(T^{\pi}(T))$$

In particular $\mathcal{V}^{\pi}(\Omega) = -\lambda \mathcal{S}(T^{\pi}) + \mathcal{H}(T^{\pi}) = \mathcal{H}_{\lambda}(T^{\pi})$.

Proposition 3 links the value of a policy to the regularised accuracy of its DT. On the other hand, since any DT T is constructed with successive splits starting from Ω, there always exists a policy π such that $T^{\pi} = T$, and therefore $\mathcal{V}^{\pi}(\Omega) = \mathcal{H}_{\lambda}(T^{\pi}) = \mathcal{H}_{\lambda}(T)$. This result concludes the equivalence between finding the optimal DT and the optimal policy:

$$\begin{cases} T^* = \text{Argmax}_T \{\mathcal{H}_{\lambda}(T)\}, \ \pi^* = \text{Argmax}_{\pi} \{\mathcal{V}^{\pi}(\Omega)\} \\ T^* = T^{\pi^*} \end{cases} \tag{3.10}$$

Our objective now is to find π^* and then deduce T^* as T^{π^*}. We abbreviate the notation $\mathcal{V}^{\pi^*} \equiv \mathcal{V}^*$. Moreover, when considering the state-action value function defined in Eq. (2.4), we also abbreviate the notation for the optimal quantity via $\mathcal{Q}^{\pi^*} \equiv \mathcal{Q}^*$.

3.3 The Algorithm: DT-Miner

DT-Miner is a Tree Search algorithm that is enhanced with a B&B pruning component. The idea pertains to estimating the optimal state action values $\mathcal{Q}^*(T, a)$ and then deducing π^* as the greedy policy with respect to $\mathcal{Q}^*$:

$$\pi^*(T) = \text{Argmax}_{a \in \mathcal{A}(T)} \mathcal{Q}^*(T, a)$$

We denote $\mathcal{Q}(T, a)$ these estimates. Each DT T can be reached from Ω via different policies. The DP portion of the algorithm stores the estimates $\mathcal{Q}(T, a)$ of T in a memo (cache) and reuses

them whenever T is encountered again regardless of the path taken from Ω to reach it. On the other hand, the B&B portion defines these estimates $\mathcal{Q}(T,a)$ as upper bounds on the true values $\mathcal{Q}^*(T,a)$. The following sections describe how the estimates $\mathcal{Q}(T,a)$ are defined and the steps followed by the search algorithm.

3.3.1 Estimating the optimal state-action values $\mathcal{Q}^*(T,a)$

The value of any policy from an absorbing state $\overline{T}$ is 0, thus there is no need for estimating the values $\mathcal{Q}^*\left(\overline{T},a\right)$. Let $T=\{l_1,\ldots,l_{|T|}\}$ be a non-absorbing state and $a\in\mathcal{A}(T)$. For the terminal action $\overline{a}$, $\mathcal{Q}^*(T,\overline{a})$ is directly accessible from the data via Eq. (3.6) and the following:

$$\mathcal{Q}(T,\overline{a})=\mathcal{Q}^*(T,\overline{a})=r(T,\overline{a})=\mathcal{H}(T) \tag{3.11}$$

Now it remains to estimate $\mathcal{Q}(T,a)$ for a split action $a\in\mathcal{A}(T)\setminus\{\overline{a}\}$. Let $a=(l,i)$ be a split action. We recall that:

$$T\xrightarrow{(l,i)}T(l,i)=T\setminus\{l\}\cup\mathrm{Ch}(l,i)$$

The Bellman equations, introduced in Proposition 1, yield:

$$\begin{aligned}\mathcal{Q}^*(T,a)&=-\lambda+\mathcal{V}^*(T(l,i))\\ \mathcal{V}^*(T)&=\mathcal{Q}^*(T,\pi^*(T))=\max_{a\in\mathcal{A}(T)}\mathcal{Q}^*(T,a)\end{aligned}$$

which suggests defining the following recursive estimates:

$$\mathcal{Q}(T,(l,i))=-\lambda+\mathcal{V}(T(l,i)) \tag{3.12}$$

$$\mathcal{V}(T(l,i))=\max_{a\in\mathcal{A}(T(l,i))}\mathcal{Q}(T(l,i),a) \tag{3.13}$$

The estimate $\mathcal{Q}(T,(l,i))$ in Eq. (3.12) can only be calculated if the estimates $\mathcal{V}(T(l,i))$ in Eq. (3.13) are available. Otherwise they are initialised according to Proposition 4.

Proposition 4 (Purification Bounds)**.** *Let $T=\{l_1,\ldots,l_{|T|}\}$ be a non-absorbing state and without loss of generality, let $(l_1,i)\in\mathcal{A}(T)\setminus\{\overline{a}\}$ be a split action. We define the Purification Bounds estimates with:*

$$\begin{aligned}\mathcal{Q}(T,(l_1,i))&=-\lambda+\mathbb{P}[l_1(X)=1]+\sum_{u=2}^{|T|}\max\{\mathcal{H}(l_u),-\lambda+\mathbb{P}[l_u(X)=1]\}\\ &=-\lambda+\frac{n(l_1)}{n}+\sum_{u=2}^{|T|}\max\left\{\frac{n_{k^*(l_u)}(l_u)}{n},-\lambda+\frac{n(l_u)}{n}\right\}\end{aligned} \tag{3.14}$$

$$\begin{aligned}\mathcal{V}(T)&=\sum_{u=1}^{|T|}\max\{\mathcal{H}(l_u),-\lambda+\mathbb{P}[l_u(X)=1]\}\\ &=\sum_{u=1}^{|T|}\max\left\{\frac{n_{k^*(l_u)}(l_u)}{n},-\lambda+\frac{n(l_u)}{n}\right\}\end{aligned} \tag{3.15}$$

These estimates are upper bounds on $\mathcal{Q}^*(T,(l,i))$ and $\mathcal{V}^*(T)$:

$$\begin{aligned}\mathcal{Q}(T,(l,i))&\geq\mathcal{Q}^*(T,(l,i))\\ \mathcal{V}(T)&\geq\mathcal{V}^*(T)\end{aligned}$$

The bounds in Proposition 4 are based on the following intuition. Consider a non-absorbing state T, for each branch $l \in T$, we can either split l or not. If we do not split it, then its contribution to the estimate $\mathcal{V}(T)$ is $\mathcal{H}(l)$, otherwise its contribution is rather $-\lambda$ plus the contributions of its children, which is upper bounded by $-\lambda + \mathbb{P}[l(X) = 1]$. In fact, this value is only reached if the children are **pure**, i.e. each child l' contains only examples of the same class:

$$\exists! k \in \{1, \dots, K\} : \forall m \in \{1, \dots, n\} : l'(X_m) = 1 \implies Y_m = k$$

where we recall that $\mathcal{D} = \{(X_m, Y_m)\}_{m=1}^n$. It is as if we compare the contribution of purifying each branch l with $\mathcal{H}(l)$ and we take the maximum. Hence the name **Purification Bounds**.

Summary: $\mathcal{Q}(T, \overline{a})$ is calculated with Eq. (3.11) for the terminal action. For a split action (l, i), $\mathcal{Q}(T, (l, i))$ is calculated with Eq. (3.12) when estimates for the children are available, otherwise it is initialised with Eq. (3.14).

It is not sufficient to define these estimates, we also need to carefully choose which state-action pairs to estimate in order to achieve a fast convergence towards the true optimal values $\mathcal{Q}^*(T, a)$. Indeed, we cannot for instance just calculate these estimates for all possible Decision Trees, the search space is immense. We circumvent this issue with an adequate Tree search strategy, which we introduce in the next section.

3.3.2 The Search strategy

Initially, all the states T are labelled as unvisited and incomplete, which means that the values $\mathcal{V}^*(T)$ are unknown. Similarly, the state-action pairs (T, a) are also labelled as incomplete because their values $\mathcal{Q}^*(l, a)$ are also unknown. We initialise an empty memo where we store the estimates of the encountered states and state-action pairs, this is crucial for Dynamic Programming. At each iteration, DT-Miner follows the Tree Search steps below:

- **Selection:** Initialise an empty list *path*. Starting from the root $T = \Omega$. Choose the action maximising the estimate:
  $$a = \text{Argmax}_{a' \in \mathcal{A}(T)} \mathcal{Q}(T, a')$$
  Append (T, a) to *path* and transition to the next state. Repeat this process until reaching an unvisited or an absorbing state T.
- **Expansion:** If T is absorbing, then we move to the Backpropagation step. Otherwise, we first set $\mathcal{Q}(T, \overline{a}) = \mathcal{H}(T)$ according to Eq. (3.11), we label $(T, \overline{a})$ as complete, and we store $\mathcal{Q}(T, \overline{a})$ in the memo. Then we estimate $\mathcal{Q}(T, (l, i))$ for split actions. For each split action (l, i), if the memo contains $\mathcal{V}(T(l, i))$, we define $\mathcal{Q}(T, (l, i))$ according to Eq. (3.12), otherwise it is initialised via Eq. (3.14). If $T(l, i)$ is complete, then we label $(T, (l, i))$ as complete as well. After all the split actions have been estimated, $\mathcal{V}(T)$ is then deduced with Eq. (3.13) and stored in the memo. If $(T, (l, i))$ is complete for all split actions (l, i), we label T as complete. We end this step by labelling T as visited.
- **Backpropagation:** Update $\mathcal{Q}(T, a)$ and $\mathcal{V}(T)$ for all (T, a) in *path* by a Backward recursion. For $j = \textit{length}(\textit{path}) - 1, \dots, 0$, let $(T, a) = \textit{path}[j]$, we update $\mathcal{Q}(l, a)$ and $\mathcal{V}(l)$ with Eq. (3.12) and Eq. (3.13) respectively. We also label T and (T, a) as complete in similar fashion to the Expansion step. We update $\mathcal{R}(T)$ in the memo.

The algorithm runs until Ω is complete, in which case DT-Miner has achieved and proved optimal convergence, and the optimal policy satisfies:

$$\pi^*(T) = \text{Argmax}_{a \in \mathcal{A}(T)} \mathcal{Q}(T, a)$$

Table 3.1: Number of examples n, number of features q, number of classes K and penalty parameter λ for the different datasets used in our experiments.

Dataset	n	q	K	λ
monk1-l	124	11	2	0.01
monk1-f	124	11	2	0.001
monk1-o	124	6	2	0.01
monk2-l	169	11	2	0.001
monk2-f	169	11	2	0.001
monk2-o	169	6	2	0.001
monk3-l	122	11	2	0.001
monk3-f	122	11	2	0.001
monk3-o	122	6	2	0.001
tic-tac-toe	958	18	2	0.005
tic-tac-toe-o	958	9	2	0.005
car-eval	1728	15	4	0.005
car-eval-o	1728	6	4	0.005
nursery	12960	19	5	0.01
nursery-o	12960	8	4	0.01
mushroom	8124	95	2	0.01
mushroom-o	8124	22	2	0.01
kr-vs-kp	3196	37	2	0.01
kr-vs-kp-o	3196	36	2	0.01
zoo	101	20	7	0.001
zoo-o	101	16	7	0.001
lymph	148	18	4	0.01
lymph-o	148	41	4	0.01
balance	576	16	2	0.01
balance-o	576	4	2	0.01

Remark: DT-Miner is a Tree Search algorithm because its Selection step can be viewed as descending a Tree graph (search space) where nodes are states (DTs) and edges are actions.

3.4 Experiments

We compare DT-Miner with the state of the art BFS algorithms OSDT, GOSDT and PyGOSDT based on the following metrics: **optimal convergence**, **runtime** and **number of iterations**. The reason we restrict this comparison to BFS algorithms is due to DT-Miner's weak performance. As such, there is no insight to gain by further comparing it to DFS methods as well. We delay the full comparison to Chapter 4, where we introduce the more refined Branches algorithm.

We employ 11 datasets from the UCI repository, which we chose because of their frequent use in benchmarking optimal Decision Tree algorithms. For each dataset, different types of encodings are considered: Suffix -l indicates a One-Hot Encoding where the last category of each feature is dropped, likewise -f indicates a One-Hot Encoding that rather drops the first category, while -o is for an Ordinal Encoding. The datasets are described in Table 3.1. We chose different

Table 3.2: Comparing DT-Miner with the state of the art BFS algorithms.

Dataset	OSDT			PyGOSDT			GOSDT			DT-Miner		
	objective	time (s)	iterations	objective	time (s)	iterations	objective	time (s)	iterations	objective	time (s)	iterations
monk1-l	**0.93**	71	2e6	**0.93**	181	3e6	**0.93**	**0.71**	3e4	**0.93**	3.9	**6e3**
monk1-f	0.97	*TO*	2e4	0.97	*TO*	2e3	**0.983**	**4.02**	**9e4**	0.870	*TO*	4e5
monk1-o	—	—	—	—	—	—	—	—	—	**0.9**	**12**	**2e4**
monk2-l	0.95	*TO*	7e4	0.95	*TO*	400	**0.968**	**10**	**1e5**	0.762	*TO*	2e5
monk2-f	0.90	*TO*	4e4	0.90	*TO*	3e4	**0.933**	**11.1**	**1e5**	0.697	*TO*	2e5
monk2-o	—	—	—	—	—	—	—	—	—	0.742	*TO*	3e5
monk3-l	0.979	*TO*	596	0.979	*TO*	123	**0.981**	**7.38**	**8e4**	0.893	*TO*	4e5
monk3-f	0.975	*TO*	1e4	0.973	*TO*	9e3	**0.983**	**2.13**	**5e4**	0.952	*TO*	3e5
monk3-o	—	—	—	—	—	—	—	—	—	0.974	*TO*	4e5
tic-tac-toe	0.765	*TO*	40	0.808	*TO*	37	**0.850**	**41**	**1.6e6**	0.738	*TO*	1e5
tic-tac-toe-o	—	—	—	—	—	—	—	—	—	0.734	*TO*	1e5
car-eval	—	—	—	—	—	—	**0.799**	**18**	**9e5**	0.785	*TO*	1.4e5
car-eval-o	—	—	—	—	—	—	—	—	—	0.793	*TO*	1e5
nursery	—	—	—	—	—	—	**0.765**	*TO*	7e5	0.744	*TO*	3e4
nursery-o	—	—	—	—	—	—	—	—	—	0.817	*TO*	3e4
mushroom	**0.945**	*TO*	4e6	**0.945**	*TO*	2e6	0.925	*TO*	1e6	0.938	*TO*	1e4
mushroom-o	—	—	—	—	—	—	—	—	—	**0.975**	**0.17**	**5**
kr-vs-kp	**0.900**	*TO*	6e4	**0.900**	*TO*	2e4	0.815	*TO*	4e5	**0.900**	*TO*	5e4
kr-vs-kp-o	—	—	—	—	—	—	—	—	—	**0.900**	*TO*	5e4
zoo	—	—	—	—	—	—	**0.992**	**34**	**3e5**	**0.992**	139	4e5
zoo-o	—	—	—	—	—	—	—	—	—	**0.993**	**139**	**2e5**
lymph	—	—	—	—	—	—	0.784	*TO*	1e6	**0.794**	*TO*	8e4
lymph-o	—	—	—	—	—	—	—	—	—	0.808	*TO*	1e5
balance	**0.693**	*TO*	1e5	**0.693**	*TO*	3e4	**0.693**	**21**	**1e6**	0.680	*TO*	1.7e5
balance-o	—	—	—	—	—	—	—	—	—	0.670	*TO*	1.7e5

encodings because they yield problems with varying degrees of difficulty. Moreover, the state of the art algorithms exclusively consider binary features, thus necessitating a preliminary binary encoding. This seemingly benign detail can significantly harm performance by introducing unnecessary splits to the problem, as we will discuss in detail in Chapter 4. The advantage of our approach is that it can sidestep this issue via a direct application to the original dataset (in its Ordinal Encoding form).

Table 3.2 summarises the results where each algorithm is given a time limit of 5 minutes. When comparing the Python implementations OSDT, PyGOSDT and DT-Miner, the monk1-l experiment seems to indicate that our Purification Bound prunes the search space more efficiently than the bounds used by OSDT and PyGOSDT. Indeed, DT-Miner achieves optimality significantly faster and with substantially fewer iterations. Furthermore, DT-Miner's ability to treat ordinal encoded data can significantly improve its performance as shown by the mushroom-o experiment. Nevertheless, these experiments showcasing the superiority of DT-Miner over OSDT and PyGOSDT are rare, and as such, they are inconclusive with regard to the advantages of employing DT-Miner's Purification Bound strategy. Moreover, when compared with GOSDT, a C++ implementation, DT-Miner's performance is far from being satisfactory. In many experiments, DT-Miner reaches timeout while it takes GOSDT only few seconds to retrieve the

optimal DT, this is the case for monk1-f, monk2-l, monk2-f, monk3-l, monk3-f, tic-tac-toe and car-eval; and even when DT-Miner terminates, GOSDT achieves optimality significantly faster, as shown by monk1-l and zoo. This is more of a testament to the superiority of C++ implementations over Python implementations as evident by the large gap in performance between GOSDT and PyGOSDT. Nevertheless, with these findings, DT-Miner cannot pretend to be a serious competitor of GOSDT.

3.5 Conclusion

In this first chapter, we introduced a new way of formulating DT optimisation within an MDP framework. This formulation provided us with sufficient tools for deriving a new analytical bound for pruning the search space, which we call Purification Bound. The resulting BFS algorithm, DT-Miner, can potentially prune the search space more efficiently than OSDT and PyGOSDT, a fact that is corroborated by some experiments, although not many. In addition, we have shown that DT-Miner performs significantly worse when compared to a C++ implementation such as GOSDT, and as a consequence, DT-Miner is not competitive with GOSDT for practitioners.

Nevertheless, the MDP formulation and the Purification Bound remain promising ideas to explore. There are two main avenues we can investigate for improving DT-Miner:

- Implementing DT-Miner in C++. This could yield a significant improvement as is the case for GOSDT over PyGOSDT.
- Refining the MDP so that the induced DP procedure benefits from the full structural properties of DT optimisation.

In Chapter 4 we explore the second option since it is more promising because the resulting algorithm, Branches, can itself be implemented in C++, thus resulting in significantly larger improvements. We will see that, indeed, Branches outperforms the state of the art, both BFS and DFS algorithms, on many datasets all while being implemented in Python.

3.6 Appendix: Table of Notation

Table 3.3: Table of Notation

X	$=$	$\left(X^{(1)}, \ldots, X^{(q)}\right)$, an input of features.
$X^{(i)}$	$\in$	$\{1, \ldots, C_i\}$, a feature.
$\mathcal{X}$	$=$	$\prod_{i=1}^{q} \{1, \ldots, C_i\}$ the features space.
Y	$\in$	$\{1, \ldots, K\}$, the class variable.
$\mathcal{D}$	$=$	$\{(X_m, Y_m)\}_{m=1}^{n}$, Dataset of labelled examples.
l	$=$	$\bigwedge_{v=1}^{\mathcal{S}(l)} \mathbb{1}\left\{X^{(i_v)} = j_v\right\}$ a branch.
$\mathcal{S}(l)$	$\triangleq$	The number of splits in l, the number of clauses in l.
$l(X)$	$\triangleq$	Valuation of l for input X. When $l(X) = 1$, we say that X is in l.
Ω	$\triangleq$	The root. Branch that valuates to 1 for all possible inputs.
$\mathrm{Ch}(l, i)$	$\triangleq$	Children of l when splitting with respect to feature i.
$\mathrm{Ch}(l, i)$	$=$	$\{l_1, \ldots, l_{C_i}\}$, $l_j = l \wedge \mathbb{1}\left\{X^{(i)} = j\right\}$
$n(l)$	$\triangleq$	Number of examples in l.
$n_k(l)$	$\triangleq$	Number of examples in l of class k.
$k^*(l)$	$=$	$\mathrm{Argmax}_{1 \leq k \leq K}\{n_k(l)\}$, majority class in l.
$\mathbb{P}[l(X) = 1]$	$\triangleq$	Empirical probability that X is in l.
$\mathcal{H}(l)$	$=$	$\mathbb{P}[l(X) = 1, Y = k^*(l)]$, probability that an example is in l and is correctly classified.
DT	$\triangleq$	Collection of branches partitioning Ω, it stems from a series of splits of Ω.
$T(X)$	$\triangleq$	Predicted class of X by T. Majority class of the branch containing X.
$\mathcal{H}(T)$	$=$	$\mathbb{P}[T(X) = Y]$, accuracy of DT T.
$\mathcal{H}_\lambda(T)$	$\triangleq$	Regularized Objective function evaluated at DT T.
$\mathcal{H}_\lambda(T)$	$=$	$\mathbb{P}[T(X) = Y] - \lambda \mathcal{S}(T)$
$\mathcal{S}(T)$	$\triangleq$	Number of splits to construct DT T.
λ	$\in$	$[0, 1]$, penalty parameter.
T^*	$=$	$\mathrm{Argmax}_T\{\mathcal{H}_\lambda(T)\}$, optimal DT.
$\mathcal{A}(T)$	$\triangleq$	Action space at state T.
$\overline{a}$	$\triangleq$	Terminal action.
$T \xrightarrow{a} T'$	$\triangleq$	Transition from T to T' through action a.
$T(l, i)$	$\triangleq$	The state we transition to when taking the split action (l, i) in state T.
$T(l, i)$	$=$	$T \setminus \{l\} \cup \mathrm{Ch}(l, i)$.
$\overline{T}$	$\triangleq$	Absorbing state, $T \xrightarrow{\overline{a}} \overline{T}$.
$r(T, a)$	$\triangleq$	Reward of taking action a in state T.
π	$\triangleq$	Policy, maps each state T to an action $\pi(T) \in \mathcal{A}(T)$.
$\mathcal{V}^\pi(T)$	$\triangleq$	Value of policy π starting from T.
$\mathcal{Q}^\pi(T, a)$	$\triangleq$	State-action value of policy π at state-action pair (T, a).
$T^\pi(T)$	$\triangleq$	DT that stems from following policy π from T.
T^π	$\equiv$	$T^\pi(\Omega)$
π^*	$=$	$\mathrm{Argmax}_\pi \mathcal{V}^\pi(\Omega)$, the optimal policy.
T^*	$=$	T^{π^*}
$\mathcal{V}^*$	$\equiv$	$\mathcal{V}^{\pi^*}$
$\mathcal{Q}^*$	$\equiv$	$\mathcal{Q}^{\pi^*}$
$\mathcal{V}(T)$	$\triangleq$	Estimated upper bound on $\mathcal{V}^*(T)$.
$\mathcal{Q}(T, a)$	$\triangleq$	Estimated upper bound on $\mathcal{Q}^*(T, a)$

3.7 Appendix: Proofs

Proposition 2. *Let T be a non-absorbing state and π a policy with $(T_t)_{t=0}^{\infty}$ its trajectory starting from T. Then there exists a non-absorbing state $T^{\pi}(T)$ and $\tau \geq 1$ such that:*

$$\begin{cases} T_{\tau-1} = T^{\pi}(T) \\ \forall t \geq \tau : T_t = \overline{T^{\pi}(T)} \end{cases}$$

When $T = \Omega$, we call $T^{\pi}(\Omega)$ the DT of π and we abbreviate the notation $T^{\pi}(\Omega) \equiv T^{\pi}$.

Proof The proof proceeds by induction on $\mathcal{S}(T)$.

If $\mathcal{S}(T) = \prod_{i=0}^{q} C_i - 1$, then T is the maximal DT that uses all the possible splits, and thus the only remaining action to take at T is the terminal action, i.e. $\mathcal{A}(T) = \{\overline{a}\}$. Therefore, for any policy π, we have $\pi(T) = \overline{a}$, which yields the result of the proposition with $T^{\pi}(T) = T$ and $\tau = 1$

Now suppose that the result holds for all DTs satisfying $\mathcal{S}(T) \geq m$ where $m \leq \prod_{i=1}^{q} -1$ is fixed and let π be a policy.
If $\pi(T) = \overline{a}$ then $T^{\pi}(T) = T$. Otherwise, let $\pi(T) = (l, i)$ be some split action. We recall that $T \xrightarrow{(l,i)} T(l, i)$ according to (3.9). We have $\mathcal{S}(T(l, i)) = \mathcal{S}(T) + 1$ and as such, the inductive hypothesis holds, i.e. there exists $T^{\pi}(l, i)$ non-absorbing and $\tilde{tau} \geq 1$ such that:

$$\begin{cases} \left(\tilde{T}_t\right)_{t=0}^{\infty} \text{ is the trajectory of } \pi \text{ starting from } T(l, i) \\ \tilde{T}_{\tilde{\tau}-1} = T^{\pi}(T(l, i)) \\ \forall t \geq \tilde{\tau} : \tilde{T}_t = \overline{T^{\pi}(l, i)} \end{cases}$$

Define the following:

$$\begin{cases} T_0 = T \\ \forall t \geq \tau : T_t = \tilde{T}_{t-1} \end{cases}$$

Then the result holds for T with $\tau = \tilde{\tau} + 1$ and $T^{\pi}(T) = T^{\pi}(T(l, i))$. This concludes our induction proof. ■

Proposition 3. *For any policy π and state T, the value of π from T satisfies:*

$$\mathcal{V}^{\pi}(T) = -\lambda \left[\mathcal{S}(T^{\pi}(T)) - \mathcal{S}(T)\right] + \mathcal{H}(T^{\pi}(T))$$

In particular $\mathcal{V}^{\pi}(\Omega) = -\lambda \mathcal{S}(T^{\pi}) + \mathcal{H}(T^{\pi}) = \mathcal{H}_{\lambda}(T^{\pi})$.

Proof Using Proposition 2, we write the trajectory of π from T as follows:

$$T = T_0 \xrightarrow[-\lambda]{\pi(T_0)} T_1 \xrightarrow[-\lambda]{\pi(T_1)} \dots \xrightarrow[-\lambda]{\pi(T_{\tau-2})} T_{\tau-1} = T^{\pi}(T) \xrightarrow[\mathcal{H}(T^{\pi}(T))]{\pi(T_{\tau-1})=\overline{a}} \overline{T^{\pi}(T)} \xrightarrow[0]{} \dots$$

Under the arrow of each transition, we wrote the corresponding reward. This yields the following:

$$\mathcal{V}^{\pi}(T) = -\lambda(\tau - 1) + \mathcal{H}(T^{\pi}(T))$$

$\tau - 1$ is the number of splits it took to transition from T to $T^{\pi}(T)$, which is exactly $\mathcal{S}(T^{\pi}(T)) - \mathcal{S}(T)$, and therefore:

$$\mathcal{V}^{\pi}(T) = -\lambda\left[\mathcal{S}(T^{\pi}(T)) - \mathcal{S}(T)\right] + \mathcal{H}(T^{\pi}(T))$$

In particular, for $T = \Omega$, we have $\mathcal{S}(\Omega) = 0$ and thus:

$$\mathcal{V}^{\pi}(\Omega) = -\lambda\mathcal{S}(T^{\pi}) + \mathcal{H}(T^{\pi}) = \mathcal{H}_{\lambda}(T^{\pi})$$

■

Proposition 4 (Purification Bounds)**.** *Let $T = \{l_1, \ldots, l_{|T|}\}$ be a non-absorbing state and without loss of generality, let $(l_1, i) \in \mathcal{A}(T) \setminus \{\overline{a}\}$ be a split action. We define the Purification Bounds estimates with:*

$$\begin{aligned}\mathcal{Q}(T,(l_1,i)) &= -\lambda + \mathbb{P}[l_1(X) = 1] + \sum_{u=2}^{|T|} \max\{\mathcal{H}(l_u), -\lambda + \mathbb{P}[l_u(X) = 1]\} \\ &= -\lambda + \frac{n(l_1)}{n} + \sum_{u=2}^{|T|} \max\left\{\frac{n_{k^*(l_u)}(l_u)}{n}, -\lambda + \frac{n(l_u)}{n}\right\} \qquad (3.14)\\ \mathcal{V}(T) &= \sum_{u=1}^{|T|} \max\{\mathcal{H}(l_u), -\lambda + \mathbb{P}[l_u(X) = 1]\} \\ &= \sum_{u=1}^{|T|} \max\left\{\frac{n_{k^*(l_u)}(l_u)}{n}, -\lambda + \frac{n(l_u)}{n}\right\} \qquad (3.15)\end{aligned}$$

These estimates are upper bounds on $\mathcal{Q}^*(T,(l,i))$ and $\mathcal{V}^*(T)$:

$$\begin{aligned}\mathcal{Q}(T,(l,i)) &\geq \mathcal{Q}^*(T,(l,i)) \\ \mathcal{V}(T) &\geq \mathcal{V}^*(T)\end{aligned}$$

Proof We will first show the result:

$$\mathcal{V}(T) = \mathcal{V}^*(T)$$

then we will deduce the result for the state-action values using the Bellman Equations and Eq. (3.14) and Eq. (3.15). The proof proceeds by induction on $\mathcal{S}\left(T^{\pi^*}(T)\right) - \mathcal{S}(T)$.

If $\mathcal{S}\left(T^{\pi^*}(T)\right) - \mathcal{S}(T) = 0$, then $\pi^*(T) = \overline{a}$ and $\mathcal{V}^*(T) = \mathcal{H}(T) = \sum_{u=1}^{|T|} \mathcal{H}(l_u)$. On the other hand, we have:

$$\begin{aligned}\mathcal{V}(T) &= \sum_{u=1}^{|T|} \max\{\mathcal{H}(l_u), -\lambda + \mathbb{P}[l_u(X) = 1]\} \\ &\geq \sum_{u=1}^{|T|} \mathcal{H}(l_u) = \mathcal{H}(T) = \mathcal{V}^*(T)\end{aligned}$$

Now suppose this holds for all DTs with $\mathcal{S}\left(T^{\pi^*}(T)\right) - \mathcal{S}(T) \leq m$ where $m \geq 0$ is some fixed integer. Now suppose that $\mathcal{S}\left(T^{\pi^*}(T)\right) - \mathcal{S}(T) = m + 1$, then $\pi^*(T)$ is necessarily a split action.

Without loss of generality, suppose that $\pi^*(T) = (l_1, i)$, then:

$$T \xrightarrow{\pi^*(T)} T(l_1, i)$$

On the other hand we have:

$$\mathcal{V}^*(T) = -\lambda + \mathcal{V}^*(T(l_1, i)) \tag{3.16}$$

Since $\mathcal{S}(T(l_1, i)) = \mathcal{S}(T) + 1$ then we get:

$$\mathcal{S}\left(T^{\pi^*}(T)\right) - \mathcal{S}(T(l_1, i)) = m$$

Moreover, since $T \xrightarrow{\pi^*(T)} T(l_1, i)$ then we have $T^{\pi^*}(T) = T^{\pi^*}(T(l_1, i))$, thus yielding:

$$\mathcal{S}\left(T^{\pi^*}(T(l_1, i))\right) - \mathcal{S}(T(l_1, i)) = m$$

Therefore, the inductive hypothesis holds and we get:

$$\mathcal{V}(T(l_1, i)) \geq \mathcal{V}^*(T(l_1, i))$$

From Eq. (3.16), we then get:

$$\begin{aligned}\mathcal{V}^*(T) &\leq -\lambda + \mathcal{V}(T(l_1, i)) \\ &\leq -\lambda + \sum_{u=2}^{|T|} \max\{\mathcal{H}(l_u), -\lambda + \mathbb{P}[l_u(X) = 1]\} + \\ &\quad \sum_{l' \in \mathrm{Ch}(l_1, i)} \max\{\mathcal{H}(l'), -\lambda + \mathbb{P}[l'(X) = 1]\}\end{aligned}$$

On the other hand we have:

$$\mathcal{V}(T) = \max\{\mathcal{H}(l_1), -\lambda + \mathbb{P}[l_1(X) = 1]\} + \sum_{u=2}^{|T|} \max\{\mathcal{H}(l_u), -\lambda + \mathbb{P}[l_u(X) = 1]\}$$

This yields:

$$\mathcal{V}(T) - \mathcal{V}^*(T) \geq \mathbb{P}[l_1(X) = 1] - \sum_{l' \in \mathrm{Ch}(l_1, i)} \max\{\mathcal{H}(l'), -\lambda + \mathbb{P}[l'(X) = 1]\}$$

We write:

$$\mathbb{P}[l_1(X) = 1] = \sum_{l' \in \mathrm{Ch}(l_1, i)} \mathbb{P}[l'(X) = 1] \tag{3.17}$$

For each child $l' \in \mathrm{Ch}(l_1, i)$ we have:

$$\begin{cases}\mathbb{P}[l'(X) = 1] > -\lambda + \mathbb{P}[l'(X) = 1] \\ \mathbb{P}[l'(X) = 1] \geq \mathbb{P}[l'(X) = 1, k^*(l') = Y] = \mathcal{H}(l')\end{cases}$$

Thus implying:

$$\forall l' \in \mathrm{Ch}(l_1, i) : \mathbb{P}[l'(X) = 1] \geq \max\{\mathcal{H}(l'), -\lambda + \mathbb{P}[l'(X) = 1]\}$$

Using this with Eq. (3.17) yields:

$$\begin{aligned}&\mathbb{P}[l'(X) = 1] \geq \sum_{l' \in \mathrm{Ch}(l_1, i)} \max\{\mathcal{H}(l'), -\lambda + \mathbb{P}[l'(X) = 1]\} \\ &\implies \mathcal{V}(T) \geq \mathcal{V}^*(T)\end{aligned} \tag{3.18}$$

This concludes our proof by induction for the value function. Let us now show the result for the state-action values.

$$\mathcal{Q}^*\left(T,(l_1,i)\right) = -\lambda + \mathcal{V}^*\left(T\left(l_1,i\right)\right) \leq -\lambda + \mathcal{V}\left(T\left(l_1,i\right)\right)$$

On the other hand we have:

$$\mathcal{V}\left(T\left(l_1,i\right)\right) \leq \sum_{u=2}^{|T|} \max\{\mathcal{H}\left(l_u\right), -\lambda + \mathbb{P}\left[l_u\left(X\right)=1\right]\} + \underbrace{\sum_{l' \in \mathrm{Ch}(l_1,i)} \max\{\mathcal{H}\left(l'\right), -\lambda + \mathbb{P}\left[l'\left(X\right)=1\right]\}}_{\leq \mathbb{P}[l_1(X)=1] \text{ according to Eq. (3.18)}}$$

Thus according to Eq. (3.14), we have:

$$\begin{aligned} & \mathcal{V}\left(T\left(l_1,i\right)\right) \leq \lambda + \mathcal{Q}\left(T,(l_1,i)\right) \\ \implies & \mathcal{Q}^*\left(T,(l_1,i)\right) \leq -\lambda + \mathcal{V}\left(T\left(l_1,i\right)\right) \leq \mathcal{Q}\left(T,(l_1,i)\right) \end{aligned}$$

which concludes our proof. ■

Chapter 4

Mining Optimal Sparse Decision Trees from the Space of Branches

Contents

4.1 Introduction . . . 65
4.2 Preliminaries . . . 66
4.3 The Algorithm: Branches . . . 69
4.4 Theoretical Analysis . . . 74
4.5 Implementation Details . . . 76
4.6 Branches vs GOSDT . . . 80
4.7 Experiments . . . 82
4.8 Drawbacks of Binary (One-Hot) Encoding . . . 94
4.9 Conclusion . . . 98
4.10 Appendix: Table of Notation . . . 101
4.11 Appendix: Proofs . . . 102
4.12 Appendix: Auxiliary Procedures . . . 113
4.13 Appendix: Experiments on depth analysis . . . 115

4.1 Introduction

In the last chapter, we introduced DT-Miner but showed that it frequently performs worse than C++ implementations such as GOSDT. This is mainly due DT-Miner's inability to take full advantage of the properties that the Decision Tree optimisation problem exhibits. This chapter's purpose is to address this limitation via a fine-grained MDP formulation that allows our Dynamic Programming procedure to reuse information more efficiently. The resulting algorithm is called Branches and it is the main contribution of this thesis.

Branches is a DP and B&B algorithm that can be viewed as an AO* algorithm (Martelli and Montanari, 1978; Nilsson, 2014) solving an AND/OR graph search problem on the space of DTs. Among the state of the art algorithms, GOSDT is the closest to Branches because they are both BFS methods operating at the level of branches. However, Branches differs from GOSDT on a number of crucial points that we will discuss in depth in Section 4.6. We present below a brief summary of the differences of Branches and GOSDT and the contributions of this chapter:

- Unlike GOSDT, Branches supports non-binary DTs that stem from an ordinal encoding of the data. As we shall explain in Section 4.8 and illustrate in the experiments of Section 4.7, handling non-binary DTs leads to a significant decrease in runtimes and thus yields better scalability compared to methods that require a prior binary encoding.

- The purification bound employed by Branches is an adaptation of DT-Miner's purification bound to the more refined state space of our new MDP. In the context of Branches, this bound bears similarities with (Lin et al., 2020, bounds (9) and (10)) as they both stem from a similar reasoning which we call purification and explain later when discussing the intuition behind Proposition 7. (Lin et al., 2020, Bounds (9) and (10)) have the additional benefit of handling equivalent points, a situation that arises when the dataset includes duplicates with different classes. As of the current Branches' implementation, the purification bound does not support equivalent points but this extension will be incorporated in the future. On the other hand, while (Lin et al., 2020, bounds (9) and (10)) are specialised for binary classification with binary features and a penalty on the number of leaves, Branches' purification bound is general to multiclass classification with categorical features having any number of categories per feature, and it is formulated for the setting where the number of splits is penalised instead of the number of leaves.

- Branches' caching procedure is per branch while GOSDT's is per support set (the subset of data contained in a branch). The reason for this choice is that we believe the support set to be insufficient for defining a branch because branches with different number of splits can still share the same support set. This is a problem for sparsity because we can potentially end up mapping a support set to a more complex branch and thus yielding a suboptimal solution. In our experiments, GOSDT did not run into any suboptimality issues. Nevertheless, in the absence of a proof of optimality with support set caching (to our knowledge), we prefer to restrain Branches' caching for now to branches as it is safer. On this note, MurTree and STreeD support both caching procedures.

- Branches' search strategy allows for more flexibility in the Selection and Backpropagation steps as we will see in Section 4.3.2 and Section 4.5. This flexibility induces significantly faster termination in terms of the number of iterations and a better anytime behaviour. Moreover, it also allows us to clearly outline a future promising parallelisation implementation.

- We perform a computational complexity analysis of BRANCHES by bounding the number of branch evaluations before reaching termination (Theorem 10 and Corollary 11). To our knowledge, only Hu et al. (2019) perform such analysis in their Theorem E.2. In Section 4.4 we compare the two bounds and showcase that ours is substantially smaller.
- In Section 4.7 we perform three sets of experiments. The first one compares pareto fronts of optimal sparse DTs to greedy DTs and optimal DTs (subject to constraints). The second one compares the ability of different optimal sparse DT methods at achieving optimality for a large maximum depth, the aim being to show the robustness of BFS (specifically BRANCHES) to large maximum depths compared to DFS approaches. The last experiments compare the performance of optimal sparse DT methods for various values of the maximum depth parameter.

It is worth noting that BRANCHES is currently implemented in Python, yet it still manages to outperform its C++ competitors on a variety of experiments and metrics. When compared to Python implementations from the literature, specifically OSDT and PyGOSDT, BRANCHES outperforms them by a large margin.

Lastly, we note that this chapter presents BRANCHES under the MDP formalism, which is how we formulated the algorithm in the beginning. We later discovered the existence of literature from the 70's and 80's on AND/OR graph search and AO* (Montanari et al., 1975; Martelli and Montanari, 1978; Pearl, 1984; Nilsson, 2014) that describe equivalent notions and techniques to the ones we developed for BRANCHES. In fact, BRANCHES is an instantiation of AO* for our specific state space and with our Purification Bound acting as a heuristic function (in the specific definition of heuristic employed in this literature). These two formalisms of BRANCHES, MDP and AO*, are equivalent and describe the same algorithm. In this chapter, we keep the MDP formalism to unify the presentation of all our works in this thesis, nevertheless, we will be referring to the equivalent notions from the AO* formalism whenever possible. For the AO* formalism of BRANCHES, refer to our paper (Chaouki et al., 2025).

In this context, we note that Martelli and Montanari (1978) employed AO* (with the early name HS) to seeking optimal DTs, but in a cost-sensitivity context (Lomax and Vadera, 2013), which is distinct from solving for sparsity. Another difference is that the authors sought DTs that perfectly classify a dataset while we seek DTs on the pareto front jointly maximising accuracy and minimising the number of splits. Verhaeghe et al. (2020) also employ an AND/OR formulation, but within a Constraint-Programming (CP) paradigm. The induced CP algorithm solves a similar problem to DL8.5 where the maximum DT depth is constrained instead of solving for sparsity. In an empirical comparison, Aglin et al. (2020) thoroughly showed that DL8.5 outperforms CP.

4.2 Preliminaries

In this section, we keep the notions and notation we introduced in Section 3.2.1 and we add a more general definition of Decision Trees. Let l be a branch, a sub-DT rooted in l is a collection of branches $T = \{l_1, \dots, l_{|T|}\}$ that stems from successive splits starting from from l, we denote $\mathcal{S}(T)$ the number of these splits. Intuitively, $\mathcal{S}(T)$ is the number of *internal nodes* in T. Fig. 4.1 provides an example of a sub-DT. T partitions l in the following sense:

$$\begin{cases} l = \bigvee_{u=1}^{|T|} l_u \\ \forall u, u' \in \{1, \dots, |T|\} : u \neq u' \implies l_u \wedge l_{u'} = 0 \end{cases}$$

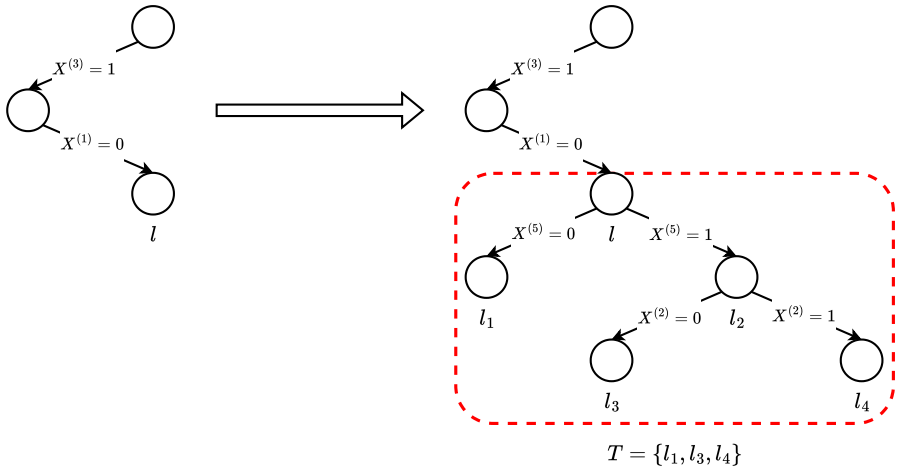


Figure 4.1: Consider a feature space with five binary features $X^{(1)}, X^{(2)}, X^{(3)}, X^{(4)}, X^{(5)} \in \{0,1\}$. The figure provides an example of a sub-DT $T = \{l_1, l_3, l_4\}$ rooted in l that stems from splitting branch l with respect to feature $X^{(5)}$ and splitting branch l_2 with respect to feature $X^{(2)}$, the red perimeter emphasises the fact that T is rooted in l. Here $\mathcal{S}(T) = 2, l = \mathbb{1}\{X^{(3)} = 1\} \wedge \mathbb{1}\{X^{(1)} = 0\}, l_1 = l \wedge \mathbb{1}\{X^{(5)} = 0\}, l_2 = l \wedge \mathbb{1}\{X^{(5)} = 1\}, l_3 = l_2 \wedge \mathbb{1}\{X^{(2)} = 0\}, l_4 = l_2 \wedge \mathbb{1}\{X^{(2)} = 1\}$.

For any datum X in l, which we recall means that $l(X) = 1$, T predicts the majority class of the branch l_u containing X:

$$T(X) = \sum_{u=1}^{|T|} l_u(X)\, k^*(l_u)$$

The probability that a datum is in l and is correctly classified by T is:

$$\mathcal{H}(T) = \mathbb{P}[l(X) = 1, T(X) = Y] = \sum_{u=1}^{|T|} \mathcal{H}(l_u)$$

where the additivity is due to $\{l_1, \ldots, l_{|T|}\}$ forming a partition of l.

A DT is a sub-DT that is rooted in the root Ω. Let T be a DT, since $\Omega(X) = 1$ for any datum X, then:

$$\underbrace{\mathcal{H}(T) = \mathbb{P}[\Omega(X) = 1, T(X) = Y] = \mathbb{P}[T(X) = Y]}_{\text{Accuracy}}$$

In similar fashion to Chapter 3, we consider the regularised objective:

$$\mathcal{H}_\lambda(T) = -\lambda \mathcal{S}(T) + \mathcal{H}(T)$$

The subtle difference is that, in Chapter 3, we defined $\mathcal{H}_\lambda(T)$ for DTs only. Here it is generalised to sub-DTs as well.

4.2.1 Markov Decision Process (MDP)

BRANCHES differs from DT-MINER in the MDP formulation. The more refined MDP formulation of BRANCHES pertains to generalising the definition of the state space to all sub-DTs rather than just DTs. Below is BRANCHES' undiscounted MDP.

State space: The set of all possible sub-DTs. A state with only one branch $T = \{l\}$ is called a unit-state. We make the notation lighter by just denoting it l. The initial state is the root Ω.

Action space: $\mathcal{A}(l)$ is the set of permissible actions in a unit-state l. There are two types of actions:

- The terminal action $\overline{a}$. It transitions from l to an absorbing state $\overline{l}$. We denote $l \xrightarrow{\overline{a}} \overline{l}$.
- Split actions. Consider a unit-state $l = \bigwedge_{v=1}^{\mathcal{S}(l)} \mathbb{1}\{X^{(i_v)} = j_v\}$, the set of possible split actions at l is $\{1, \dots, q\} \setminus \{i_1, \dots, i_{\mathcal{S}(l)}\}$. Let i be a split action, taking i transitions l to the sub-DT rooted in l that is formed by the children $T = \text{Ch}(l, i)$, where $\text{Ch}(l, i)$ is defined in Eq. (3.2). We denote this transition $l \xrightarrow{i} T = \text{Ch}(l, i)$.

The set of all permissible actions at l is therefore $\mathcal{A}(l) = \{\overline{a}\} \cup \{1, \dots, q\} \setminus \{i_1, \dots, i_{\mathcal{S}(l)}\}$. When $\mathcal{S}(l) = q$, i.e. the clauses of l include all the features, then $\mathcal{A}(l) = \{\overline{a}\}$ and we can only transition to $\overline{l}$. We now define the set of permissible actions for any state $T = \{l_1, \dots l_{|T|}\}$ as $\mathcal{A}(T) = \mathcal{A}(l_1) \times \dots \times \mathcal{A}\left(l_{|T|}\right)$. Taking action $a = \left(a_1, \dots, a_{|T|}\right) \in \mathcal{A}(T)$ in T is equivalent to taking each action a_u in l_u for $1 \leq u \leq |T|$, and transitioning to the corresponding sub-DT:

$$\begin{cases} T \xrightarrow{a} T' = \bigcup_{u=1}^{|T|} T'_u \\ \forall u \in \{1, \dots, |T|\} : l_u \xrightarrow{a_u} T'_u \end{cases}$$

Reward function: Let l be a unit-state and $a \in \mathcal{A}(l)$. We denote $r(l, a)$ the reward of taking action a in l.

- If a is a split action, then $r(l, a) = -\lambda$ regardless of l (except if l is absorbing).
- If $a = \overline{a}$, then $r(l, \overline{a}) = \mathcal{H}(l)$.
- If $l = \overline{l}$, i.e. l is an absorbing state, then $r\left(\overline{l}, a\right) = 0$.

For any state $T = \{l_1, \dots, l_{|T|}\}$ and action $a = \left(a_1, \dots, a_{|T|}\right) \in \mathcal{A}(T)$, the reward $r(T, a)$ is equal to the sum of the rewards of taking each action a_u in l_u for $u \in \{1, \dots |T|\}$:

$$r(T, a) = \sum_{u=1}^{|T|} r(l_u, a_u)$$

A policy π maps each state T to one of its actions $\pi(T) \in \mathcal{A}(T)$. The trajectory of π starting from T is the sequence of states $(T_t)_{t=0}^{\infty}$ such that $T_0 = T$ and $\forall t \geq 0 : T_t \xrightarrow{\pi(T_t)} T_{t+1}$. Then, as we discussed in Section 3.2.3, the value of π at T is:

$$\mathcal{V}^{\pi}(T) = \sum_{t=0}^{\infty} r(T_t, \pi(T_t))$$

Proposition 5 ensures that all policies have finite returns.

Proposition 5. *For any policy π and unit-state l, there exists $\tau \geq 0$ such that for any $t \geq \tau$, $T_t = \{\overline{l_1}, \dots, \overline{l_{|T_\tau|}}\}$ is an absorbing state. In which case we call $T_l^{\pi} = \{l_1, \dots, l_{|T_\tau|}\}$ the sub-DT of π rooted in l. If $l = \Omega$ we abbreviate the notation $T_{\Omega}^{\pi} \equiv T^{\pi}$ and call T^{π} the DT of π.*

Proposition 6 links the DT optimisation problem to solving the MDP.

Proposition 6. *For any policy π and unit-state l, the value of π from l satisfies:*

$$\mathcal{V}^{\pi}(l) = \mathcal{H}_{\lambda}(T_l^{\pi}) = -\lambda \mathcal{S}(T_l^{\pi}) + \mathcal{H}(T_l^{\pi})$$

In particular $\mathcal{V}^{\pi}(\Omega) = \mathcal{H}_{\lambda}(T^{\pi}) = -\lambda \mathcal{S}(T^{\pi}) + \mathcal{H}(T^{\pi})$.

Since any DT T is constructed with successive splits starting from Ω, there always exists a policy π such that $T^{\pi} = T$, and therefore $\mathcal{V}^{\pi}(\Omega) = \mathcal{H}_{\lambda}(T)$. This result provides the equivalence between finding the optimal DT and the optimal policy:

$$T^* = \text{Argmax}_T\{\mathcal{H}_{\lambda}(T)\},\ \pi^* = \text{Argmax}_{\pi}\{\mathcal{V}^{\pi}(\Omega)\}$$

in which case the optimal DT is the DT of π^*, i.e. $T^* = T^{\pi^*}$. To conclude this section, our objective now is to find π^* and then deduce T^* as T^{π^*}. We abbreviate the notation $\mathcal{V}^{\pi^*} \equiv \mathcal{V}^*$.

4.3 The Algorithm: Branches

BRANCHES is a search algorithm that can be viewed as an instantiation of AO* with heuristic values defined as upper bounds on the optimal state-action values. The next sections describe how BRANCHES defines these upper bounds along with its different algorithmic steps.

4.3.1 Estimating the optimal state-action values $\mathcal{Q}^*(T, a)$

Let l be a non-absorbing unit-state and $a \in \mathcal{A}(l)$, we denote $\mathcal{Q}(l, a)$ the estimate of $\mathcal{Q}^*(l, a)$, where $\mathcal{Q}^* \equiv \mathcal{Q}^{\pi^*}$, and we recall the definition (2.4) of the state-action values. For the terminal action $\overline{a}$, $\mathcal{Q}^*(l, \overline{a})$ is directly accessible from the data via:

$$\mathcal{Q}(l, \overline{a}) = \mathcal{Q}^*(l, \overline{a}) = r(l, \overline{a}) = \mathcal{H}(l) = \frac{n_{k^*(l)}(l)}{n} \tag{4.1}$$

For a split action $a \in \mathcal{A}(l) \setminus \{\overline{a}\}$, such that $l \xrightarrow{a} T = \{l_1, \ldots, l_{|T|}\}$, the Bellman equations, introduced in Proposition 1, yield:

$$\mathcal{Q}^*(l, a) = -\lambda + \mathcal{V}^*(T) = -\lambda + \sum_{u=1}^{|T|} \mathcal{V}^*(l_u)$$
$$\mathcal{V}^*(l) = \mathcal{Q}^*(l, \pi^*(l)) = \max_{a \in \mathcal{A}(l)} \mathcal{Q}^*(l, a)$$

thus suggesting the following recursive definitions of the estimates:

$$\mathcal{Q}(l, a) = -\lambda + \sum_{u=1}^{|T|} \mathcal{V}(l_u) \tag{4.2}$$
$$\forall u \in \{1, \ldots, |T|\} : \mathcal{V}(l_u) = \max_{a \in \mathcal{A}(l_u)} \mathcal{Q}(l_u, a) \tag{4.3}$$

The estimate $\mathcal{Q}(l, a)$ in Eq. (4.2) can only be calculated if the estimates $\mathcal{V}(l_u)$ in Eq. (4.3) are available. Otherwise we initialise $\mathcal{Q}(l, a)$ with Eq. (4.4) according to Proposition 7.

Proposition 7 (Purification Bound)**.** *For any non-absorbing unit-state l and split action $a \in \mathcal{A}(l) \setminus \{\overline{a}\}$, we define the Purification Bound estimates:*

$$\mathcal{Q}(l,a) = -\lambda + \mathbb{P}[l(X) = 1] = -\lambda + \frac{n(l)}{n} \tag{4.4}$$

$$\mathcal{V}(l) = \max\{\mathcal{H}(l), -\lambda + \mathbb{P}[l(X) = 1]\} = \max\left\{\frac{n_{k^*(l)}(l)}{n}, -\lambda + \frac{n(l)}{n}\right\} \tag{4.5}$$

Then the estimates $\mathcal{Q}(l,a)$ and $\mathcal{V}(l)$ are upper bounds on $\mathcal{Q}^*(l,a)$ and $\mathcal{V}^*(l)$ respectively.

If the split action a yields $l \xrightarrow{a} T = \{l_1, \dots, l_{|T|}\}$ such that all the data in the resulting children branches l_u are correctly classified (in which case, the branches l_u are called pure), then:

$$\mathcal{Q}^*(l,a) = -\lambda + \mathcal{H}(T) = -\lambda + \mathbb{P}[T(X) = Y, l(X) = 1] = -\lambda + \mathbb{P}[l(X) = 1]$$

Thus the bound (4.4) coincides exactly with the optimal state-action value of an action that *purifies* l (if it exists), hence the name *Purification Bound*.

Now we can straightforwardly define $\mathcal{Q}(T,a)$ for any state T that is not necessarily a unit-state. Let $T = \{l_1, \dots, l_{|T|}\}$ be such a state and consider $a = \left(a_1, \dots, a_{|T|}\right) \in \mathcal{A}(T)$ such that:

$$\begin{cases} T \xrightarrow{a} T' = \bigcup_{u=1}^{|T|} T'_u \\ \forall u \in \{1, \dots, |T|\} : l_u \xrightarrow{a_u} T'_u \end{cases}$$

Then we have the following:

$$\begin{aligned} \mathcal{Q}^*(T,a) = r(T,a) + \mathcal{V}^*(T') &= \sum_{u=1}^{|T|} r(l_u, a_u) + \sum_{u=1}^{|T|} \mathcal{V}^*(T'_u) \\ &= \sum_{u=1}^{|T|} \left(r(l_u, a_u) + \mathcal{V}^*(T'_u)\right) = \sum_{u=1}^{|T|} \mathcal{Q}^*(l_u, a_u) \end{aligned}$$

Therefore, we can define the estimate $\mathcal{Q}(T,a)$ directly with:

$$\mathcal{Q}(T,a) = \sum_{u=1}^{|T|} \mathcal{Q}(l_u, a_u) \tag{4.6}$$

Summary: For any unit-state l, the estimate $\mathcal{Q}(l,\overline{a})$ for the terminal action $\overline{a}$ is known in advance and calculated with Eq. (4.1). For any split action $a \in \mathcal{A}(l) \setminus \{\overline{a}\}$, $\mathcal{Q}(l,a)$ is calculated with Eq. (4.2) when estimates for the children are available, otherwise it is initialised with Eq. (4.4). For a general state T, the estimate is deduced straightforwardly via Eq. (4.6).

As was the case for DT-Miner, it is not sufficient to estimate the optimal state-action values; we also need to specify a search strategy to decide which state-action pairs to estimate in the hope of achieving fast optimal convergence. The next section introduces Branches' search strategy.

4.3.2 The Search strategy

Initially, all the non-terminal unit-states l are labelled as unvisited and incomplete, by incomplete we mean that $\mathcal{V}^*(l)$ are still unknown. The absorbing states are labelled as complete on the

Algorithm 1 BRANCHES

Input: Dataset $\mathcal{D} = \{(X_m, Y_m)\}_{m=1}^{n}$, penalty parameter $\lambda \geq 0$.
memo $\leftarrow \{\}$ ▷ Initialise an empty memo
INITIALISE$(\Omega, \mathcal{D})$
while not Ω.complete **do**
    $(l, \text{path}) \leftarrow$ SELECT()
    if l.complete **then**
        BACKPROPAGATE(path)
    else
        EXPAND$(l, \mathcal{D})$
        BACKPROPAGATE(path)
    end if
end while
return INFER()

other hand because their optimal values are known to be 0. Moreover, the state-action pairs (l, a) (for non-absorbing unit-states l) are also labelled as incomplete since we do not know $\mathcal{Q}^*(l, a)$ either. We initialise an empty memo where the encountered state values estimates $\mathcal{V}(l)$ are stored. Each iteration of BRANCHES follows the pipeline below:

- **Selection:** Initialise an empty list *path*. Starting from the root $l = \Omega$, choose the action maximising the optimal state-action value estimate:

$$a = \text{Argmax}_{a' \in \mathcal{A}(l)} \mathcal{Q}(l, a')$$

Append (l, a) to *path* and transition $l \xrightarrow{a} T = \{l_1, \ldots, l_{|T|}\}$. Choose an incomplete unit-state $l_u \in T$ and make it the current state $l = l_u$. This choice can be arbitrary or according to some heuristic. Repeat this process until reaching an unvisited or absorbing unit-state l. Note that the *path* list does not include this final state l. This step is equivalent to steps 4 and 5 in (Nilsson, 2014, p.104 Procedure AO*). Regarding the choice of an incomplete branch, we use a heuristic that chooses branch l of lowest $\mathcal{H}_\lambda\left(\widehat{T}_l\right)$ where $\widehat{T}_l$ is the best sub-DT rooted in l found so far in terms of the objective $\mathcal{H}_\lambda$. The reason is to quickly get rid of branches that contribute low $\mathcal{H}_\lambda\left(\widehat{T}_l\right)$ to the solution. This either yields better subtrees rooted in these branches, or bad subtrees in which case we would quickly understand that this region of the search space is unpromising. A similar idea was proposed in (Nilsson, 2014, p.107):

> *Possibly the expansion of that leaf node having the highest h value would most likely result in a changed estimate.*

The h value in (Nilsson, 2014, p.107) refers to the heuristic value, which is akin to our $\mathcal{V}$ value albeit for a minimisation context instead of a maximisation one. As such, a direct application of (Nilsson, 2014, p.107) to our setting would yield choosing the incomplete branch l with lowest $\mathcal{V}(l)$. However, we noticed in practice that employing $\mathcal{H}_\lambda\left(\widehat{T}_l\right)$ instead yields better anytime behaviour for the reasons explained above. Currently, we have not investigated other strategies of this type, and it is a promising idea for future work. GOSDT employs a global priority queue that does not support this type of selection process.

- **Expansion:** If l is complete, then we move to the Backpropagation step. Otherwise we calculate $\mathcal{Q}(l, a)$ for all $a \in \mathcal{A}(l)$ as explained below.

  For the terminal action, we set $\mathcal{Q}(l, \overline{a}) = \mathcal{H}(l)$ as per Eq. (4.1) and we label $(l, \overline{a})$ as complete. For any split action $a \in \mathcal{A}(l) \setminus \{\overline{a}\}$, let $l \xrightarrow{a} T = \{l_1, \ldots, l_{|T|}\}$. We calculate $\mathcal{Q}(l, a)$ according to Eq. (4.2):

$$\mathcal{Q}(l, a) = -\lambda + \sum_{u=1}^{|T|} \mathcal{V}(l_u)$$

  where for each $l_u \in T$, $\mathcal{V}(l_u)$ is retrieved from the memo in case l_u is labelled as visited, otherwise $\mathcal{V}(l_u)$ is initialised with Eq. (4.5):

$$\mathcal{V}(l_u) = \max\left\{\frac{n_{k^*(l_u)}(l_u)}{n}, -\lambda + \frac{n(l_u)}{n}\right\}$$

  If $\mathcal{V}(l_u) = \frac{n_{k^*(l_u)}(l_u)}{n}$ then l_u is complete and we know that $\mathcal{V}^*(l_u) = \mathcal{V}(l_u) = \frac{n_{k^*(l_u)}(l_u)}{n}$. If all children l_u are complete, then we label (l, a) as complete and we have $\mathcal{Q}(l, a) = \mathcal{Q}^*(l, a)$. Once we have calculated $\mathcal{Q}(l, a)$ for all actions $a \in \mathcal{A}(l)$, we deduce the state value estimate of l as follows:

$$\mathcal{V}(l) = \max_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a)$$

  If $a^* = \text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a)$ is such that (l, a^*) is complete, then we label l as complete and we have $\mathcal{V}^*(l) = \mathcal{V}(l) = \mathcal{Q}(l, a^*) = \mathcal{Q}^*(l, a^*)$. This step corresponds to step 6 in (Nilsson, 2014, p.104 Procedure AO*).

- **Backpropagation:** Update $\mathcal{Q}(l, a)$ and $\mathcal{V}(l)$ for all (l, a) in *path* via Backward recursion. For $j = length(path) - 1, \ldots, 0$, let $(l, a) = path[j]$ with $l \xrightarrow{a} T = \{l_1, \ldots, l_{|T|}\}$, then we update $\mathcal{Q}(l, a)$ and $\mathcal{V}(l)$ with Eq. (4.2) and Eq. (4.3) respectively:

$$\mathcal{Q}(l, a) = -\lambda + \sum_{u=1}^{|T|} \mathcal{V}(l_u)$$
$$\mathcal{V}(l) = \max_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a)$$

  We update $\mathcal{V}(l)$ in the memo. If all children l_u are complete, we label (l, a) as complete. If $a^* = \text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a)$ is such that (l, a^*) is complete, then we label l as complete. This step corresponds to steps 7-12 in (Nilsson, 2014, p.104 Procedure AO*), which bear the names bottom-up, cost-revision, connector-marking. We note that, contrary to GOSDT, our backpropagation step only updates estimates along the encountered branches in *path*, even though there exist different paths that could lead to the selected branch, and which as a result could be updated as well. This choice is motivated by computational concerns. Indeed, the number of paths leading to the selected branch is equal to the number of permutations of the splits constituting the branch, which grows exponentially with $\mathcal{S}(l)$. Moreover, there are many branches along these paths that do not necessitate updates because they will be pruned pre-emptively. In fact, those that need updates will inevitably be visited in subsequent selection steps of Branches, and their estimated values will be aggressively decreased by then because many of their subsequent descendents in the search graph would have already been updated. This idea of restricting the backpropagation step is discussed in (Nilsson, 2014, p. 106):

Therefore, not all ancestors need have cost revisions, but only those ancestors having best partial solution graphs containing descendants with revised costs (hence step 12).

BRANCHES terminates when the root Ω becomes complete, upon which Theorem 8 proves the optimality of the retrieved solution.

Theorem 8 (Optimality of BRANCHES)**.** *When BRANCHES terminates, the optimal policy is the greedy policy with respect to the estimated state-action values $\mathcal{Q}(l,a)$, i.e.*

$$\pi^{*}(T) = \text{Argmax}_{a \in \mathcal{A}(T)} \mathcal{Q}(T,a)$$

Algorithm 1 summarises these steps. The INFER() procedure at the end retrieves the optimal Decision Tree based on the fact that $T^{*} = T^{\pi^{*}}$, as we established at the end of Section 4.2.1. All the auxiliary procedures are described in Section 4.12.

4.3.3 Anytime behaviour

There are settings where the search takes a substantial or even unfeasible amount of time to find the optimal sparse DT. In these situations, we set a time limit on BRANCHES' execution and even if it does not terminate within this time, we expect it to still return a good solution. This property is called the anytime behaviour. For BFS methods, it is straightforward to implement while it is less trivial for DFS algorithms. STreeD for example, as of v.1.3.1, does not enjoy the anytime property.

We formulate BRANCHES' anytime behaviour within our MDP framework as follows. For each branch l that is currently in the memo, we define its greedy value $\widehat{\mathcal{V}}(l)$ recursively as:

$$\widehat{\mathcal{V}}(l) = \begin{cases} \mathcal{H}(l) \text{ if } l \text{ has no children in the memo} \\ \max_{a \in \mathcal{A}(l) \setminus \{\overline{a}\}} \left\{ \mathcal{H}(l), -\lambda + \sum_{l' \in \text{Ch}(l,a)} \widehat{\mathcal{V}}(l') \right\} \end{cases} \tag{4.7}$$

Basically $\widehat{\mathcal{V}}(l)$ is the best value of the regularised objective found so far for all sub-DTs rooted in l. With the notation introduced in the Selection step in Section 4.3.2, we have $\widehat{\mathcal{V}}(l) = \mathcal{H}_{\lambda}\left(\widehat{T}_{l}\right)$. Based on this definition, we also introduce the greedy state-action values and the greedy policy:

$$\widehat{\mathcal{Q}}(l,a) = \begin{cases} \mathcal{H}(l) \text{ if } a = \overline{a} \\ -\lambda + \sum_{l' \in \text{Ch}(l,a)} \widehat{\mathcal{V}}(l') \text{ if } a \in \mathcal{A}(l) \setminus \{\overline{a}\} \end{cases}$$
$$\widehat{\pi}(l) = \text{Argmax}_{a \in \mathcal{A}(l)} \widehat{\mathcal{Q}}(l,a)$$

The estimates $\widehat{\mathcal{V}}(l)$ and $\widehat{\mathcal{Q}}(l,a)$ are updated in similar fashion to $\mathcal{V}(l)$ and $\mathcal{Q}(l,a)$ during the Backpropagation step. When BRANCHES reaches timeout we have $\widehat{T}_{\Omega} = T_{\Omega}^{\widehat{\pi}}$, which means that the best DT found so far (in terms of the regularised objective $\mathcal{H}_{\lambda}$) is the DT of $\widehat{\pi}$. As a consequence, we can unroll $\widehat{\pi}$ starting from Ω to get $\widehat{T}_{\Omega}$, this is the proposed solution by BRANCHES when it runs out of time.

Unfortunately, unlike settings where BRANCHES terminates, the anytime behaviour does not provide any theoretical guarantees with respect to the quality of the proposed solution. Such a guarantee necessitates the assumption of some smoothness properties of $\mathcal{H}_{\lambda}$ on the space of sub-DTs. Nevertheless, the use of heuristics such as the one we propose in the Selection step can lead to empirically satisfying results for BRANCHES' anytime behaviour as we shall illustrate in our experiments.

4.4 Theoretical Analysis

We established the optimality of BRANCHES in Theorem 8. For a satisfying analysis of the algorithm, we still need to derive guarantees on its computational complexity. **Expansion** is the most costly step, this is due to the potential evaluations $\mathcal{H}(l) = \frac{n_{k^*(l)}(l)}{n}$ at the level of branches. The reason this is costly (comparatively to the other steps) is due to the term $n_{k^*(l)}(l)$, which we recall is the number of examples in l of the majority class $k^*(l)$. Computationally speaking, to calculate $n_{k^*(l)}(l)$, we extract from $\mathcal{D}$ the subset of data contained in l, we calculate the number of examples per class in this subset, and only then we deduce $k^*(l)$ and $n_{k^*(l)}(l)$. By contrast with **Expansion**, the **Selection** step consists in just choosing $a = \text{Argmax}_{a' \in \mathcal{A}(l)} \mathcal{Q}(l, a')$ each time, which we do efficiently by storing the estimates in heap queues, we explain this in detail in Section 4.5. As for **Backpropagation**, it consists of straightforward updates of the estimates along the list *path*, these updates only involve sums and maximums over few terms, of which there are at most $\max_{1 \leq i \leq q} C_i$, the maximum number of categories a feature can take. With these remarks, the overall number of evaluations $\mathcal{H}(l)$, that BRANCHES makes before terminating, constitutes an accurate evaluation of the algorithm's computational complexity. In the DP and B&B literature, we are only aware of (Hu et al., 2019, Theorem E.2) and GOSDT (Lin et al., 2020, Theorem H.1) as attempts at performing a similar analysis, albeit the latter only provides the number of binary DTs that can be constructed from a set of binary features instead of a bound on the number of branch evaluations.

Before deriving our results, we note that there is a slight mistake in (Hu et al., 2019, Theorem E.2), the sum should be up to the maximum depth of the optimal DT rather than the maximum number of its leaves, we reformulate the authors' result below:

Theorem 9. *Let $\Gamma_{\text{OSDT}}(q, \lambda)$ denote the total number of evaluations that OSDT performs for an instance of the binary classification problem with $q \geq 2$ binary features and penalty parameter $0 \leq \lambda \leq 1$, then we have:*

$$\Gamma_{\text{OSDT}}(q, \lambda) \leq 1 + \sum_{h=1}^{\kappa} \left\{ N_h + \binom{q}{h} - P(q, h) \right\}$$

Where $P(q, h)$ is the number $h-$permutations of q, N_h is the number of possible binary DTs of depth h defined in (Hu et al., 2019, Formula (1)) and:

$$\kappa = \min \left\{ \left\lfloor \frac{1}{2\lambda} \right\rfloor - 1, q \right\}$$

The difference with (Hu et al., 2019, Theorem E.2) is in the term κ, the authors write it as:

$$\kappa = \min \left\{ \left\lfloor \frac{1}{2\lambda} \right\rfloor, 2^q \right\}$$

The term 2^q is the maximum number of leaves that any DT can have, however, following the authors' reasoning, it should be the maximum possible depth, which is q. Furthermore, the term $\left\lfloor \frac{1}{2\lambda} \right\rfloor$ is an upper bound on the maximum number of leaves the optimal solution can have. Such solution has at most a depth of $\left\lfloor \frac{1}{2\lambda} \right\rfloor - 1$. Indeed, the maximum depth of a DT T with $|T|$ leaves is $|T| - 1$, this corresponds to a DT that only splits one node at each depth (for example, always splitting the right child node), in which case the depth of T is equal to the number of its

Table 4.1: Comparing the complexity bounds of BRANCHES and OSDT for binary features and for different values of λ and q. INF means that the number is so large that it leads to overflow issues.

λ	$q = 10$		$q = 15$		$q = 20$	
	BRANCHES	OSDT	BRANCHES	OSDT	BRANCHES	OSDT
0.1	$\mathbf{5.70 \times 10^4}$	5.61×10^{13}	$\mathbf{5.80 \times 10^5}$	6.86×10^{16}	$\mathbf{2.82 \times 10^6}$	8.35×10^{18}
0.05	$\mathbf{3.94 \times 10^5}$	7.52×10^{271}	$\mathbf{7.53 \times 10^7}$	1.53×10^{473}	$\mathbf{3.01 \times 10^9}$	5.69×10^{576}
0.01	$\mathbf{3.94 \times 10^5}$	1.64×10^{392}	$\mathbf{1.43 \times 10^8}$	INF	$\mathbf{4.65 \times 10^{10}}$	INF

internal nodes.

We first provide two upper bounds on the number of evaluations of BRANCHES. The first one, in Theorem 10, is problem-dependent and necessitates knowledge of $\mathcal{S}(T^*)$ and $\mathcal{H}(T^*)$. The second bound however, in Corollary 11, is general across all possible problems, it is looser but it can be evaluated without knowledge of $\mathcal{S}(T^*)$ and $\mathcal{H}(T^*)$.

Theorem 10 (Problem-dependent complexity of BRANCHES)**.** *Let $\Gamma(q, C, \lambda)$ denote the total number of branch evaluations performed by BRANCHES for an instance of the classification problem with $q \geq 2$ features, $0 < \lambda \leq 1$ the penalty parameter, and $C \geq 2$ the number of categories per feature. Then, $\Gamma(q, C, \lambda)$ satisfies the following bound:*

$$\Gamma(q, C, \lambda) \leq \sum_{h=0}^{\kappa} (q-h) C^{h+1} \binom{q}{h}; \; \kappa = \min\left\{ \left\lfloor \mathcal{S}(T^*) - 1 + \frac{1 - \mathcal{H}(T^*)}{\lambda} \right\rfloor, q \right\}$$

Corollary 11 (Problem-independent complexity of BRANCHES)**.** *Let $\Gamma(q, \lambda, C)$ be defined as in Theorem 10, then we have:*

$$\Gamma(q, C, \lambda) \leq \sum_{h=0}^{\kappa} (q-h) C^{h+1} \binom{q}{h}; \; \kappa = \min\left\{ \left\lfloor \frac{1}{K\lambda} \right\rfloor - 1, q \right\}$$

Table 4.1 and Fig. 4.2 compare the bounds on the complexities of OSDT and BRANCHES according to Theorem 9 and Corollary 11. We note that we could not make plots for λ values lower than 0.005 because the bounds of OSDT become extremely large. Table 4.1 and Fig. 4.2 clearly show the vast computational gains that our bound offers over OSDT's. We note however, that the immense numbers upper bounding the complexity of OSDT *do not reflect OSDT's true complexity* but rather the fact that the bound is too loose. Indeed, the reasoning behind OSDT's complexity bound pertains to counting all the possible DTs which depths do not exceed the maximum depth of the optimal sparse DT. The analysis does not incorporate OSDT's pruning capacity. In fact, it is unclear how such analysis could be performed with OSDT's bounds. The MDP framework and the purification bound on the other hand offer an intuitive reasoning behind such analysis.

Theorem 10 and Corollary 11 are sufficient to emphasize BRANCHES' computational efficiency. Nonetheless, a more careful analysis yields further tighter bounds, albeit at the cost of more complicated formulae. These tighter bounds are provided in Theorem 12 and Corollary 13.

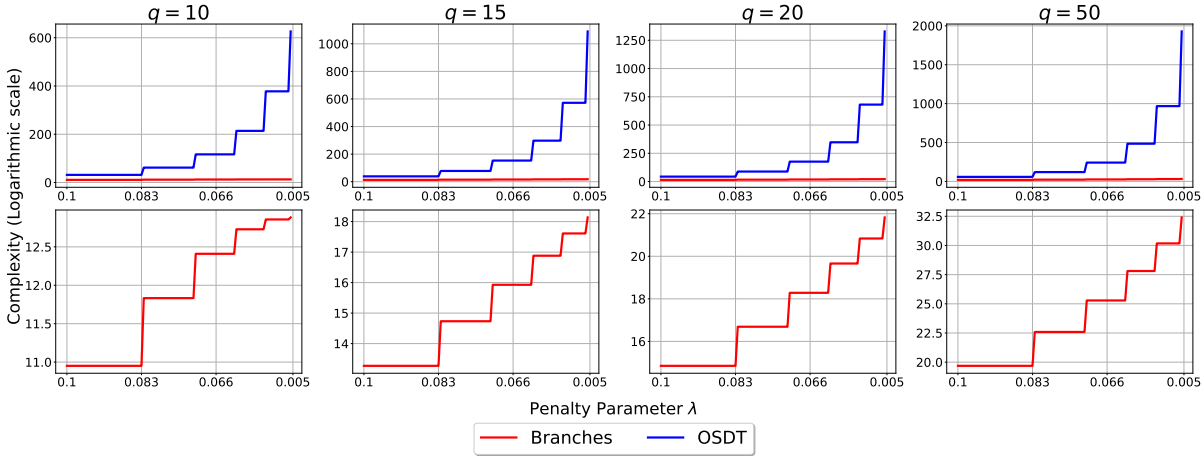


Figure 4.2: Comparing the complexity bounds of BRANCHES and OSDT as a function of λ for different values of q. The complexity is provided in logarithmic scale. We note that the complexity is piecewise constant due to the floor function $\lfloor\rfloor$.

Theorem 12. *Let $\Gamma(q, \lambda, C)$ be defined as in Theorem 10, then we have:*

$$\Gamma(q, C, \lambda) \leq \sum_{h=0}^{\kappa_-} (q-h)\, C^{h+1} \binom{q}{h} + \sum_{h=\kappa_-+1}^{\kappa} (q-h)\,(h-\kappa_-)\, C^h \binom{q}{h}$$

$$\kappa_- = \min\left\{\mathcal{S}(T^*) - C + \left\lfloor \frac{1-\mathcal{H}(T^*)}{\lambda} \right\rfloor, q\right\}$$

$$\kappa = \min\left\{\mathcal{S}(T^*) - 1 + \left\lfloor \frac{1-\mathcal{H}(T^*)}{\lambda} \right\rfloor, q\right\}$$

Corollary 13. *Let $\Gamma(q, \lambda, C)$ be defined as in Theorem 10, then we have:*

$$\Gamma(q, C, \lambda) \leq \sum_{h=0}^{\kappa_-} (q-h)\, C^{h+1} \binom{q}{h} + \sum_{h=\kappa_-+1}^{\kappa} (q-h)\left(h - \left\lfloor \frac{1}{K\lambda} \right\rfloor + C\right) C^h \binom{q}{h}$$

$$\kappa_- = \min\left\{\left\lfloor \frac{1}{K\lambda} \right\rfloor - C, q\right\};\ \kappa = \min\left\{\left\lfloor \frac{1}{K\lambda} \right\rfloor - 1, q\right\}$$

4.5 Implementation Details

The search strategy we introduced in Section 4.3.2 is an abstract description of BRANCHES. In this section, we provide concrete elements for the implementation of the algorithm.

4.5.1 Branch objects

For each branch $l = \bigwedge_{v=1}^{\mathcal{S}(l)} \mathbb{1}\{X^{(i_v)} = j_v\}$, we define an object with the following elements:

- `id_branch`: l is identified with the unique string "$(i_1, j_1)(i_1, j_2)\ldots(i_{\mathcal{S}(l)}, j_{\mathcal{S}(l)})$". We recall that this string is unique because we impose the condition $i_1 < i_2 < \ldots < i_{\mathcal{S}(l)}$. We store

each encountered branch in a memo dictionary using its identifier.

- `attributes_categories`: Dictionary containing the number of categories per unused feature in l. We recall that the set of unused features is the set of split actions.
- `bit_vector`: Vector of the indices of the data contained in l. This vector allows quick access to the data in l.
- `children`: Dictionary containing the children of l, i.e. the set $\mathrm{Ch}(l, i)$ for each unused feature i in l. Initialised with an empty dictionary.
- `attribute_opt`: The greedy action $\widehat{\pi}(l)$ as per the definition in Section 4.3.3. If $\widehat{\pi}(l) = \overline{a}$ then we set `attribute_opt` to `None`.
- `terminal`: Boolean describing whether l is terminal or not, we say that l is terminal if the set of permissible actions at l only includes the terminal action, i.e. $\mathcal{A}(l) = \overline{a}$.
- `complete`: Boolean describing whether l is complete or not.
- `value`: The estimated value $\mathcal{V}(l)$.
- `value_terminal`: The value of the terminal action at l.

$$\mathcal{Q}^*(l, \overline{a}) = \mathcal{H}(l) = \mathbb{P}[l(X) = 1, k^*(l) = Y] = \frac{n_{k^*(l)}(l)}{n}$$

- `value_greedy`: The greedy value $\widehat{\mathcal{V}}(l)$ as per the definition in Section 4.3.3.
- `freq`: Proportion of examples in l:

$$\texttt{freq} = \mathbb{P}[l(X) = 1] = \frac{n(l)}{n} = \frac{1}{n}\sum_{m=1}^{n} l(X_m)$$

- `pred`: Majority class at l:

$$\texttt{pred} = k^*(l) = \mathrm{Argmax}_{1 \leq k \leq K} n_k(l)$$

- `queue`: Heap queue containing `(-value, -value_complete, attribute, children)` tuples. For each unused feature (split action) `attribute`: `value` is the estimate:

$$\texttt{value} = \mathcal{Q}(l, \texttt{attribute}) = -\lambda + \sum_{l' \in \mathrm{Ch}(l, \texttt{attribute})} \mathcal{V}(l')$$

  On the other hand, `value_complete` is the sum of the estimated values $\mathcal{V}(l')$ of the children $l' \in \mathrm{Ch}(l, \texttt{attribute})$ that are complete. By definition, the complete children l' satisfy $\mathcal{V}(l') = \mathcal{V}^*(l')$, we store the sum of their values in `value_complete`, which serves to efficiently update $\mathcal{Q}(l, \texttt{attribute})$ during the Backpropagation step. `children` is a dictionary containing the incomplete children, it is from this dictionary that we choose the next branch to visit during the Selection step. During Backpropagation, If an incomplete branch l' in `children` becomes complete, it is discarded from `children`. We note that these tuples are stored in `queue`, thus the first element of `queue` is always the tuple with the highest `value`, i.e. `queue[0][2]` is the split action maximising $\mathcal{Q}(l, a)$. This is the reason we store `-value` in the tuple, Python implements min heaps instead of max heaps. We do not need to sort all actions by their values, but rather to just keep track of the action with the highest value. As a result, l becomes complete if and only if one of the following holds:

– The terminal action is the current best action:

$$\mathcal{Q}(l, \overline{a}) = \text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a)$$

This happens if:

$$-\texttt{queue[0][0]} \leq \texttt{value\_terminal}$$

– The tuple containing l and the current best split action is complete. This happens if the dictionary of incomplete children (that result from taking the current best split action in l) `queue[0][3]` is empty.

There is an additional benefit to storing `-value_complete`. When there are multiple split actions `attribute` that maximise `value`, then we prioritise the one maximising `value_complete`. This further serves as a heuristic allowed by the flexibility of BRANCHES' selection step. To see the benefit of this scheme, consider such situation and suppose that one of these split actions, `attribue`, maximising the estimate is such that $(l, \texttt{attribute})$ is complete. Ideally we want to prioritise this action, i.e. we want `queue[0][2] = attribute`. The motivation behind this is to immediately conclude that l is complete and thus deduce that $\texttt{attribute} = \text{Argmax}_{a \in \mathcal{A}(l) \setminus \{\overline{a}\}} \mathcal{Q}^*(l, a)$. This is exactly what happens by using a priority based on `value_complete` as well. Indeed, if $(l, \texttt{attribute})$ is complete, then by definition `attribute` maximises `value_complete` among all the split actions maximising `value`.

4.5.2 The Algorithm

In this section, we go over BRANCHES' search strategy, introduced in Section 4.3.2, and we outline it from an implementation perspective. We initialise the root Ω, then we apply the search steps at each iteration as follows:

Selection: Initialise the current branch $l = \Omega$ and the path list to `path = [1]`. While l is incomplete and l.`children` is not empty, i.e. l has been expanded. Consider the tuple:

$$\texttt{(-value, -value\_complete, attribute, children)} = l.\texttt{queue[0]}$$

As we have seen in Section 4.5.1, `attribute` is the optimal split action with respect to the current estimates $\mathcal{Q}(l, a)$ and `children` is the subset of incomplete children in $\text{Ch}(l, \texttt{attribute})$. Therefore, we choose the next branch l from the dictionary `children`. As explained in Section 4.3.2, here choosing the incomplete child of lowest `value_greedy` is our practical choice. Append l to `path`.

Expansion: Let l be the Selected branch. If l.`complete`, we go to the Backpropagation step. Otherwise, for each (unused) feature-category $(i, j) \in l.\texttt{attributes\_categories}$ let $l_{ij} = l \wedge \mathbb{1}\{X^{(i)} = j\}$ be the child branch of l that corresponds to feature i taking the value j. Our objective is to calculate $\mathcal{V}(l_{ij})$. We first check whether l_{ij}.`id_branch` is in the memo, if it is, then we can directly access a previously calculated $\mathcal{V}(l_{ij})$. Otherwise, we initialise $\mathcal{V}(l_{ij})$ according to Eq. (4.5). To do this efficiently, consider a fixed feature i and let us go over its categories $j \in \{1, \dots, C_i\}$ one by one. For l_{i1}, we first extract the data in l using l.`bit_vector`:

$$\mathcal{D}_l = \{X_m \in \mathcal{D} : l(X_m) = 1\} = \mathcal{D}[l.\texttt{bit\_vector}]$$

Since $l_{i1}(X) = 1 \implies l(X) = 1$, we can extract the data in l_{i1} directly from the smaller set $\mathcal{D}_l$ instead of $\mathcal{D}$:

$$\mathcal{D}_{l_{i1}} = \{X_m \in \mathcal{D} : l_{i1}(X_m) = 1\} = \{X_m \in \mathcal{D}_l : l_{i1}(X_m) = 1\}$$

The indices of the data in $\mathcal{D}_{l_{i1}}$ form the vector l_{i1}.`bit_vector`. Now we can initialise $\mathcal{V}(l_{i1})$ with Eq. (4.5) using $\mathcal{D}_{l_{i1}}$. For l_{i2}, if l_{i2}.`id_branch` is not in the memo, then to initialise $\mathcal{V}(l_{i2})$, instead of extracting $\mathcal{D}_{l_{i2}}$ from $\mathcal{D}_l$ via:

$$\mathcal{D}_{l_{i2}} = \{X_m \in \mathcal{D}_l : l_{i2}(X_m) = 1\}$$

We rather use the fact that l_{i1} and l_{i2} are mutually exclusive, in the sense that:

$$\forall X \in \mathcal{X} : l_{i2}(X) = 1 \implies l_{i1}(X) = 0$$

Which means that we can extract $\mathcal{D}_{l_{i2}}$ from the smaller set $\mathcal{D}_l \setminus \mathcal{D}_{l_{i1}}$ instead of $\mathcal{D}_l$ and then initialise $\mathcal{V}(l_{i2})$. We repeat this process for all categories $j \in \{1, \ldots, C_i\}$ and then we do the same thing for the remaining unused features in l.`attributes_categories`. These micro-optimisations we perform allow for a substantial computational gain in practice.

Backpropagation: For $j = \mathit{length}(\texttt{path}) - 1, \ldots, 1$ let `parent = path[j-1]` and `child = path[j]`, then we pop the heap queue `parent.queue`:

$$\texttt{(-value, -value\_complete, attribute, children)} = \texttt{parent.queue.pop()}$$

During the Selection step, `attribute` was the action taken at the branch `parent` to transition to the branch `child`. Now during Backpropagation, we need to update the estimates $\mathcal{Q}(\texttt{parent, attribute})$ and $\mathcal{V}(\texttt{parent})$, hence why we pop the corresponding tuple from `parent.queue`, and once we update its values, we will push the tuple back in the heap queue. If `child.complete` then we add its value to `value_complete`:

$$\texttt{value\_complete} \leftarrow \texttt{value\_complete + child.value}$$

and we pop `child` from the dictionary of incomplete children `children.pop(child)`. Now we push the tuple back in `parent.queue`, this rearranges the tuples is such a way that the most promising one (with maximum `value`) is at `parent.queue[0]`. The next step is to check:

$$\texttt{(-value, -value\_complete, attribute, children)} = \texttt{parent.queue[0]}$$

if `attribute` is the same attribute we have just treated with the last tuple, then we stop the update of `parent.queue`. Otherwise, we pop the queue again and we update this tuple in similar fashion to what we did with the previous tuple. The reason we check this is that an update could already be made here because the incomplete children in dictionary `children` might have been updated during other iterations of Branches. We continue this process until we get an `attribute` that has already been treated in this process, in which case there is no need for further updates. Now `parent.queue[0]` is the tuple corresponding to the best split action:

$$\texttt{(-value, -value\_complete, attribute, children)} = \texttt{parent.queue[0]}$$

Therefore, the value of `parent` is equal to the maximum between the value of taking this best split action and the value of taking the terminal action:

$$\mathcal{V}(\texttt{parent}) = \max\left\{\mathcal{Q}(\texttt{parent}, \overline{a}), \mathcal{Q}(\texttt{parent, attribute})\right\}$$

Which, in our implementation translates to:

$$\texttt{parent.value} \leftarrow \max\left\{\texttt{parent.value\_terminal}, \texttt{value}\right\}$$

Moreover, if $\mathcal{V}(\texttt{parent}) = \mathcal{Q}(\texttt{parent}, \overline{a})$, then $\overline{a} = \text{Argmax}_{a \in \mathcal{A}(\texttt{parent})} \mathcal{Q}(\texttt{parent}, a)$, and since we know that $\mathcal{Q}^*(\texttt{parent}, \overline{a}) = \mathcal{Q}(\texttt{parent}, \overline{a})$ (according to Eq. (4.1)), then we deduce that `parent` is complete and $\mathcal{V}^*(\texttt{parent}) = \mathcal{Q}^*(\texttt{parent}, \overline{a})$. Therefore we update:

$$\texttt{parent.complete} \leftarrow \texttt{True}$$

This is not the only condition that makes `parent` complete. Indeed, `parent` can also be complete if `(parent, attribute)` is complete, which happens when the dictionary `children` is empty.

4.6 Branches vs GOSDT

Among the optimal sparse DT algorithms, GOSDT is the closest to Branches. Indeed, MurTree and STreeD are DFS methods unlike OSDT, GOSDT and Branches. Moreover OSDT operates on the space of DTs instead of the space of branches unlike GOSDT and Branches. Nevertheless, Branches and GOSDT are fundamentally different algorithms even in their search strategies, we summarise below their main differences.

Support for ordinal encoding: The first straightforward difference is Branches' support for non-binary DTs that stem from ordinal encoding. In contrast, GOSDT (as well as all the algorithms we compare with) is only applicable to binary features, which necessitates a preliminary binary encoding. As we shall see in Section 4.7.2 and Section 4.7.3, this detail is crucial, it confers Branches' more scalability potential as these non-binary DT solutions are always found significantly faster (in the span of few seconds) compared to their binary counterparts. In Section 4.8 we shall explain the drawbacks of binary encoding from a theoretical standpoint.

The purification bound: It is true that (Lin et al., 2020, bounds (9) and (10)) and the purification bound stem from the same reasoning, which we call purification when explaining the intuition behind Proposition 7. Furthermore (Lin et al., 2020, bounds (9) and (10)) support equivalent points, a situation that arises when the dataset includes duplicate instances with different classes, while our bound does not currently support this. However, this is straightforward to incorporate in the purification bound and it will be in a future version of Branches. On the other hand, we emphasize that while (Lin et al., 2020, bounds (9) and (10)) and the purification bound stem from a similar reasoning, they are different. The purification bound pertains to penalising the number of splits while (Lin et al., 2020, bounds (9) and (10)) rather correspond to penalising the number of leaves. This would lead to differences when considering non-binary features. Moreover, (Lin et al., 2020, bounds (9) and (10)) are formulated for a binary classification setting with binary features. The purification bound is formulated for the general case of multiclass classification with categorical features that are not necessarily binary. We further prove in Proposition 7 that the purification bound is an upper bound on the true optimal values $\mathcal{Q}^*(l, a)$ and $\mathcal{V}^*(l)$ when it is initialised and also when it is updated during Backpropagation.

The Selection process: By Selection process, we refer to the strategy of choosing the branch to explore, by expanding it into children branches and then updating its value estimate (or bound). For Branches, this is performed within an MDP framework where we start from Ω and follow the action with highest value estimate $\text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a)$ until reaching a branch that is either complete or that has not been expanded yet (this can also be viewed as an AO* Selection process within an AND/OR graph). This on-policy process is thoroughly explained in Section 4.3.2 and Section 4.5.2, it is achieved through the use of multiple small priority

queues at the level of each branch to keep track of $\text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a)$. In contrast, GOSDT employs one global priority queue where all the encountered branches are stored. Using this queue, GOSDT selects the most promising branch, i.e. the branch with the best (lowest in GOSDT's setting) bound. However, this branch would not necessarily be the most promising one for BRANCHES because it does not necessarily lie in BRANCHES' selection path. Indeed, the branch with highest $\mathcal{V}(l)$ does not necessarily lie in the path following $\text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a)$ from Ω; in the terminology of (Nilsson, 2014, p.105), the selected branch by GOSDT does not necessarily belong to the current best partial solution graph. This difference in the selection strategies explains the large difference in the number of iterations of BRANCHES and GOSDT, which we showcase to be largely in favour of BRANCHES in our experiments. In addition to this fundamental difference, our Selection process allows us to incorporate heuristics for adequately choosing an incomplete child, as explained in Section 4.3.2. This flexibility has the potential of improving the anytime behaviour, and indeed our experiments in Section 4.7.2 and Section 4.7.3 show BRANCHES' anytime behaviour to always outperform GOSDT's.

The Backpropagation process: BRANCHES updates value estimates along the selection path only. It is true that multiple paths lead to the same selected branch l, and as such the branches along these paths could also be updated in principle. However, the number of these paths is equal to the number of permutations of the splits of l, which becomes computationally costly. Moreover, many of these branches will be pre-emptively pruned anyway and thus updating them would be a waste of time. Those that are not pruned will inevitably be visited by BRANCHES, and only then do we update them. In contrast, GOSDT updates the values along all the ancestors of the selected branch. In addition to this difference, we explained in Section 4.5 that BRANCHES further sorts split actions via their `value_complete`. As a consequence, if many split actions share the value $\text{Argmax}_{a \in \mathcal{A}(l) \setminus \{\overline{a}\}} \mathcal{Q}(l, a)$, then the action for which (l, a) is complete (when it exists) will always be prioritised, thus deducing immediately that l is complete and avoiding wasting time exploring it again in the future.

Caching procedure: BRANCHES uses branch caching while GOSDT uses dataset caching or support set caching. We could implement this caching procedure in the future for BRANCHES, however, we have some doubts about its soundness when it comes to solving for sparsity, or even for optimisation with a hard constraint on depth. Indeed, two branches l_1 and l_2 could share the same support set but differ in their number of splits $\mathcal{S}(l_1) < \mathcal{S}(l_2)$. In this case, the support set is insufficient to fully grasp the optimisation problem from each branch. From l_1 we have more splits to use before reaching the maximum depth than from l_2. Using dataset caching in this situation might lead to a suboptimal solution. This warrants further investigation in the future. In the absence of a proof of optimality specific to dataset caching (to our knowledge), we chose the conservative option of branch caching for BRANCHES.

Computational complexity analysis: In Section 4.4 we analysed the computational complexity of BRANCHES by bounding the number of branch evaluations it performs before terminating. Such analysis was not conducted for GOSDT. (Lin et al., 2020, Theorem H.1) provides a big-O bound on the total number of binary DTs that can be constructed from a set of M binary features, but it does not specifically analyse GOSDT's computational complexity. To our knowledge, only (Hu et al., 2019, Theorem E.2) provides such analysis for OSDT, hence why we compared with it in Section 4.4.

Empirical results: The experiments we conduct in Section 4.7 show that BRANCHES dominates GOSDT on the majority of experiments and for different metrics. From a smaller runtime

and significantly fewer iterations to a better anytime behaviour. These findings further support the efficiency of BRANCHES' search strategy compared to GOSDT's.

4.7 Experiments

We split our empirical study into three sections. We first compare optimal sparse DTs with optimal DTs and greedy DTs in terms of their train and test performance frontiers with respect to accuracy and number of splits. Then we compare BRANCHES with the state of the art optimal sparse DT methods in terms of their performances at large maximum depths. Lastly, we further compare the performances of these methods across a wide range of maximum depths. Below we provide references to the implementations we used:

- BRANCHES `https://github.com/Chaoukia/branches.git`.
- DL8.5 (Aglin et al., 2021).
- OSDT `https://github.com/xiyanghu/OSDT.git`.
- GOSDT `https://github.com/ubc-systopia/gosdt-guesses.git`.
- MurTree `https://github.com/MurTree/pymurtree.git`.
- STreeD (v1.3.1) `https://github.com/AlgTUDelft/pystreed.git`.

MurTree, STreeD and GOSDT use support set caching and the similarity bound. On the other hand, the current implementation of BRANCHES only supports branch caching and it does not include a similarity bound. Additionally, we set the maximum nodes for MurTree to 80. We noticed that without this constraint, the kernel dies immediately. The optimal sparse DT for all the experiments we consider have less than 80 nodes, thus this constraint should not cause MurTree to be suboptimal.

4.7.1 Optimal Sparse DTs vs Optimal DTs vs Greedy DTs

For optimal sparse DTs, we employ BRANCHES because it is our proposed approach. It is not necessary here to include other optimal sparse DT methods because, upon terminating, they yield the same solutions. Their performances will be compared in Section 4.7.2 and Section 4.7.3. For optimal DTs, subject to constraints on maximum depth, we use DL8.5 and for greedy DTs we chose CART. Fig. 4.3 and Fig. 4.4 report the median and the quartiles of the accuracy and number of splits of the proposed solutions induced by a 10 fold crossvalidation, branches-o in the legends represents BRANCHES applied to ordinal encoding. For DL8.5, we ran it with maximum depths raging between 1 and 10. As for BRANCHES and CART, they were run with $\lambda \in \{0.1, 0.05, 0.025, 0.01, 0.005, 0.001\}$. Note that λ in the context of CART refers to the cost-complexity parameter (ccp_alpha in scikit-learn) that is used for pruning. It plays a similar role to the penalty parameter λ, hence why we tested both BRANCHES and CART with the same set of λ values. A similar set of experiments was conducted by Hu et al. (2020) albeit with the maximum depth for CART instead of the cost-complexity parameter. We noticed that the cost-complexity parameter yields better frontiers for CART than the maximum depth, hence our choice. Fig. 4.5 and Fig. 4.6 report the same results as Fig. 4.3 and Fig. 4.4 respectively but with the inclusion of DL8.5. We made this distinction for clarity purposes because DL8.5's solutions become so complex (high number of splits) that the differences between the frontiers of CART and BRANCHES become less apparent.

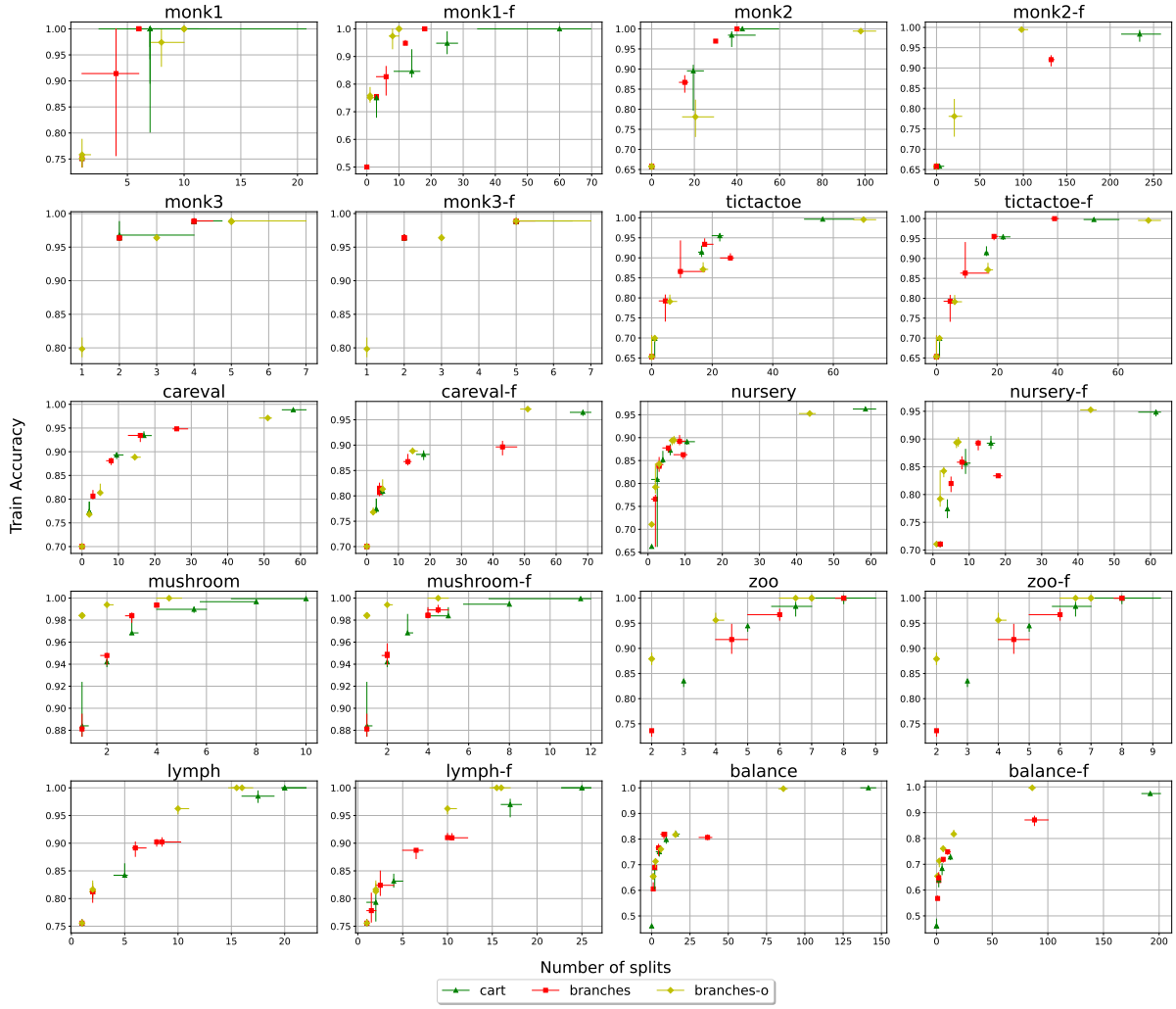


Figure 4.3: Pareto fronts of training accuracy against the number of splits of the proposed solutions. branches-o indicated that BRANCHES is applied to an ordinal encoding of the data.

Fig. 4.3 shows that BRANCHES displays better training frontiers than those of CART, and on many occasions branches-o yields even better training frontiers as we can see in monk1-f, monk2-f, careval-f, nursery, nursery-f, mushroom, mushroom-f, zoo, zoo-f, lymph-f and balance-f. This indicates that applying BRANCHES on an ordinal encoding of the data is not only good from a scalability standpoint (as we shall see with the extremely fast termination in Table 4.2) but can also yield better solutions in terms of accuracy and the number of splits. On the other hand, the differences in test frontiers, depicted in Fig. 4.4 are less clear than for the training frontiers. Nevertheless, we can still observe some advantage for BRANCHES' test frontiers over CART's. A similar observation was made in (Lin et al., 2020, Section 5) where the authors state:

Learning theory provides guarantees that training and test accuracy are close for sparse trees.

Fig. 4.4 illustrates another important point, which is the tendency of branches-o to overfit as we observe in monk2, monk2-f, tictactoe, tictactoe-f, lymph, lymph-f, balance and balance-f. This phenomenon can be explained by the tendency of non-binary DTs on ordinal encodings to induce branches containing smaller subsets of data than those induced by their binary DTs counterparts, thus increasing the risk of overfitting. Notice that this overfitting risk is mitigated

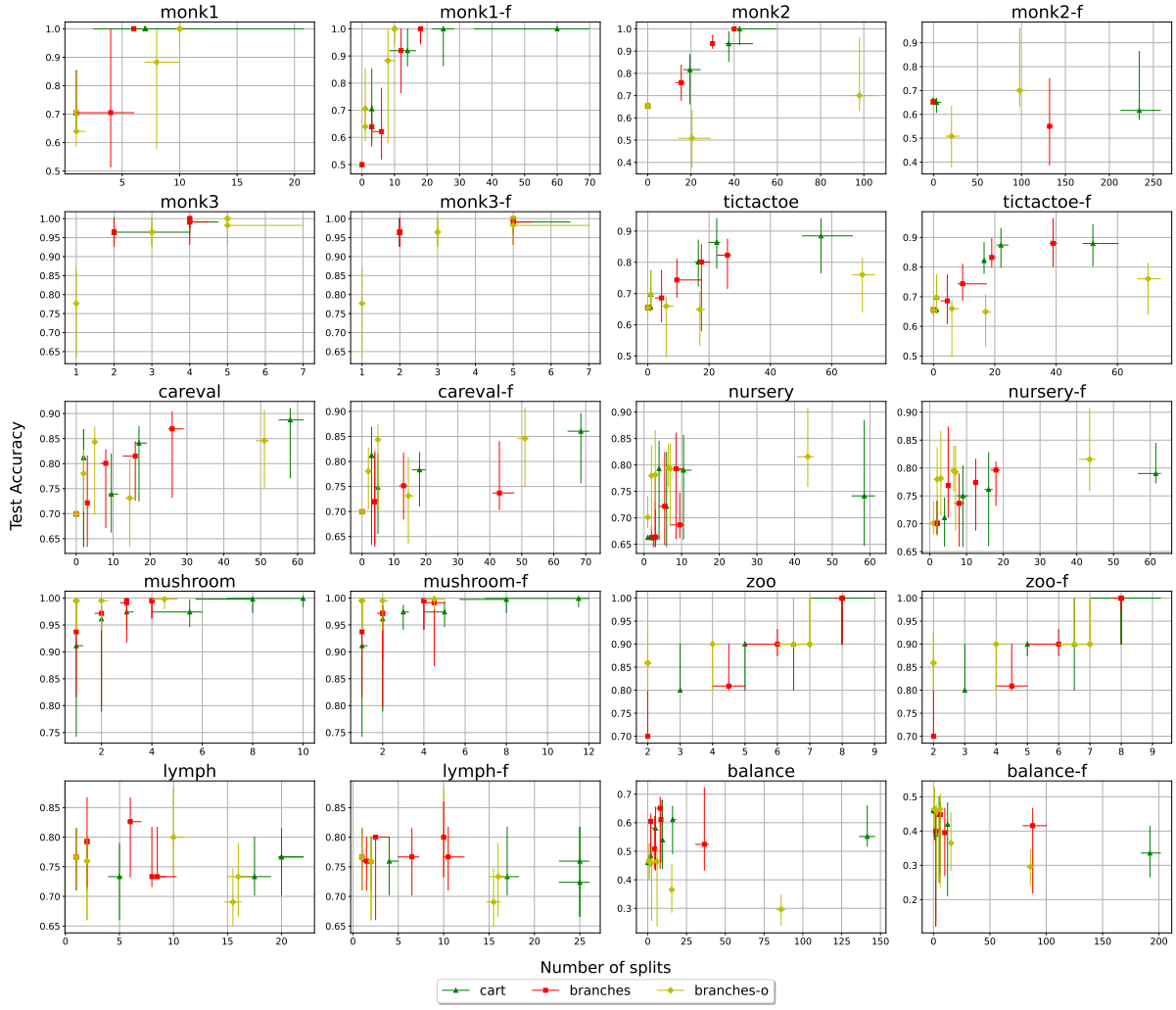


Figure 4.4: Pareto fronts of test accuracy against the number of splits of the proposed solutions. branches-o indicated that BRANCHES is applied to an ordinal encoding of the data.

for large datasets such as nursery (12960 data) and mushroom (8124 data).

From Fig. 4.5 and Fig. 4.6, we observe that DL8.5 actively seeks more and more complex DTs as the maximum depth parameter increases, thus disregarding sparsity concerns and inducing poor frontiers compared to BRANCHES and CART. To alleviate this tendency, Aglin et al. (2020) further include a constraint on the minimum support size of DL8.5, i.e. the minimum number of data that a branch has to contain in order to be considered for expansion. In our next experiments, we investigate this constraint and show that even with the best configurations DL8.5 is still suboptimal from a sparsity perspective.

In all the experiments, we set the maximum depth of DL8.5 to the depth of the true optimal sparse DT that we derive using BRANCHES. We analyse DL8.5's performance over 20 values of minimum support size taken between 1 and 50. Fig. 4.7 shows that, even with the knowledge of the best possible maximum depth parameter and the best configuration of minimum support size, DL8.5 almost never approaches the baseline derived by BRANCHES. Fig. 4.8 further illustrates that BRANCHES' solution is always outside the accuracy-splits frontier dis-

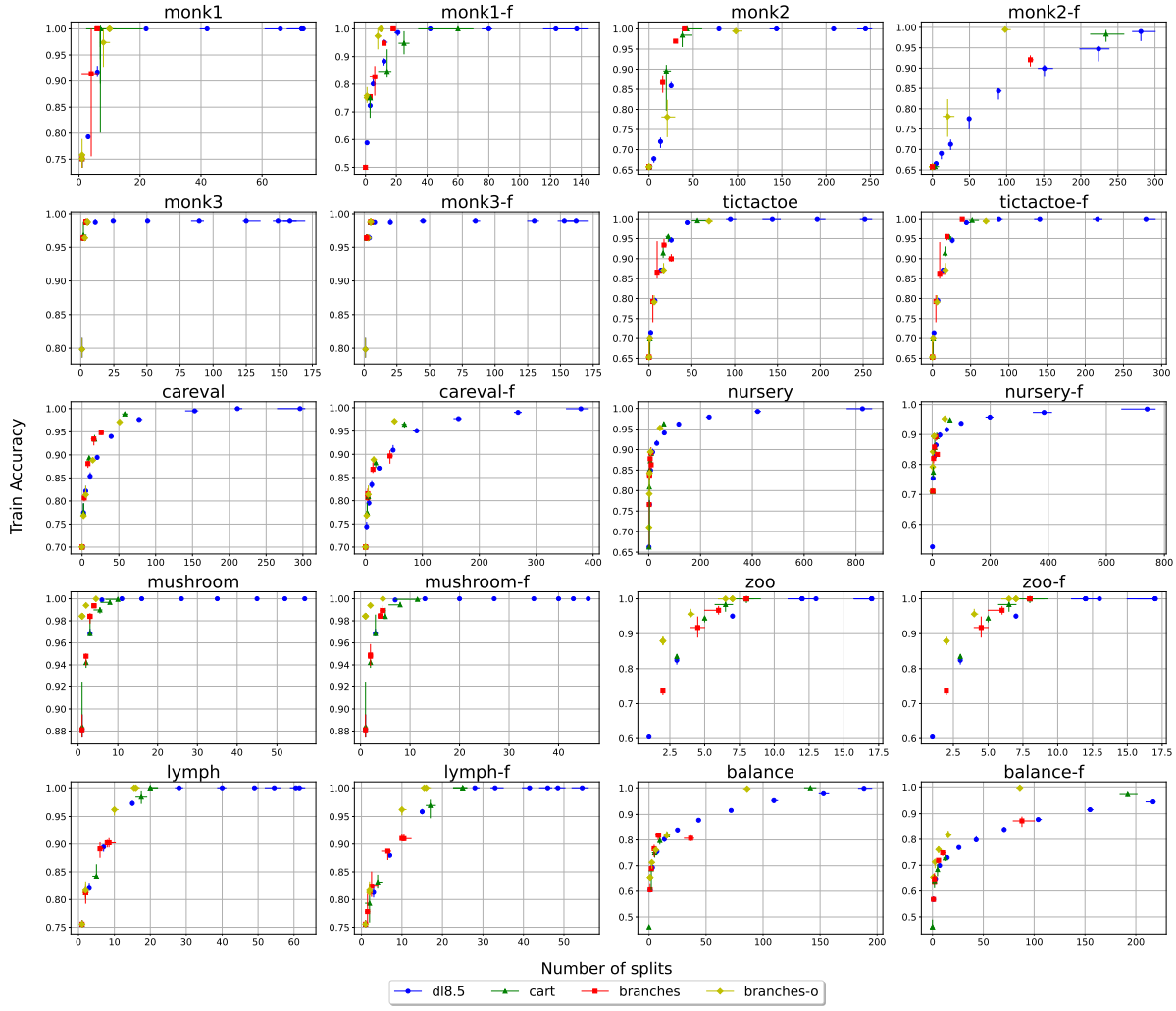


Figure 4.5: Pareto fronts of training accuracy against the number of splits of the proposed solutions. This figure is similar to Fig. 4.3 but further includes DL8.5.

played by DL8.5, indicating that BRANCHES' solution always dominates DL8.5's solution from the accuracy-sparsity perspective in these experiments.

4.7.2 Comparison on a large maximum depth

We employ 11 datasets from the UCI repository. For each dataset we use different types of encodings: suffix -o indicates ordinal encoding, suffix -f indicates binary encoding where the first category of each feature is dropped and no suffix designates a binary encoding retaining all categories. An additional binary encoding dropping the last category is considered in monk1-l, the reason being to showcase that different dropping options can lead to widely differing optimal sparse DTs. Moreover, our motivation behind dropping a category in the first place pertains to reducing the number of resulting binary features, this in turn can greatly simplify the task of finding an optimal sparse DT and help the algorithms be more scalable as we will see shortly in Table 4.2. It is worth noting that the algorithms we compare with exclusively consider binary features, thus necessitating a preliminary binary encoding. This seemingly benign detail can significantly harm performance by introducing a large amount of unnecessary splits as we shall explain in Section 4.8. BRANCHES can sidestep this issue since it supports non-binary DTs on

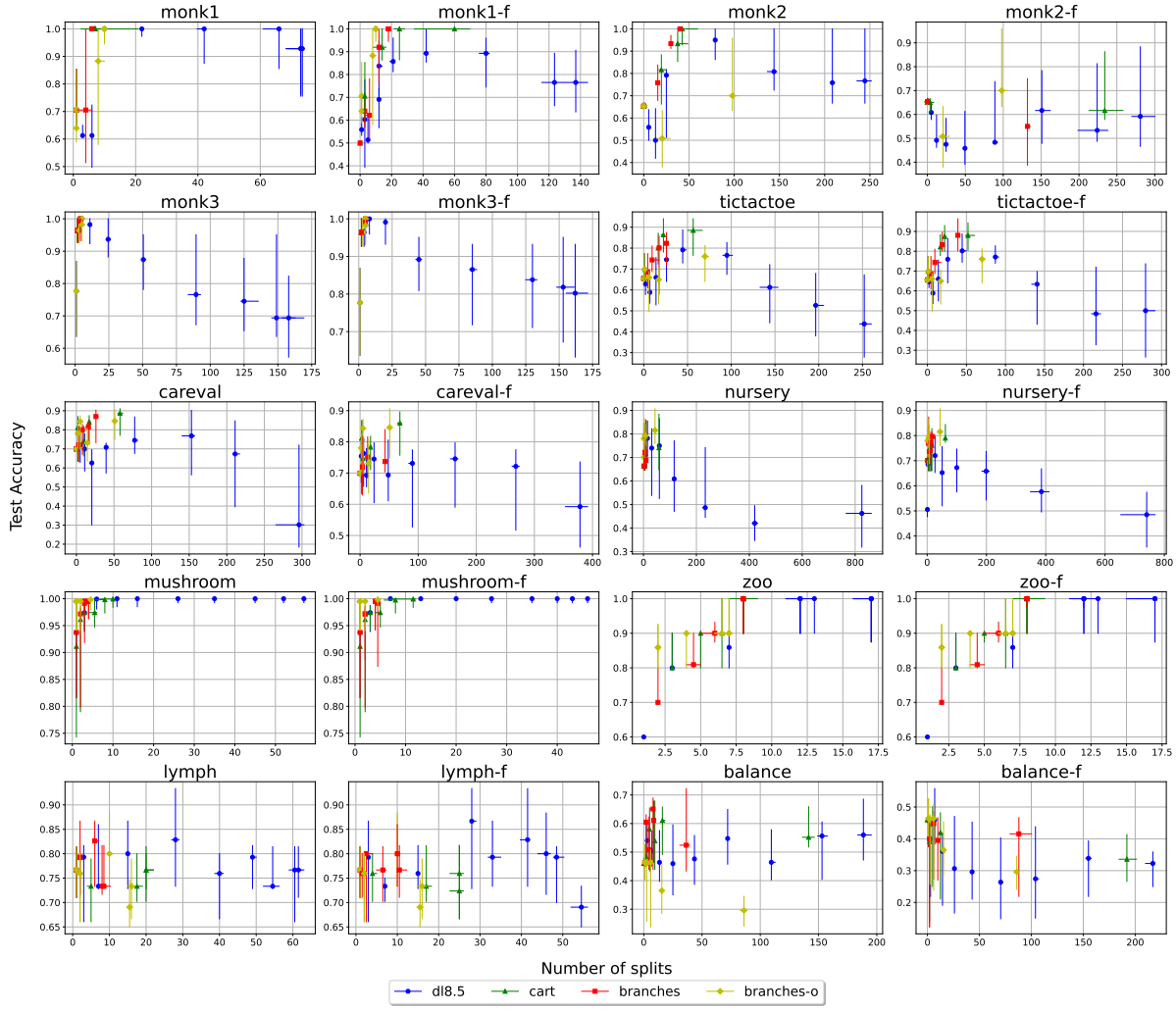


Figure 4.6: Pareto fronts of test accuracy against the number of splits of the proposed solutions. This figure is similar to Fig. 4.4 but further includes DL8.5.

ordinal encoding of the data. We set a time limit of 5 minutes for all algorithms. It is necessary to fix a maximum depth for DFS methods, in these experiments, the aim is to analyse performance at a large maximum depth, for this reason we fix it to 20 for for MurTree and STreeD while the BFS methods GOSDT and Branches run with infinite maximum depth. Table 3.1 summarises the characteristics of the datasets we consider.

We first discuss MurTree's results. The implementation we use from `https://github.com/MurTree/pymurtree.git` displays a suboptimal behaviour (in terms of the objective $\mathcal{H}_\lambda$) even when the algorithm terminates unlike STreeD, GOSDT and Branches. This phenomenon happens for monk2, monk2-f, monk3, monk3-f, car-eval-f, nursery-f, zoo and zoo-f. Furthermore, according to `https://bitbucket.org/EmirD/murtree/src/master/`, the authors of MurTree indicate the release of a *a newer and more general version of the algorithm* referring to STreeD. For this reason, we will restrict the comparison to STreeD, GOSDT and Branches in the rest of the discussion.

STreeD is always optimal when it terminates similarly to GOSDT and Branches. On mush-

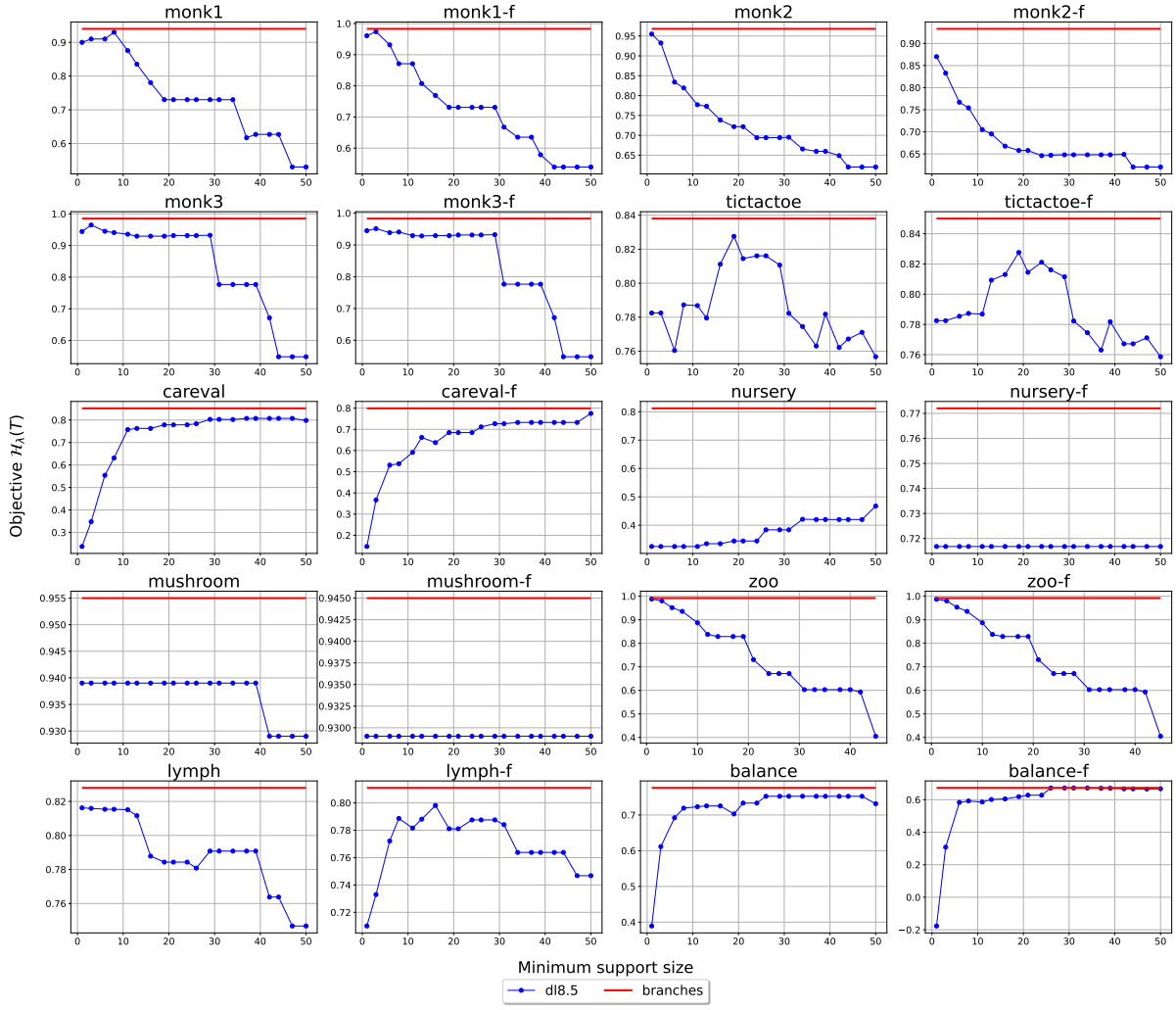


Figure 4.7: Comparing $\mathcal{H}_\lambda(T)$ of the proposed solutions by DL8.5 for different values of the minimum support size with the BRANCHES baseline. The maximum depth of DL8.5 is set to the depth of the true optimal solution.

room and mushroom-f it is clearly the superior method as it finds the optimal sparse DT in 10.8s and 126s respectively while both GOSDT and BRANCHES reach timeout on these datasets. On the other hand, despite reaching timeout, the anytime behaviour of BRANCHES still proposes the true optimal sparse DT in this situation unlike GOSDT, making it the superior anytime behaviour on these datasets. On the remaining datasets however, STreeD's DFS strategy suffers from the large maximum depth while the BFS methods GOSDT and BRANCHES achieve optimality much faster even while running at an infinite maximum depth, this is especially the case for BRANCHES because STreeD outperforms GOSDT on monk3, monk3-f, mushroom, mushroom-f and zoo-f, while it outperforms BRANCHES only on mushroom and mushroom-f. Furthermore, STreeD does not enjoy an anytime behaviour which causes it to propose no solution when it reaches timeout. This happens for monk2, monk2-f, tic-tac-toe, car-eval, car-eval-f, nursery, nursery-f, kr-vs-kp, lymph, lymph-f, balance and balance-f.

Remark on STreeD's anytime behaviour: It is possible to endow STreeD with an anytime behaviour through an iterative deepening where a loop is performed over multiple values

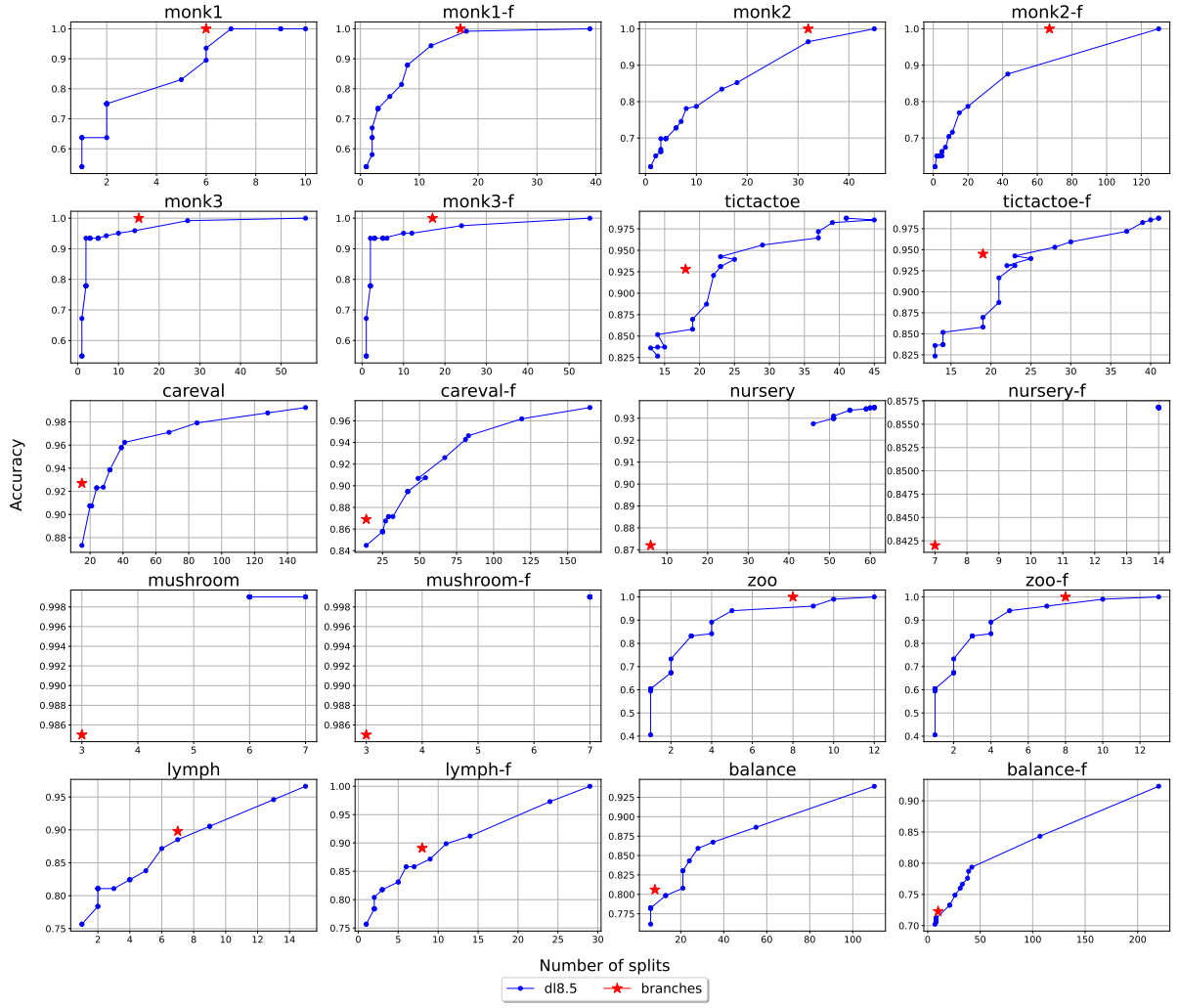


Figure 4.8: Comparing the Accuracy and number of splits of the induced solution by DL8.5 for different values of the minimum support size with the Branches baseline.

of maximum depth starting from 1. Upon reaching timeout, STreeD can propose the best previously found solution during this loop. However, such scheme would strip STreeD from its storage-economy advantage, as it would have to store the graph structure to maintain optimality, all while still employing an uninformative search strategy. As a consequence, this could lead to substantially slower runtimes and even more memory consumption that BFS methods. We note that we have not compared with this scheme for STreeD because at the time of the writing of this thesis, STreeD's Python API (v1.3.1) did not include it.

When comparing the BFS methods GOSDT and Branches, we notice that Branches outperforms GOSDT on all experiments except for car-eval-f and balance-f where GOSDT terminates faster. Occasionally, Branches can even terminate while GOSDT runs out of time and thus fails to produce the optimal solution. This happens in car-eval and nursery-f. Moreover, whenever both algorithms reach timeout, Branches proposes a better solution than GOSDT's as observed for tic-tac-toe, nursery, mushroom, kr-vs-kp, lymph, lymph-f and balance, thus showcasing a better anytime behaviour for Branches.

Table 4.2: Comparing optimal sparse DT methods for a large maximum depth 20. BRANCHES is the only method that is applicable to ordinal encoding, hence why we put — for the remaining methods. Furthermore, we ran into memory issues with MurTree that kill the kernel, in these cases also we put —. For STreeD, when it reaches timeout, it does not propose a solution due to its lack of anytime behaviour, we indicate those cases with — as well.

Dataset	MurTree				STreeD				GOSDT				Branches			
	objective	accuracy	splits	time (s)	objective	accuracy	splits	time (s)	objective	accuracy	splits	time (s)	objective	accuracy	splits	time (s)
monk1	**0.940**	**1**	**6**	**0.04**	**0.940**	**1**	**6**	3.28	**0.940**	**1**	**6**	3.05	**0.940**	**1**	**6**	**0.05**
monk1-l	**0.930**	**1**	**7**	**0.03**	**0.930**	**1**	**7**	2.80	**0.930**	**1**	**7**	0.87	**0.930**	**1**	**7**	**0.02**
monk1-f	**0.983**	**1**	**17**	**0.05**	**0.983**	**1**	**17**	6.18	**0.983**	**1**	**17**	4.09	**0.983**	**1**	**17**	0.39
monk1-o	—	—	—	—	—	—	—	—	—	—	—	—	**0.900**	**1**	**10**	**0.02**
monk2	0.967	**1**	33	0.55	—	—	—	*TO*	**0.968**	**1**	**32**	91.9	**0.968**	**1**	**32**	**14.2**
monk2-f	0.922	**1**	78	1.07	—	—	—	*TO*	**0.933**	**1**	**67**	9.78	**0.933**	**1**	**67**	**2.94**
monk2-o	—	—	—	—	—	—	—	—	—	—	—	—	**0.955**	**1**	**45**	**0.18**
monk3	0.978	**1**	25	0.04	**0.985**	**1**	**15**	7.87	**0.985**	**1**	**15**	25.2	**0.985**	**1**	**15**	**4.05**
monk3-f	0.975	**1**	25	0.04	**0.983**	**1**	**17**	4.82	**0.983**	**1**	**17**	2.09	**0.983**	**1**	**17**	**0.36**
monk3-o	—	—	—	—	—	—	—	—	—	—	—	—	**0.987**	**1**	**13**	**0.03**
tic-tac-toe	—	—	—	—	—	—	—	*TO*	0.757	0.792	7	*TO*	**0.838**	**0.928**	**18**	*TO*
tic-tac-toe-f	**0.850**	**0.945**	**19**	**16.4**	**0.850**	**0.945**	**19**	207	**0.850**	**0.945**	**19**	57.8	**0.850**	**0.945**	**19**	**16.3**
tic-tac-toe-o	—	—	—	—	—	—	—	—	—	—	—	—	**0.773**	**0.858**	**17**	**0.68**
car-eval	**0.852**	**0.927**	**15**	**154**	—	—	—	*TO*	**0.852**	**0.927**	**15**	*TO*	**0.852**	**0.927**	**15**	204
car-eval-f	**0.799**	**0.869**	**14**	58	—	—	—	*TO*	**0.799**	**0.869**	**14**	**21.1**	**0.799**	**0.869**	**14**	26.6
car-eval-o	—	—	—	—	—	—	—	—	—	—	—	—	**0.812**	**0.882**	**14**	**0.09**
nursery	—	—	—	—	—	—	—	*TO*	0.810	0.860	5	*TO*	**0.812**	**0.872**	**6**	*TO*
nursery-f	**0.772**	**0.842**	**7**	151	—	—	—	*TO*	0.765	0.835	7	*TO*	**0.772**	**0.842**	**7**	**24.9**
nursery-o	—	—	—	—	—	—	—	—	—	—	—	—	**0.822**	**0.892**	**7**	**0.24**
mushroom	**0.955**	**0.985**	**3**	11.1	**0.955**	**0.985**	**3**	**10.8**	0.925	0.945	2	*TO*	**0.955**	**0.985**	**3**	*TO*
mushroom-f	**0.945**	**0.985**	**4**	143	**0.945**	**0.985**	**4**	**126**	0.925	0.945	2	*TO*	**0.945**	**0.985**	**4**	*TO*
mushroom-o	—	—	—	—	—	—	—	—	—	—	—	—	**0.975**	**0.985**	**1**	**0.15**
kr-vs-kp	—	—	—	—	—	—	—	*TO*	0.815	0.845	3	*TO*	**0.900**	**0.940**	**4**	*TO*
zoo	0.989	**1**	11	0.04	**0.992**	**1**	**8**	184	**0.992**	**1**	**8**	87.3	**0.992**	**1**	**8**	**44.6**
zoo-f	0.989	**1**	11	0.3	**0.992**	**1**	**8**	23.4	**0.992**	**1**	**8**	33.5	**0.992**	**1**	**8**	**2.09**
zoo-o	—	—	—	—	—	—	—	—	—	—	—	—	**0.993**	**1**	**7**	**0.87**
lymph	—	—	—	—	—	—	—	*TO*	0.790	0.810	2	*TO*	**0.828**	**0.898**	**7**	*TO*
lymph-f	—	—	—	—	—	—	—	*TO*	0.784	0.804	2	*TO*	**0.811**	**0.891**	**8**	*TO*
lymph-o	—	—	—	—	—	—	—	—	—	—	—	—	**0.852**	**0.952**	**10**	**12.3**
balance	—	—	—	—	—	—	—	*TO*	0.712	0.737	5	*TO*	**0.776**	**0.806**	**8**	*TO*
balance-f	—	—	—	—	—	—	—	*TO*	**0.673**	**0.723**	**10**	**162**	**0.673**	**0.723**	**10**	190
balance-o	—	—	—	—	—	—	—	—	—	—	—	—	**0.713**	**0.763**	**10**	**0.004**

Another point worth mentioning here is that the experiments on ordinal encoded datasets achieve optimality significantly faster than their binary encoded counterparts. Indeed, no experiment with ordinal encoding reaches timeout, the slowest one being lymph-o where it took BRANCHES 12.3s to terminate. In Section 4.8 we explain the cause of this phenomenon. Furthermore, on many occasions, the non-binary optimal sparse DTs obtained with ordinal encoding are of better quality (in terms of $\mathcal{H}_\lambda$) than those obtained through a preliminary binary encoding. This happens for monk3-o, nursery-o, mushroom-o, zoo-o and lymph-o, but this is not the case for the remaining datasets. In fact, we shall see in Section 4.8 that the obtained non-binary sparse DTs can themselves be *collapsed* into sparser solutions (fewer splits) that are more interpretable. Knowing that scalability is a major issue in the literature of optimal sparse DTs, the ability to support non-binary DTs is very helpful in this regard because these solutions can be found more quickly. BRANCHES being the only method with this property has a clear advantage from a scalability standpoint.

Lastly Table 4.2 confirms our hypothesis that dropping a category per feature during binary

Table 4.3: Comparing GOSDT and Branches. This table also includes the number of iterations.

Dataset	GOSDT					Branches				
	objective	accuracy	splits	time (s)	iterations	objective	accuracy	splits	time (s)	iterations
monk1	**0.940**	**1**	**6**	3.05	83928	**0.940**	**1**	**6**	**0.05**	**146**
monk1-l	**0.930**	**1**	**7**	0.87	29770	**0.930**	**1**	**7**	**0.02**	**117**
monk1-f	**0.983**	**1**	**17**	4.09	92782	**0.983**	**1**	**17**	**0.39**	**2125**
monk1-o	—	—	—	—	—	**0.900**	**1**	**10**	**0.02**	**64**
monk2	**0.968**	**1**	**32**	91.9	392759	**0.968**	**1**	**32**	**14.2**	**60611**
monk2-f	**0.933**	**1**	**67**	9.78	149912	**0.933**	**1**	**67**	**2.94**	**28968**
monk2-o	—	—	—	—	—	**0.955**	**1**	**45**	**0.18**	**1213**
monk3	**0.985**	**1**	**15**	25.2	185974	**0.985**	**1**	**15**	**4.05**	**14807**
monk3-f	**0.983**	**1**	**17**	2.09	59151	**0.983**	**1**	**17**	**0.36**	**3026**
monk3-o	—	—	—	—	—	**0.987**	**1**	**13**	**0.03**	**156**
tic-tac-toe	0.757	0.792	7	*TO*	2279999	**0.838**	**0.928**	**18**	*TO*	**390000**
tic-tac-toe-f	**0.850**	**0.945**	**19**	57.8	1670379	**0.850**	**0.945**	**19**	**16.3**	**74627**
tic-tac-toe-o	—	—	—	—	—	**0.773**	**0.858**	**17**	**0.68**	**3339**
car-eval	**0.852**	**0.927**	**15**	*TO*	5893659	**0.852**	**0.927**	**15**	**204**	**456452**
car-eval-f	**0.799**	**0.869**	**14**	**21.1**	927221	**0.799**	**0.869**	**14**	26.6	**108640**
car-eval-o	—	—	—	—	—	**0.812**	**0.882**	**14**	**0.09**	**579**
nursery	0.810	0.860	5	*TO*	299999	**0.812**	**0.872**	**6**	*TO*	**110000**
nursery-f	0.765	0.835	7	*TO*	629999	**0.772**	**0.842**	**7**	**24.9**	**48063**
nursery-o	—	—	—	—	—	**0.822**	**0.892**	**7**	**0.24**	**195**
mushroom	0.925	0.945	2	*TO*	79999	**0.955**	**0.985**	**3**	*TO*	**21000**
mushroom-f	0.925	0.945	2	*TO*	99999	**0.945**	**0.985**	**4**	*TO*	**24000**
mushroom-o	—	—	—	—	—	**0.975**	**0.985**	**1**	**0.15**	**6**
kr-vs-kp	0.815	0.845	3	*TO*	159999	**0.900**	**0.940**	**4**	*TO*	**46000**
zoo	**0.992**	**1**	**8**	87.3	401799	**0.992**	**1**	**8**	**44.6**	**39199**
zoo-f	**0.992**	**1**	**8**	33.5	300387	**0.992**	**1**	**8**	**2.09**	**4659**
zoo-o	—	—	—	—	—	**0.993**	**1**	**7**	**0.87**	**1456**
lymph	0.790	0.810	2	*TO*	659999	**0.828**	**0.898**	**7**	*TO*	**100000**
lymph-f	0.784	0.804	2	*TO*	1079999	**0.811**	**0.891**	**8**	*TO*	**170000**
lymph-o	—	—	—	—	—	**0.852**	**0.952**	**10**	**12.3**	**16154**
balance	0.712	0.737	5	*TO*	1119999	**0.776**	**0.806**	**8**	*TO*	**640000**
balance-f	**0.673**	**0.723**	**10**	**162**	2292545	**0.673**	**0.723**	**10**	190	**676149**
balance-o	—	—	—	—	—	**0.713**	**0.763**	**10**	**0.004**	**178**

encoding can significantly decrease runtime, and thus yield better scalability. On tic-tac-toe for example, all algorithms run out of time while they terminate on tic-tac-toe-f in a few seconds: 16.3s for Branches, 16.4s for MurTree, 57.8s for GOSDT and 207s for STreeD. A similar observation applies to nursery vs nursery-f and balance vs balance-f. The only experiment where we noticed an increase in runtime due to dropping a feature is monk1/monk1-f. On the other hand, while it is true that this encoding strategy can yield more complex solutions (higher number of splits), this gap in complexity is usually very small. The widest gaps we reported pertain to monk1/monk1-f and monk2/monk2-f where the number of splits increased significantly from 6 to 17 and from 32 to 67 respectively. However, on the remaining experiments the maximum gap was only of 2 splits; car-eval/car-eval-f even showcased (surprisingly) a decrease in the number of splits of the solution from 15 to 14.

In addition to these comparisons and in light of the computational complexity analysis we performed in Section 4.4, we wanted to compare the number evaluations performed by the algorithms before terminating. Unfortunately this option is not provided by the APIs of the algorithms we use. Nonetheless, GOSDT returns the number of iterations to termination, this metric will serve as a proxy for the total number of evaluations and we compare GOSDT and Branches on this basis. Table 4.3 demonstrates that Branches always terminates in significantly fewer iterations than GOSDT, thus indicating a clear computational efficiency advantage

for Branches over GOSDT.

Below we summarise the findings of this section:

- BFS methods have a clear advantage over DFS methods for a large maximum depth, except on the mushroom datasets. This is important because BFS methods can run with an infinite maximum depth, thus alleviating the need to tune the maximum depth. This in turn can allow practitioners to divert all tuning efforts to other parameters such as λ and the minimum size of support sets.
- On all experiments, except car-eval-f and balance-f, Branches is faster than GOSDT.
- Branches displays a better anytime behaviour than GOSDT. On all experiments where they both reached timeout, Branches proposed better solutions.
- Achieving optimality with non-binary DTs on ordinal encoded data is always significantly faster than for binary DTs with binary features. The quality of those solutions are however not always better. Nevertheless, this substantial gain in runtime favours Branches over the other methods from a scalability perspective.
- Branches always achieves optimality within significantly fewer iterations compared to GOSDT, thus indicating a better search efficiency.

4.7.3 Comparison across a wide range of maximum depths

In this second set of experiments, we compare the performance of Branches, GOSDT and STreeD on a set of maximum depth values ranging from 4 to 20. For all the algorithms, we compare the objective $\mathcal{H}_\lambda$, accuracy and number of splits of the proposed solution in addition to the execution time. Moreover, we compare GOSDT and Branches in terms of the number of iterations. We also compare the depth of the proposed solutions between Branches and STreeD only because, to our knowledge, the implementation of GOSDT does not provide this metric. For presentation purposes, all the results are provided in Section 4.13. The legends of the figures contain:

- **branches:** Branches applied to binary encoding.
- **branches-o:** Branches applied to ordinal encoding.
- **gosdt:** GOSDT applied to binary encoding.
- **streed:** STreeD applied to binary encoding.
- **streed1:** STreeD with the depth 2 solver, introduced by Demirović et al. (2022), disabled. The reason we introduce this baseline is to assess the contribution of the depth 2 solver to STreeD's performance. We shall see that this option improves STreeD's performance significantly.

We start by discussing the performance of GOSDT. Interestingly, the objective $\mathcal{H}_\lambda$ reported for GOSDT does not match the one reported for Branches and STreeD. This might indicate that the implementation of max depth for GOSDT forces the solution to have a depth strictly lower than max depth, unlike Branches and STreeD where the solution is allowed to reach max depth. Nevertheless, this does not undermine the following discussion. Surprisingly, GOSDT performs often worse when limiting its maximum depth than otherwise. There are even cases where it runs out of memory even though this phenomenon has not been observed in any experiment from

Section 4.7.2. This happens for mushroom, mushroom-f, lymph, lymph-f, tic-tac-toe, krvskp, nursery and nursery-f, hence why GOSDT's results are missing in those figures. We note that this phenomenon has been observed by McTavish et al. (2022), the authors that incorporate the maximum depth parameter into GOSDT. They state:

Interestingly, using a large depth constraint is often less efficient than removing the depth constraint entirely for GOSDT, because when we use a branch-and-bound approach with recursion, the ability to re-use solutions of recurring sub-problems diminishes in the presence of a depth constraint.

On the other hand, despite being a BFS method as well and as such being more memory consuming than DFS methods, Branches never ran into this issue in both sets of experiments. This is likely explained by Branches' ability to terminate significantly earlier than GOSDT in terms of iterations, and thus terminating before running into memory issues. This large discrepancy in the number of iterations was observed in Section 4.7.2 and is further observed in all the experiments in this section. Furthermore, except on few cases such as balance, Branches dominates GOSDT in terms of runtime.

On these experiments the comparison with STreeD is more insightful. A common pattern is that STreeD dominates Branches in terms of speed for small depths up until a certain point where Branches becomes the dominating method. This is best observed in tic-tac-toe-f, car-eval-f, nursery-f, zoo and balance-f. We note also that in some cases such as balance, lymph-f, lymph and tic-tac-toe, STreeD proposes better solutions altogether than Branches because the latter reaches timeout, albeit for higher depths we cannot even compare the solutions of Branches and STreeD because STreeD reaches timeout and lacks an anytime behaviour. A very insightful experiment here is mushroom. We have seen in Section 4.7.2 that STreeD performs exceptionally well on mushroom and mushroom-f even for a large maximum depth. The experiments on this section further support this observation. Thus we naturally wonder: **Why does STreeD perform exceptionally well on the mushroom datasets? Is it because of the DFS strategy or something else?** To investigate this, we looked into a powerful tool in STreeD's arsenal, the depth 2 solver that was introduced by Demirović et al. (2022). This solver allows for finer estimates to be computed early on, it has been proven to yield significant runtime improvements, and moreover neither GOSDT nor Branches utilise it for now. Fortunately, STreeD's implementation allows the disabling of the depth 2 solver. Indeed, the depth 2 solver turns out to be paramount to STreeD's performance, without it, STreeD always runs out of time on the mushroom datasets. Moreover, now Branches largely dominates STreeD across the full range of max depth values, except on very few cases such as depths $13, 14$ and 15 in balance-f. With this, we conclude that STreeD's better performance on mushroom and smaller depths is not due to its DFS strategy, but rather to the depth 2 solver. This observation motivates us to consider adapting the depth 2 solver to the purification bound and incorporating it in a future version of Branches. This is a promising idea that could significantly improve Branches' performance.

Next we discuss the anytime behaviour. We have seen in Section 4.7.2 that STreeD lacks the anytime behaviour unlike GOSDT and Branches, which hinders its applicability. To cite a few examples, STreeD (with the depth 2 solver) starts reaching timeout from depth 7 for lymph and depth 8 for lymph-f and tic-tac-toe. On the other hand, notice that the solutions proposed by Branches even after reaching timeout always dominate those proposed by GOSDT in terms of the objective $\mathcal{H}_\lambda$. This is especially the case for car-eval, balance and balance-f and we recall that we could not report GOSDT's performance on many other datasets because of memory issues, this further supports our earlier anytime comparison between GOSDT and Branches in

Table 4.4: Comparing BRANCHES with the state of the art Python implementations for a large maximum depth 20.

Dataset	OSDT					PYGOSDT					Branches				
	objective	accuracy	splits	time (s)	iterations	objective	accuracy	splits	time (s)	iterations	objective	accuracy	splits	time (s)	iterations
monk1	**0.940**	**1**	**6**	2.38	94901	**0.940**	**1**	**6**	6.03	174523	**0.940**	**1**	**6**	**0.05**	**146**
monk1-l	**0.930**	**1**	**7**	71	2028577	**0.930**	**1**	**7**	181	3731292	**0.930**	**1**	**7**	**0.02**	**117**
monk1-f	0.971	**1**	29	*TO*	22308	0.970	**1**	30	*TO*	2018	**0.983**	**1**	**17**	**0.39**	**2125**
monk1-o	—	—	—	—	—	—	—	—	—	—	**0.900**	**1**	**10**	**0.02**	**64**
monk2	0.948	**1**	52	*TO*	41	0.948	**1**	52	*TO*	44	**0.968**	**1**	**32**	**14.2**	**60611**
monk2-f	0.904	0.982	76	*TO*	44083	0.903	0.982	77	*TO*	32475	**0.933**	**1**	**67**	**2.94**	**28968**
monk2-o	—	—	—	—	—	—	—	—	—	—	**0.955**	**1**	**45**	**0.18**	**1213**
monk3	0.976	0.991	15	*TO*	17728	0.976	0.991	15	*TO*	5765	**0.985**	**1**	**15**	**4.05**	**14807**
monk3-f	0.975	0.991	15	*TO*	11875	0.973	0.991	17	*TO*	897	**0.983**	**1**	**17**	**0.36**	**3026**
monk3-o	—	—	—	—	—	—	—	—	—	—	**0.987**	**1**	**13**	**0.03**	**156**
tic-tac-toe	0.794	0.869	15	*TO*	76	0.794	0.869	15	*TO*	69	**0.838**	**0.928**	**18**	*TO*	**390000**
tic-tac-toe-f	0.764	0.824	11	*TO*	40	0.808	0.824	11	*TO*	37	**0.850**	**0.945**	**19**	**16.3**	**74627**
tic-tac-toe-o	—	—	—	—	—	—	—	—	—	—	**0.773**	**0.858**	**17**	**0.68**	**3339**
mushroom	**0.955**	**0.985**	**3**	**76.2**	**1186819**	**0.955**	**0.985**	**3**	211	2681260	**0.955**	**0.985**	**3**	*TO*	21000
mushroom-f	**0.945**	**0.985**	**4**	*TO*	4704419	**0.945**	**0.985**	**4**	*TO*	2487909	**0.945**	**0.985**	4	*TO*	24000
mushroom-o	—	—	—	—	—	—	—	—	—	—	**0.975**	**0.985**	**1**	**0.15**	**6**
kr-vs-kp	**0.900**	**0.940**	**4**	*TO*	67161	**0.900**	**0.940**	**4**	*TO*	25379	**0.900**	**0.940**	4	*TO*	46000

Section 4.7.2.

Lastly, we discuss the application of BRANCHES to ordinal encoding. On all experiments, the optimal sparse DT is found significantly faster in this setting, even for the largest dataset nursery with 12960 examples. This further supports the scalability potential of this property. Moreover, while it is true that from the objective's perspective, the induced solutions with ordinal encodings are not always the best compared to those induced by binary encoding, however, they happen to be better many times such as in monk2-f, monk3, monk3-f, car-eval-f, nursery, nursery-f zoo, zoo-f, lymph, lymph-f, balance-f, mushroom and mushroom-f.

4.7.4 Comparing Branches with Python implementations

In this section, we compare BRANCHES with Python implementations and show a vast difference in performance in favour of BRANCHES.

Table 4.4 compares BRANCHES with OSDT and PYGOSDT, it contains less datasets than Table 4.2 because the implementations of OSDT and PYGOSDT are restricted to binary classification problems. Table 4.4 shows that BRANCHES outperforms OSDT and GOSDT in terms of runtime, number of iterations and quality of the proposed solution on all the experiments except mushroom, mushroom-f and kr-vs-kp. On mushroom, OSDT and PYGOSDT terminate in $76.2s$ and $211s$ respectively while BRANCHES reaches timeout. Nevertheless, BRANCHES' anytime behaviour still allowed the retrieval of the true optimal sparse DT in this case albeit with a disadvantage in terms of runtime. On mushroom-f and kr-vs-kp, all the methods reached timeout but derived similar solutions.

Table 4.4 showcases the shortcomings of Python implementations with regard to solving for sparsity. Both OSDT and PYGOSDT reach timeout and are suboptimal on the majority of datasets. Due to the optimisation difficulty of the problem, a lot of care has to be dedicated to low-level optimisation of the proposed implementations, a property that C++ offers better than

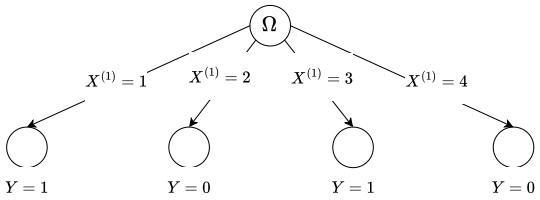


Figure 4.9: Optimal sparse DT depicting the class variable that satisfies $Y = 1$ if and only if $X^{(1)} = 1$ or $X^{(1)} = 3$ on the space $\mathcal{X} = \{1, 2, 3, 4\}$.

Python, hence the vast adoption of C++ in the state of the art algorithms solving for sparsity. BRANCHES being implemented in Python and yet outperforming its C++ competitors is a testament to its efficient search strategy. Moreover, it shows that a future C++ implementation of BRANCHES is very promising, especially when coupled with multi-threading as we shall explain in Section 4.9.

4.8 Drawbacks of Binary (One-Hot) Encoding

In Section 4.7.2, we have shown that optimal sparse DTs are always found significantly faster in the context of ordinal encoding compared to binary (one-hot) encoding. In this section, we investigate the reason behind this phenomenon.

To answer this question, let us consider the following simple binary classification problem. Suppose there is only one feature $X^{(1)}$ with 4 categories, i.e. $\mathcal{X} = \{1, 2, 3, 4\}$ and that class Y satisfies $Y = 1$ if and only if $X^{(1)} = 1$ or $X^{(1)} = 3$. The optimal sparse DT in this case consists of only one split splitting the root Ω with respect to feature $X^{(1)}$, as shown in Fig. 4.9. In this setting, BRANCHES only needs one iteration to terminate. Indeed, on its first iteration, it expands Ω, estimates $\mathcal{Q}(\Omega, \overline{a})$ and $\mathcal{Q}(\Omega, a)$ where a is the split action with respect to $X^{(1)}$. In this case, BRANCHES can already deduce that:

$$\mathcal{Q}^*(\Omega, a) = \mathcal{Q}(\Omega, a) = -\lambda + \underbrace{\mathbb{P}[\Omega(X) = 1]}_{=1} > \mathbb{P}[\Omega(X) = 1, k^*(\Omega) = Y] = \mathcal{Q}^*(\Omega, \overline{a})$$

and therefore that Ω is complete and $a = \text{Argmax}_{a' \in \mathcal{A}(\Omega)} \mathcal{Q}^*(\Omega, a')$.

Consider now the binary encoding of $\mathcal{X}$, this yields a new feature space $\mathcal{X}' = \{0, 1\} \times \{0, 1\} \times \{0, 1\} \times \{0, 1\}$ where the new features $X'^{(1)}, X'^{(2)}, X'^{(3)}, X'^{(4)}$ describe the existence of a category or its absence:

$$\forall i \in \{1, 2, 3, 4\} : X'^{(i)} = \mathbb{1}\{X^{(1)} = i\}$$

Fig. 4.10 depicts the new optimal sparse DT on $\mathcal{X}'$. In this setting, BRANCHES can no longer achieve optimality from the first iteration, because the first iteration only explores branches of size 1 and the optimal solution includes also branches of sizes 2. Moreover, this type of binary encoding introduces unnecessary branches that make the search space larger than necessary, thereby wasting some of the search time. To see this, consider the branch:

$$l' = \mathbb{1}\{X'^{(1)} = 1\} \wedge \mathbb{1}\{X'^{(2)} = 1\}$$

This branch exists in the new search space of branches constructed on $\mathcal{X}'$ and it could be explored at some point by the search algorithm. However, this would be a waste of time because l' does

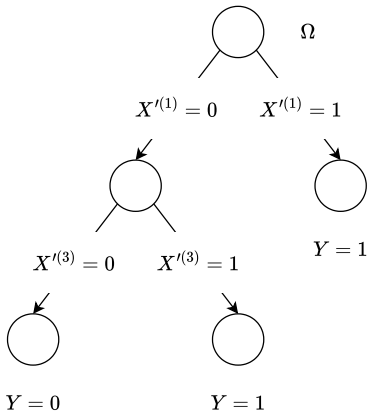


Figure 4.10: The new optimal sparse DT on the new feature space $\mathcal{X}'$.

not describe a possible subset of $\mathcal{X}$. Indeed, translating l' to its corresponding branch on $\mathcal{X}$ yields:

$$l = \mathbb{1}\{X^{(1)} = 1\} \wedge \mathbb{1}\{X^{(1)} = 2\}$$

which always valuates to 0 for any datum $X \in \mathcal{X}$. As a consequence, l can never be part of the optimal solution, in fact, it can never be part of any Decision Tree on $\mathcal{X}$, l is not even a proper branch by virtue of the definition in Section 3.2.1 as it violates condition in (3.1). While it is true that the algorithm will prune this branch because its support set is null, it still has to waste time calculating this support set. To properly evaluate the computational inefficiency induced by this binary encoding, we analyse the number of these introduced unnecessary branches. Theorem 14 provides this analysis for the case where all features have an equal number categories.

Theorem 14. *Consider a classification problem where all features share the same number of categories C, i.e. $\mathcal{X} = \{1, \dots, C\}^q$. Performing a binary encoding on $\mathcal{X}$ yields the new feature space $\mathcal{X}' = \{0,1\}^{qC}$. We define an unnecessary branch l on $\mathcal{X}'$ as a branch that valuates to 0 for any input vector $X \in \mathcal{X}$:*

$$\forall X \in \mathcal{X} : l(X) = 0$$

The number of unnecessary branches introduced by binary encoding is:

$$\mathcal{U}(q, C) = \mathcal{A}(q, C) - \mathcal{B}(q, C) = 3^{qC} - [2C + 1]^q$$

There is a subtlety here. We define l on $\mathcal{X}'$, which means that it involves clauses defined with the features of $\mathcal{X}'$, and yet the definition in Theorem 14 pertains to valuating l on inputs from the feature space $\mathcal{X}$. There is no mistake or lack of rigour in this definition, we are allowed to do this because the Binary Encoding is an injective map from $\mathcal{X}$ to $\mathcal{X}'$, thus implicitly, valuating l on an input $X \in \mathcal{X}$ is defined as valuating l on the image of X in $\mathcal{X}'$ with this map.

Fig. 4.11 is a contour plot of the number of unnecessary branches as a function of q and C in Logarithmic scale. It shows how immense this number becomes as q and C increase. We should note that, not all of these unnecessary branches will be explored by BRANCHES, in fact many of them (depending on the problem) will be pruned pre-emptively. Nevertheless, many

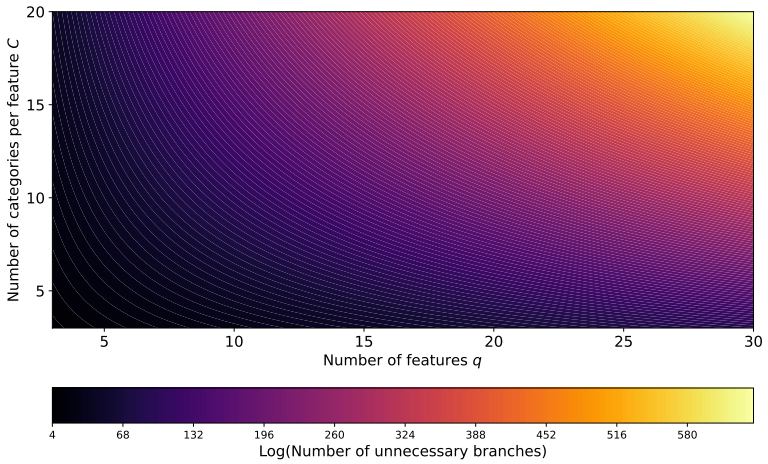


Figure 4.11: The number of unnecessary branches introduced by Binary Encoding.

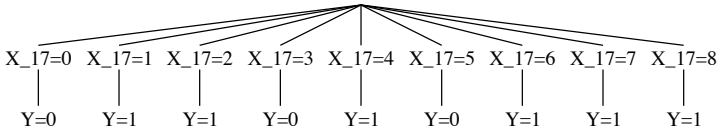


Figure 4.12: Optimal sparse DT for mushroom-o.

will inevitably be visited, which hinders the search efficiency as we clearly demonstrated in Table 4.2. This inefficiency is most apparent on the mushroom dataset, where GOSDT and BRANCHES reach timeout on the binary encoding. However, in contrast, when applied to the ordinal encoding of mushroom, BRANCHES achieves an extremely fast optimal convergence in only 0.16s and 6 iterations. Moreover, in the mushroom dataset case, the optimal sparse DT only involves one split as depicted in Fig. 4.12. This solution can further be collapsed into a single split DT with only two leaves as shown in Fig. 4.13, which is a highly interpretable solution. On the other hand, binary encoding does not allow for such flexibility and its solution, depicted in Fig. 4.14, is clearly less interpretable than the one in Fig. 4.13.

When using binary encoding, it is usually a good idea to drop one category per feature. The reason pertains to reducing the number of the resulting binary features, which in turn makes it easier for the algorithm to quickly find the optimal sparse DT. Notice in Table 4.2 the difference in runtimes between dropping and not dropping a category. In some cases like tic-tac-toe/tic-tac-toe-f, nursery/nursery-f and balance/balance-f, just dropping the first category pushes BRANCHES to terminate in time while it reaches timeout otherwise. However, choosing the adequate category to drop is not trivial and can lead to widely varying solutions and difficulties. We illustrate this point with monk1-l and monk1-f. Fig. 4.15 and Fig. 4.16 show very different optimal sparse DTs, with the option of dropping the last category leading to a solution with only 7 splits while dropping the first category induces a solution with 17 splits. Obviously in this case we prefer dropping the last category, but there is no trivial way of knowing this beforehand. On the other hand, employing a direct ordinal encoding yields the optimal solution in Fig. 4.17 with 10 splits. By noticing sibling branches that share the same sub-DTs rooted in them, this solution can be collapsed into the DT in Fig. 4.18 reducing its number of splits to 7. In this case, ordinal encoding allowed us to achieve a solution of similar quality to the one induced by monk1-l without the need to guess an adequate category to drop, and also while terminating significantly faster.

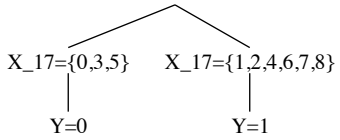


Figure 4.13: Collapsed optimal sparse DT for mushroom-o.

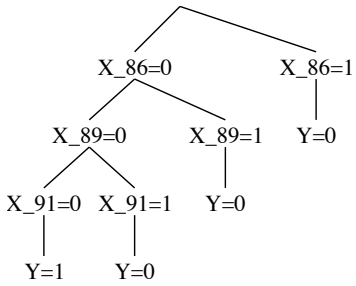


Figure 4.14: Optimal sparse DT for mushroom.

We note that this section analyses one-hot encoding specifically, which is the type of binary encoding that is used most often, including in the works we compare with. However, there exist other types of binary encoding that can alleviate some of the issues we discussed. In the example of Fig. 4.9, choosing a binary encoding of the form:

$$X'^{(1)} = \mathbb{1}\{X^{(1)} \in \{1,3\}\} = \mathbb{1}\{X^{(1)} = 1\} + \mathbb{1}\{X^{(1)} = 3\}$$

indeed yields an optimal sparse DT with only one split. However, this would necessitate knowing this specific scheme of binary encoding before running the algorithm, which is an unreasonable assumption. To circumvent this issue, we can rather consider all possible binary encodings as follows:

$$\forall C \subset \{1,\dots,4\} : X'^{(C)} = \mathbb{1}\{X^{(1)} \in C\} = \sum_{j \in C} \mathbb{1}\{X^{(1)} = j\} \tag{4.8}$$

Among these new features, feature $X'^{(1)} = \mathbb{1}\{X^{(1)} = 1\} + \mathbb{1}\{X^{(1)} = 3\}$ exists and thus can be chosen to yield an optimal sparse DT with only one split. Nonetheless, the type of binary encoding depicted in (4.8) suffers from a severe limitation, the number of such features is 2^4 in this simple example, which is still manageable. However, for a feature space $\mathcal{X} = \prod_{i=1}^{q}\{1,\dots,C_i\}$, the number of these encoded binary features amounts to $\sum_{i=1}^{q} 2^{C_i}$, which quickly becomes computational unfeasible to handle for any search algorithm, hence the reason most works consider one-hot encoding. Furthermore, if we change the example in Fig. 4.9 to a multiclass classification problem where $Y = X^{(1)}$ then ordinal encoding would still yield an optimal sparse DT with only one split, while any type of binary encoding would fail as it would necessitate at least 3 splits to induce 4 leaves in order to perfectly classify the data.

To conclude, we demonstrated in this section the reason why finding an optimal sparse DT is significantly faster via ordinal encoding than binary encoding. Moreover, we showed that these faster to get solutions can be even more desirable from an interpretability standpoint, we showcased this for the mushroom dataset. Using the monk1 dataset example, we explained the notion of collapsing the non-binary DT solution to yield better interpretable alternatives. This notion of collapse should be further investigated in the future. It could be of great use from a scalability perspective. Indeed, for large datasets, using a preliminary binary encoding can

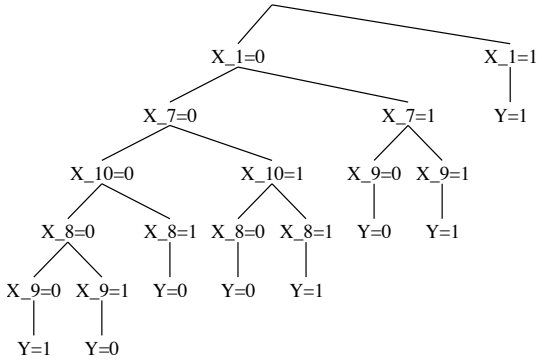


Figure 4.15: Optimal sparse DT for monk1-l, it has 7 splits.

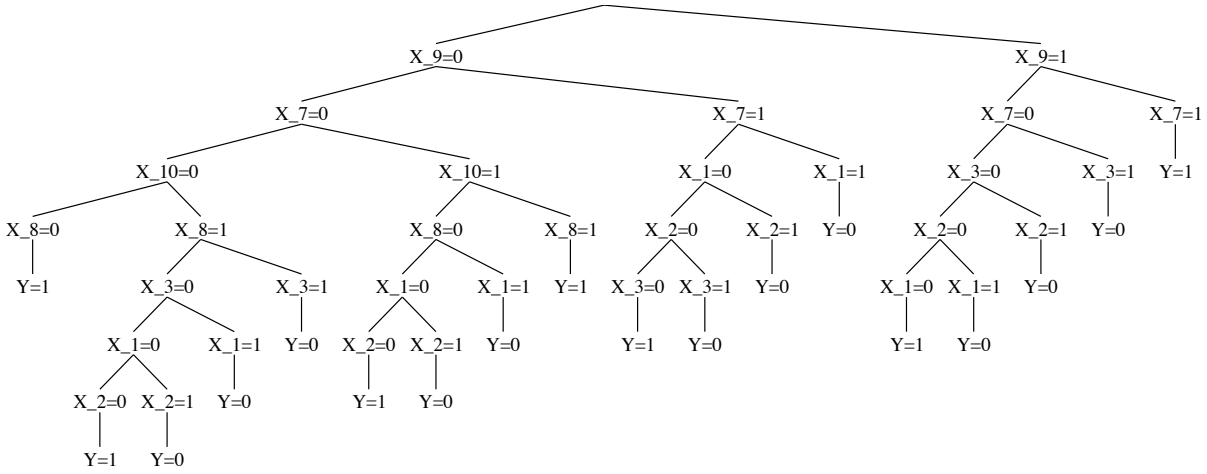


Figure 4.16: Optimal sparse DT for monk1-f, it has 17 splits.

greatly amplify the difficulty of the already challenging optimisation task. The idea would be to rather utilise ordinal encoding to quickly find a non-binary solution, and then transform this solution, through some form of collapse, into a more interpretable version, maybe even a binary DT depending on how we conduct the collapsing process.

4.9 Conclusion

In this chapter, we introduced BRANCHES, a novel BFS method for seeking optimal sparse DTs. Central to its efficiency is its AO*-type search strategy and the purification bound that effectively prunes the search space. We proved that BRANCHES solves for sparsity, as it always finds the optimal sparse DT whenever it terminates. Moreover, we also analysed its computational complexity by deriving upper bounds on the number of branch evaluations, and showing the superiority of this bound to a similar analysis from the literature (Hu et al., 2019, Theorem E.2). These theoretical findings were further validated via an in-depth empirical comparison of BRANCHES with the state of the art, showing a clear advantage for BRANCHES even from a runtime standpoint, which is hard to achieve against C++ competitors given its current Python implementation. Section 4.7.4 further supports this point. With this in mind, we propose the following ideas for future work:

- The first straightforward idea is to implement BRANCHES in C++. This can already yield significant improvements by increasing the number of iterations BRANCHES can perform

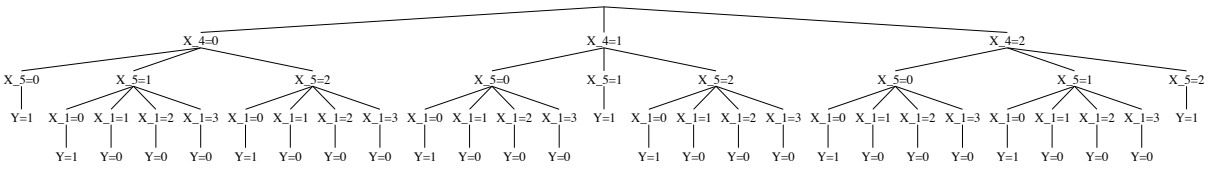


Figure 4.17: Optimal sparse DT for monk1-o, it has 10 splits.

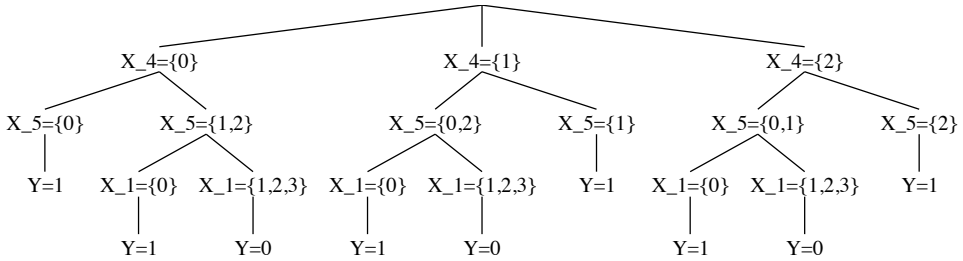


Figure 4.18: Collapsed optimal sparse DT for monk1-o, it has 7 splits similarly to the solution induced for monk1-l.

per unit time.

- The second idea stems from the findings of Section 4.7.3 where we observed the crucial importance of the depth 2 solver to STreeD. This motivates us to adapt this idea to the purification bound and incorporate it in BRANCHES' search strategy.

- BRANCHES is amenable to parallel computing. Indeed, during the Selection step, whenever we take a split action, instead of choosing only one branch among the resulting children branches, we could rather choose a subset of the children branches and keep running Selection from each of them in parallel. As a consequence, the Selection step would return a subset of branches, for which we run Expansion in parallel then Backpropagation in a synchronous parallel fashion. This would yield a notably more aggressive pruning of the search space per iteration and potentially result in a significantly faster termination. To benefit from the full potential of this parallel computing scheme, a C++ implementation is necessary.

- While the purification bound implies the optimality of BRANCHES in all settings, it could be loose and result in a poor pruning of the search space in worst-case scenarios. Especially in the presence of a large number of features, as we have seen in the mushroom datasets. To alleviate this issue, one idea is to estimate the upper bounds using an auxiliary strong classifier, such as a Multi-Layer Perceptron (MLP). Concretely, given a branch l, we would fit an MLP f on the data $\mathcal{D}_l$ contained in l. Then, if $1 \leq m \leq q$ is the number of relevant features used by f (which could be obtained via Lasso regularisation of the first layer of the MLP f for example), we would have the following new tighter upper bound:

$$\mathcal{V}(l) \leq \max\left\{\mathcal{H}(l), -m\lambda + \mathbb{P}\left[l(X) = 1, f(X) = Y\right]\right\} \leq \max\left\{\mathcal{H}(l), -\lambda + \mathbb{P}\left[l(X) = 1\right]\right\}$$

The intuition behind this bound is that an optimal sub-DT T rooted in l necessarily uses at least m splits (in order to employ all the relevant m features), thus incurring a cost $-m\lambda$, and it yields an accuracy at most:

$$\mathbb{P}\left[l(X) = 1, T(X) = Y\right] \leq \mathbb{P}\left[l(X) = 1, f(X) = Y\right]$$

under the assumption that f is a strong classifier, i.e. it exhibits better accuracy than any DT. This tighter bound would yield a more substantial pruning of the search space and thus potentially result in a significantly faster termination. However, it remains to assess whether the added cost of training the MLP outweighs this induced gain in pruning.

Lastly, we would like to come back to a fundamental limitation of Branches (and BFS methods) compared to DFS, which is the higher memory consumption. While an efficient search strategy can certainly mitigate this problem by terminating early, before running out of memory, on many cases, there are still situations with very large datasets and number of features where this could be inevitable. We have not analysed such situations in our experiments, but we recognise their existence and as a solution we suggest for a future work to include a hybrid component to Branches which switches to Backtracking search once an allocated amount of memory is reached. This idea of hybrid search is discussed in (Pearl, 1984, Section 2.5.1) under the name BF-BT.

4.10 Appendix: Table of Notation

Table 4.5: Table of Notation

X	$=$	$\left(X^{(1)}, \ldots, X^{(q)}\right)$, an input of features.
$X^{(i)}$	$\in$	$\{1, \ldots, C_i\}$, a feature.
Y	$\in$	$\{1, \ldots, K\}$, a class.
$\mathcal{X}$	$=$	$\prod_{i=1}^{q}\{1, \ldots, C_i\}$ the features space.
$\mathcal{D}$	$=$	$\{(X_m, Y_m)\}_{m=1}^{n}$, Dataset of examples.
l	$=$	$\bigwedge_{v=1}^{\mathcal{S}(l)} \mathbb{1}\left\{X^{(i_v)} = j_v\right\}$ a branch. Also, a unit-state in our MDP.
$\mathcal{S}(l)$	$\triangleq$	The size of branch l, the number of splits in l, the number of clauses in l.
$l(X)$	$\triangleq$	Valuation of l for input X. When $l(X) = 1$, we say that X is in l.
Ω	$\triangleq$	The root. Branch that valuates to 1 for all possible inputs.
$\mathrm{Ch}(l, i)$	$\triangleq$	Children of l when splitting with respect to feature i.
$\mathrm{Ch}(l, i)$	$=$	$\{l_1, \ldots, l_{C_i}\}$, $l_j = l \wedge \mathbb{1}\left\{X^{(i)} = j\right\}$
$n(l)$	$\triangleq$	Number of examples in l.
$n_k(l)$	$\triangleq$	Number of examples in l of class k.
$k^*(l)$	$=$	$\mathrm{Argmax}_{1\leq k\leq K}\{n_k(l)\}$, majority class in l
$\mathbb{P}[l(X) = 1]$	$\triangleq$	Empirical probability that X is in l.
$\mathcal{H}(l)$	$=$	$\mathbb{P}[l(X) = 1, Y = k^*(l)]$, probability that an example is in l and is correctly classified.
sub-DT rooted in l	$\triangleq$	Collection of branches partitioning l, it stems from a series of splits of l. Also a state in our MDP.
DT	$\triangleq$	A sub-DT rooted in Ω.
$T(X)$	$\triangleq$	Predicted class of X by T. Majority class of the branch containing X.
$\mathcal{H}(T)$	$=$	$\mathbb{P}[T(X) = Y]$, accuracy of DT T.
$\mathcal{H}_\lambda(T)$	$\triangleq$	Regularized Objective function of DT T.
$\mathcal{H}_\lambda(T)$	$=$	$\mathbb{P}[T(X) = Y] - \lambda\mathcal{S}(T)$.
$\mathcal{S}(T)$	$\triangleq$	Number of splits to construct sub-DT T from the branch where it is rooted.
λ	$\in$	$[0, 1]$, penalty parameter.
T^*	$=$	$\mathrm{Argmax}_T\{\mathcal{H}_\lambda(T)\}$, optimal DT.
$\mathcal{A}(T)$	$\triangleq$	Action space at state T.
$\overline{a}$	$\triangleq$	Terminal action.
$T \xrightarrow{a} T'$	$\triangleq$	Transition from T to T' through action a.
$\overline{l}$	$\triangleq$	Absorbing state, $l \xrightarrow{\overline{a}} \overline{l}$.
$r(T, a)$	$\triangleq$	Reward of taking action a in state T.
π	$\triangleq$	Policy, maps each state T to an action $\pi(T) \in \mathcal{A}(T)$.
$\mathcal{V}^\pi(T)$	$\triangleq$	Value of policy π starting from T.
$\mathcal{Q}^\pi(T, a)$	$\triangleq$	State-action value of policy π at state-action pair (T, a).
T_l^π	$\triangleq$	Sub-DT of π rooted in l. See Proposition 5.
T^π	$\equiv$	T_Ω^π
π^*	$=$	$\mathrm{Argmax}_\pi \mathcal{V}^\pi(\Omega)$, the optimal policy.
T^*	$=$	T^{π^*}
$\mathcal{V}^*$	$\equiv$	$\mathcal{V}^{\pi^*}$
$\mathcal{Q}^*$	$\equiv$	$\mathcal{Q}^{\pi^*}$
$\mathcal{V}(T)$	$\triangleq$	Estimated upper bound on $\mathcal{V}^*(T)$.
$\mathcal{Q}(T, a)$	$\triangleq$	Estimated upper bound on $\mathcal{Q}^*(T, a)$

4.11 Appendix: Proofs

Proposition 5. *For any policy π and unit-state l, there exists $\tau \geq 0$ such that for any $t \geq \tau$, $T_t = \{\overline{l_1}, \ldots, \overline{l_{|T_\tau|}}\}$ is an absorbing state. In which case we call $T_l^\pi = \{l_1, \ldots, l_{|T_\tau|}\}$ the sub-DT of π rooted in l. If $l = \Omega$ we abbreviate the notation $T_\Omega^\pi \equiv T^\pi$ and call T^π the DT of π.*

Proof Let l be a unit-state, π a policy, and $(T_t)_{t=0}^\infty$ such that:

$$\begin{cases} T_0 = l \\ \forall t \geq 0 : T_t \xrightarrow{\pi(T_t)} T_{t+1} \end{cases}$$

The proof is conducted by induction on $q - \mathcal{S}(l) \in \{0, \ldots, q\}$, where we recall that q is the number of features.

If $q - \mathcal{S}(l) = 0$, then $\mathcal{A}(l) = \{\overline{a}\}$ and $\pi(l) = \overline{a}$. Therefore $T_1 = \overline{l}$ and we deduce that the proposition holds with $\tau = 1$.

Inductive hypothesis: Suppose that the proposition is true for $q - \mathcal{S}(l') = n \in \{0, \ldots, q-1\}$, and let us show that it is true for $q - \mathcal{S}(l) = n + 1$.

If $\pi(l) = \overline{a}$, then $T_1 = \overline{l}$ and the proposition holds.
Otherwise $\pi(l)$ is a split action, and we have $l \xrightarrow{\pi(l)} T_1 = \{l_1, \ldots, l_{|T_1|}\}$ where:

$$\forall u \in \{1, \ldots, |T_1|\} : q - \mathcal{S}(l_u) = n$$

Therefore, the proposition is true for all l_u.
Let us now denote the following:

$$\begin{cases} T_1^{(u)} = l_u \\ \forall t \geq 1 : T_t^{(u)} \xrightarrow{\pi} T_{t+1}^{(u)} \end{cases}$$

According to the proposition:

$$\exists \tau_u \geq 0, \forall t \geq \tau_u : T_t^{(u)} = \left\{\overline{l_1^{(u)}}, \ldots, \overline{l_{|T_{\tau_u}^{(u)}|}^{(u)}}\right\}$$

By taking $\tau = \max_{1 \leq u \leq |T_1|}\{\tau_u\}$, we get:

$$\forall t \geq \tau, \forall u \in \{1, \ldots, |T_1|\} : T_t^{(u)} = \left\{\overline{l_1^{(u)}}, \ldots, \overline{l_{|T_j^{(u)}|}^{(u)}}\right\} = \left\{\overline{l_1^{(u)}}, \ldots, \overline{l_{|T_\tau^{(u)}|}^{(u)}}\right\}$$

On the other hand $\forall t \geq 1 : T_t = \bigcup_{u=1}^{|T_1|} T_t^{(u)}$, thus:

$$\forall t \geq \tau : T_t = \bigcup_{u=1}^{|T_1|} \left\{\overline{l_1^{(j)}}, \ldots, \overline{l_{|T_\tau^{(u)}|}^{(u)}}\right\}$$

Which concludes the inductive proof. ■

Proposition 6. *For any policy π and unit-state l, the value of π from l satisfies:*

$$\mathcal{V}^{\pi}(l) = \mathcal{H}_{\lambda}(T_l^{\pi}) = -\lambda \mathcal{S}(T_l^{\pi}) + \mathcal{H}(T_l^{\pi})$$

In particular $\mathcal{V}^{\pi}(\Omega) = \mathcal{H}_{\lambda}(T^{\pi}) = -\lambda \mathcal{S}(T^{\pi}) + \mathcal{H}(T^{\pi})$.

Proof Let l be a unit-state, π a policy, and $(T_t)_{t=0}^{\infty}$ such that:

$$\begin{cases} T_0 = l \\ \forall t \geq 0 : T_t \xrightarrow{\pi(T_t)} T_{t+1} \end{cases}$$

By Induction on $\mathcal{S}(T_l^{\pi})$:
If $\mathcal{S}(T_l^{\pi}) = 0$, then $\pi(l) = \overline{a}$ and $\forall t \geq 1 : T_t = \overline{l}$. Thus:

$$\mathcal{V}^{\pi}(l) = \underbrace{r(l, \overline{a})}_{=\mathcal{H}(l)} + \sum_{t \geq 1} \underbrace{r\left(\overline{l}, \pi\left(\overline{l}\right)\right)}_{=0} = \mathcal{H}(l)$$

On the other hand $T_l^{\pi} = l$, therefore:

$$\mathcal{H}_{\lambda}(T_l^{\pi}) = -\lambda \underbrace{\mathcal{S}(T_l^{\pi})}_{=0} + \mathcal{H}(l) = \mathcal{H}(l)$$

Hence $\mathcal{V}^{\pi}(l) = \mathcal{H}_{\lambda}(T_l^{\pi})$

Inductive hypothesis: Suppose the proposition is true up to $\mathcal{S}(T_l^{\pi}) = n \geq 0$ and let us prove it for $\mathcal{S}(T_l^{\pi}) = n + 1$

If $\pi(l) = \overline{a}$ then we have again:

$$\mathcal{V}^{\pi}(l) = \underbrace{r(l, \overline{a})}_{=\mathcal{H}(l)} + \sum_{t \geq 1} \underbrace{r\left(\overline{l}, \pi\left(\overline{l}\right)\right)}_{=0} = \mathcal{H}(l)$$

Since $T_l^{\pi} = l$, then:

$$\mathcal{H}_{\lambda}(T_l^{\pi}) = -\lambda \underbrace{\mathcal{S}(T_l^{\pi})}_{=0} + \mathcal{H}(l) = \mathcal{H}(l)$$

Hence $\mathcal{V}^{\pi}(l) = \mathcal{H}_{\lambda}(T_l^{\pi})$

Now suppose that $\pi(l)$ is a split action. We have the following:

$$\begin{aligned} \mathcal{V}^{\pi}(l) &= r(l, \pi(l)) + \sum_{t=1}^{\infty} r(T_t, T_{t+1}) \\ &= r(l, \pi(l)) + \mathcal{V}^{\pi}(T_1) \\ &= r(l, \pi(l)) + \sum_{u=1}^{|T_1|} \mathcal{V}^{\pi}(l_u) \end{aligned}$$

Where $T_1 = \{l_1, \dots, l_{|T_1|}\}$. We know that:

$$\begin{aligned}\forall u \in \{1, \dots, |T_1|\} : \mathcal{V}^\pi(l_u) &= \mathcal{H}_\lambda(T^\pi_{l_u}) \\ \implies \mathcal{V}^\pi(l) &= -\lambda + \sum_{u=1}^{|T_1|} \Big\{ -\lambda \mathcal{S}(T^\pi_{l_u}) + \mathcal{H}(T^\pi_{l_u}) \Big\} \\ &= -\lambda \Big\{ 1 + \sum_{u=1}^{|T_1|} \mathcal{S}(l_u) \Big\} + \mathcal{H}(T^\pi_l)\end{aligned}$$

We know that the total number of splits to construct T^π_l is 1 (corresponding to the split $\pi(l)$) plus the sum of the number of splits required to construct each sub-DT $T^\pi_{l_u}$, i.e.

$$\mathcal{S}(T^\pi_l) = 1 + \sum_{u=1}^{|T_1|} \mathcal{S}(T^\pi_{l_u})$$

Therefore we deduce that:

$$\mathcal{V}^\pi(l) = -\lambda \mathcal{S}(T^\pi_l) + \mathcal{H}(T^\pi_l) = \mathcal{H}_\lambda(T^\pi_l)$$

Which concludes the inductive proof. ■

Proposition 7 (Purification Bound)**.** *For any non-absorbing unit-state l and split action $a \in \mathcal{A}(l) \setminus \{\overline{a}\}$, we define the Purification Bound estimates:*

$$\mathcal{Q}(l, a) = -\lambda + \mathbb{P}[l(X) = 1] = -\lambda + \frac{n(l)}{n} \tag{4.4}$$

$$\mathcal{V}(l) = \max\{\mathcal{H}(l), -\lambda + \mathbb{P}[l(X) = 1]\} = \max\left\{\frac{n_{k^*(l)}(l)}{n}, -\lambda + \frac{n(l)}{n}\right\} \tag{4.5}$$

Then the estimates $\mathcal{Q}(l, a)$ and $\mathcal{V}(l)$ are upper bounds on $\mathcal{Q}^*(l, a)$ and $\mathcal{V}^*(l)$ respectively.

Proof Let l be a non-terminal unit-state, $a \in \mathcal{A}(l) \setminus \{\overline{a}\}$ and:

$$\mathcal{Q}(l, a) = -\lambda + \mathbb{P}[l(X) = 1]$$

Let us show that $\mathcal{Q}(l, a) \geq \mathcal{Q}^*(l, a)$. Consider $l \xrightarrow{\pi(l)} T_1 = \{l_1, \dots, l_{|T_1|}\}$, we have the following:

$$\mathcal{Q}^*(l, a) = -\lambda + \sum_{u=1}^{|T_1|} \mathcal{V}^*(l_u)$$

According to Proposition 6, we have:

$$\begin{aligned}\forall u \in \{1, \dots, |T_1|\} : \mathcal{V}^*(l_u) &= \mathcal{H}_\lambda(T^*_{l_u}) \\ \mathcal{Q}^*(l, a) &= -\lambda + \sum_{u=1}^{|T_1|} \mathcal{H}_\lambda(T^*_{l_u})\end{aligned}$$

On the other hand, we have:

$$\begin{aligned}\forall u \in \{1,\ldots,|T_1|\} : \mathcal{H}_\lambda\left(T^*_{l_u}\right) &= -\lambda\mathcal{S}\left(T^*_{l_u}\right) + \mathcal{H}\left(T^*_{l_u}\right)\\ &\leq \mathcal{H}\left(T^*_{l_u}\right)\\ &\leq \mathbb{P}\left[l_u\left(X\right)=1, T^*_{l_u}\left(X\right)=Y\right]\\ &\leq \mathbb{P}\left[l_u\left(X\right)=1\right]\end{aligned}$$

Which implies the following:

$$\mathcal{Q}^*\left(l,a\right) \leq -\lambda + \sum_{u=1}^{|T_1|}\mathbb{P}\left[l_u\left(X\right)=1\right] \leq -\lambda + \mathbb{P}\left[l\left(X\right)=1\right] = \mathcal{Q}\left(l,a\right)$$

For the optimal value function, we have:

$$\begin{aligned}\mathcal{V}^*\left(l\right) &= \max_{a\in\mathcal{A}(l)}\mathcal{Q}^*\left(l,a\right)\\ &= \max_{a\in\mathcal{A}(l)\setminus\{\overline{a}\}}\left\{\mathcal{Q}^*\left(l,\overline{a}\right), \mathcal{Q}^*\left(l,a\right)\right\}\\ &\leq \max_{a\in\mathcal{A}(l)\setminus\{\overline{a}\}}\left\{\mathcal{H}\left(l\right), \mathcal{Q}\left(l,a\right)\right\}\\ &\leq \max\left\{\mathcal{H}\left(l\right), -\lambda + \mathbb{P}\left[l\left(X\right)=1\right]\right\}\end{aligned}$$

■

Lemma 15. *For any unit-state l and action $a \in \mathcal{A}\left(l\right)$, the estimate $\mathcal{Q}\left(l,a\right)$ is an upper bound on the optimal state values.*

$$\mathcal{Q}\left(l,a\right) \geq \mathcal{Q}^*\left(l,a\right)$$

Proof For the terminal action, we always have:

$$\mathcal{Q}\left(l,\overline{a}\right) = \mathcal{H}\left(l\right) = \mathcal{Q}^*\left(l,\overline{a}\right)$$

Let us now consider a split action $a \in \mathcal{A}\left(l\right) \setminus \{\overline{a}\}$. We have the following:

$$\mathcal{Q}\left(l,a\right) = -\lambda + \sum_{u=1}^{|T_1|}\mathcal{V}\left(l_u\right)$$

Where $l \xrightarrow{a} T_1 = \{l_1,\ldots,l_{|T_1|}\}$. It suffices to show that:

$$\forall u \in \{1,\ldots,|T_1|\} : \mathcal{V}\left(l_u\right) \geq \mathcal{V}^*\left(l_u\right)$$

We define the following policy:

$$\begin{cases}\pi\left(l'\right) = \text{Argmax}_{a'\in\mathcal{A}(l')}\mathcal{Q}\left(l',a'\right) \text{ for } l' \text{ that have been visited.}\\ \pi\left(l'\right) = \overline{a} \text{ for } l \text{ that have never been visited.}\end{cases}$$

The proof now proceeds by induction on the number of visits of l which we denote here $v\left(l\right) \geq 0$. If $v\left(l\right) = 0$, then:

$$\mathcal{V}\left(l\right) = \max\left\{\mathcal{H}\left(l\right), -\lambda + \mathbb{P}\left[l\left(X\right)=1\right]\right\} \geq \mathcal{V}^*\left(l\right)$$

Induction hypothesis: Suppose that this is true for any number of visits $\leq n$ where $n \geq 0$, and let us show that the result still holds for $v(l) = n+1$. We have

$$\mathcal{V}(l) = \max\{\mathcal{H}(l), -\lambda + \sum_{u=1}^{|T_1|} \mathcal{V}(l_u)\}$$

On the other hand

$$\begin{aligned} &\forall u \in \{1, \dots, |T_1|\} : v(l_u) \leq n \\ \implies &\forall u \in \{1, \dots, |T_1|\} : \mathcal{V}(l_u) \geq \mathcal{V}^*(l_u) \end{aligned}$$

Thus

$$\mathcal{V}(l) \geq \max\{\mathcal{H}(l), -\lambda + \sum_{u=1}^{|T_1|} \mathcal{V}^*(l_u)\} = \mathcal{V}^*(l)$$

Which concludes the inductive proof, and we get that:

$$\forall u \in \{1, \dots, |T_1|\} : \mathcal{V}(l_u) \geq \mathcal{V}^*(l_u)$$

Implying

$$\mathcal{Q}(l, a) = -\lambda + \sum_{u=1}^{|T_1|} \mathcal{V}(l_u) \geq \mathcal{Q}^*(l, a)$$

Remark: During the inductive reasoning, we used the fact that the number of visits of children branches is lower than the number of visits of their parent branch. However, this is not true when Dynamic Programming is considered. Indeed, due to memoisation, some children branches could have been visited more than their parents. The result still stems from a similar induction, albeit through a more technical proof. The general idea is that, for children branches l_u that are visited more than $n+1$ times, we consider their children, and so on, until we arrive at descendant branches that are either visited less than n times or that are terminal. In both cases $\mathcal{V}(l_u) \geq \mathcal{V}^*(l_u)$, and we backpropagate this result to $\mathcal{V}(l)$. ■

Theorem 8 (Optimality of BRANCHES)**.** *When BRANCHES terminates, the optimal policy is the greedy policy with respect to the estimated state-action values $\mathcal{Q}(l, a)$, i.e.*

$$\pi^*(T) = \text{Argmax}_{a \in \mathcal{A}(T)} \mathcal{Q}(T, a)$$

Proof Define the policy $\tilde{\pi}(T) = \text{Argmax}_{\mathcal{A}(T)} \mathcal{Q}(T, a)$. First, we show that for any unit-state l, if l is complete and $a^* = \text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a)$, then $a^* = \pi^*(l)$
Since l is complete, we have $\mathcal{Q}(l, a^*) = \mathcal{Q}^*(l, a^*)$. By Lemma 15, we get

$$\begin{aligned} \forall a \in \mathcal{A}(l) : \mathcal{Q}^*(l, a^*) = \mathcal{Q}(l, a^*) \geq \mathcal{Q}(l, a) \geq \mathcal{Q}^*(l, a) \\ \implies a^* = \text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}^*(l, a) = \pi^*(l) \end{aligned}$$

On the other hand, l is complete if and only if $(l, \pi^*(l))$ is complete, which is satisfied if and only if for all $u \in \{1, \dots, |T|\} : l_u$ is complete, where $l \xrightarrow{\pi^*(l)} T = \{l_1, \dots, l_{|T|}\}$.
BRANCHES terminates when Ω is complete. Let us define the following:

$$\begin{cases} T_0 = \Omega \\ \forall t \geq 0 : T_t \xrightarrow{\tilde{\pi}(T_t)} T_{t+1};\ T_t = \{l_1^{(t)}, \dots, l_{|T_t|}^{(t)}\} \end{cases}$$

Since Ω is complete, we have shown $\tilde{\pi}(\Omega) = \pi^*(\Omega)$, and it follows that:

$$\begin{aligned}
&\forall u \in \{1, \ldots, |T_1|\} : l_u^{(1)} \text{ is complete} \\
\implies &\forall u \in \{1, \ldots, |T_1|\} : \tilde{\pi}\left(l_u^{(1)}\right) = \pi^*\left(l_u^{(1)}\right) \\
&\vdots \\
\implies &\forall u \in \{1, \ldots, |T_t|\} : l_u^{(t)} \text{ is complete} \\
\implies &\forall u \in \{1, \ldots, |T_t|\} : \tilde{\pi}\left(l_u^{(t)}\right) = \pi^*\left(l_u^{(t)}\right) \\
&\vdots
\end{aligned}$$

Thus $\tilde{\pi}$ is optimal:

$$\begin{aligned}
\mathcal{V}^*(\Omega) &= \sum_{t=0}^{\infty} r\left(T_t, \pi^*(T_t)\right) \\
&= \sum_{t=0}^{\infty} \sum_{u=1}^{|T_t|} r\left(l_u^{(t)}, \pi^*\left(l_u^{(t)}\right)\right) \\
&= \sum_{t=0}^{\infty} \sum_{u=1}^{|T_t|} r\left(l_u^{(t)}, \tilde{\pi}\left(l_u^{(t)}\right)\right) \\
&= \sum_{t=0}^{\infty} r\left(T_t, \tilde{\pi}(T_t)\right) \\
&= \mathcal{V}^{\tilde{\pi}}(\Omega)
\end{aligned}$$

■

Lemma 16. *A branch l can be chosen for Expansion only if there exists a DT T such that:*

$$\begin{cases} l \in T \setminus L \\ -\lambda \mathcal{S}(T) + \sum_{l' \in L} \mathcal{H}(l') + \sum_{l' \in T \setminus L} \{ -\lambda + \mathbb{P}[l'(X) = 1] \} \geq -\lambda \mathcal{S}(T^*) + \mathcal{H}(T^*) \end{cases}$$

Where $L = \{l' \in T : \mathcal{H}(l') \geq -\lambda + \mathbb{P}[l'(X) = 1]\}$.

Proof Let $\tilde{\pi}$ be the Selection policy, i.e. for any unit-state l:

$$\tilde{\pi}(l) = \begin{cases} \overline{a} \text{ If } l \text{ has never been visited} \\ \text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a) \text{ Otherwise.} \end{cases}$$

For the current Selection policy $\tilde{\pi}$, a branch l is chosen for Expansion only if $l \in T^{\tilde{\pi}}$, thus let us analyse the properties of $T^{\tilde{\pi}}$.

By the definition of $\tilde{\pi}$, $T^{\tilde{\pi}}$ maximises $\mathcal{V}(T)$:

$$\begin{aligned} \forall \text{ DT } T : \mathcal{V}(T) &\leq \mathcal{V}\left(T^{\tilde{\pi}}\right) \\ \implies \mathcal{V}(T^*) &\leq \mathcal{V}\left(T^{\tilde{\pi}}\right) \\ \implies \mathcal{V}^*(T^*) &\leq \mathcal{V}\left(T^{\tilde{\pi}}\right) \\ \implies -\lambda\mathcal{S}(T^*) + \mathcal{H}(T^*) &\leq \mathcal{V}\left(T^{\tilde{\pi}}\right) \end{aligned}$$

On the other hand we have:

$$\mathcal{V}\left(T^{\tilde{\pi}}\right) = \sum_{l \in T^{\tilde{\pi}}} \mathcal{V}(l)$$

Let $L = \{l \in L : \mathcal{H}(l) \geq -\lambda + \mathbb{P}[l(X) = 1]\}$. For any $l \in L$ we have $\mathcal{V}(l) = \mathcal{H}(l)$ and for any $l \in T \setminus L$ we have $\mathcal{V}(l) = -\lambda + \mathbb{P}[l(X) = 1]$. Therefore we deduce that:

$$-\lambda\mathcal{S}\left(T^{\tilde{\pi}}\right) + \sum_{l' \in L} \mathcal{H}(l') + \sum_{l' \in T^{\tilde{\pi}} \setminus L} \{-\lambda + \mathbb{P}[l'(X) = 1]\} \geq -\lambda\mathcal{S}(T^*) + \mathcal{H}(T^*)$$

The first condition for a branch l to be considered for Expansion is $l \in T^{\tilde{\pi}}$. For the second condition, l cannot be in L, because all branches in L are complete and satisfy $\overline{a} = \text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}^*(l, a)$. Indeed this is due to the following:

$$\mathcal{Q}^*(l, \overline{a}) = \mathcal{H}(l) \geq -\lambda + \mathbb{P}[l(X) = 1] \geq \mathcal{Q}^*(l, a) \ \forall a \in \mathcal{A}(l)$$

where the last inequality comes from Proposition 7. Now we deduce that the second condition for l to be considered for Expansion is $l \in T \setminus L$. ■

Theorem 10 (Problem-dependent complexity of BRANCHES)**.** *Let $\Gamma(q, C, \lambda)$ denote the total number of branch evaluations performed by BRANCHES for an instance of the classification problem with $q \geq 2$ features, $0 < \lambda \leq 1$ the penalty parameter, and $C \geq 2$ the number of categories per feature. Then, $\Gamma(q, C, \lambda)$ satisfies the following bound:*

$$\Gamma(q, C, \lambda) \leq \sum_{h=0}^{\kappa} (q - h) C^{h+1} \binom{q}{h}; \ \kappa = \min\left\{\left\lfloor \mathcal{S}(T^*) - 1 + \frac{1 - \mathcal{H}(T^*)}{\lambda} \right\rfloor, q\right\}$$

Proof Let l be a branch. According to Lemma 16, for l to be considered for Expansion, there has to exist a DT T such that:

$$\begin{cases} l \in T \setminus L \\ -\lambda\mathcal{S}(T) + \sum_{l' \in L} \mathcal{H}(l') + \sum_{l' \in T \setminus L} \{-\lambda + \mathbb{P}[l'(X) = 1]\} \geq -\lambda\mathcal{S}(T^*) + \mathcal{H}(T^*) \end{cases}$$

where $L = \{l' \in T : \mathcal{H}(l') \geq -\lambda + \mathbb{P}[l'(X) = 1]\}$. Suppose l is such a branch, then we have:

$$\begin{aligned} -\lambda\mathcal{S}(T) + \sum_{l' \in L} \underbrace{\mathcal{H}(l')}_{\leq \mathbb{P}[l'(X)=1]} + \sum_{l' \in T \setminus L} \{-\lambda + \mathbb{P}[l'(X) = 1]\} &\geq -\lambda\mathcal{S}(T^*) + \mathcal{H}(T^*) \\ \implies -\lambda\Big\{\mathcal{S}(T) + |T \setminus L|\Big\} + \sum_{l' \in T} \mathbb{P}[l'(X) = 1] &\geq -\lambda\mathcal{S}(T^*) + \mathcal{H}(T^*) \end{aligned}$$

Since $l \in T \setminus L$, then $|T \setminus L| \geq 1$ and we get:

$$\begin{aligned} -\lambda\Big\{\mathcal{S}(T)+1\Big\}+1 &\geq -\lambda\mathcal{S}(T^*)+\mathcal{H}(T^*) \\ \implies \mathcal{S}(T) &\leq \mathcal{S}(T^*)-1+\frac{1-\mathcal{H}(T^*)}{\lambda} \\ \implies \mathcal{S}(l) &\leq \mathcal{S}(T^*)-1+\frac{1-\mathcal{H}(T^*)}{\lambda} \end{aligned}$$

Let $\mathcal{C} = \left\{l \text{ branch} : \mathcal{S}(l) \leq \mathcal{S}(T^*)-1+\frac{1-\mathcal{H}(T^*)}{\lambda}\right\}$. Then the number of branches that are expanded is upper bounded by $|\mathcal{C}|$.

We recall that we rather seek to upper bound the number of branches that are evaluated, i.e. for which we calculate $\mathcal{H}(l)$. These evaluations happen during the Expansion step of BRANCHES. When a branch l is expanded, we evaluate all of its children. There are $q - \mathcal{S}(l)$ features left to use for splitting l, and for each split, C children branches are created. Thus, there are $(q-\mathcal{S}(l))C$ children of l, hence $(q-\mathcal{S}(l))C$ evaluations happen during the expansion of l. Let us now upper bound $\Gamma(q, C, \lambda)$.

For each branch $l \in \mathcal{C}$:

- We choose $\mathcal{S}(l) \in \left\{0, \ldots, \min\left\{\left\lfloor \mathcal{S}(T^*)-1+\frac{1-\mathcal{H}(T^*)}{\lambda}\right\rfloor, q\right\}\right\}$. The minimum comes from the fact that $l \in \mathcal{C}$ and $\mathcal{S}(l) \leq q$.
- For each $h = \mathcal{S}(l)$, we construct l by choosing h features among the total q features, there are $\binom{q}{h}$ such choices.
- For each choice among the $\binom{q}{h}$ choices, for each feature among the h features, there are C choices of values, therefore there are $C^h\binom{q}{h}$ branches with depth h.
- For each branch of depth h, when it is expanded, $(q-h)C$ evaluations occur.

With these considerations, we deduce that:

$$\Gamma(q, C, \lambda) \leq \sum_{h=0}^{\kappa}(q-h)C^{h+1}\binom{q}{h};\ \kappa = \min\left\{\left\lfloor \mathcal{S}(T^*)-1+\frac{1-\mathcal{H}(T^*)}{\lambda}\right\rfloor, q\right\}$$

■

Corollary 11 (Problem-independent complexity of BRANCHES)**.** *Let $\Gamma(q, \lambda, C)$ be defined as in Theorem 10, then we have:*

$$\Gamma(q, C, \lambda) \leq \sum_{h=0}^{\kappa}(q-h)C^{h+1}\binom{q}{h};\ \kappa = \min\left\{\left\lfloor \frac{1}{K\lambda}\right\rfloor - 1, q\right\}$$

Proof To make the bound problem-independent, let us upper bound κ and make it independent of T^*. We know that:

$$\begin{aligned} &\mathcal{H}_\lambda(T^*) = -\lambda\mathcal{S}(T^*)+\mathcal{H}(T^*) \geq \mathcal{H}_\lambda(\Omega) = \mathcal{H}(\Omega) = \mathbb{P}[Y = k^*(\Omega)] \geq \frac{1}{K} \\ &\implies \mathcal{S}(T^*)-1+\frac{1-\mathcal{H}(T^*)}{\lambda} \leq \frac{K-1}{K\lambda}-1 \end{aligned}$$

Which concludes the proof. ■

Theorem 12. *Let $\Gamma(q, \lambda, C)$ be defined as in Theorem 10, then we have:*

$$\Gamma(q, C, \lambda) \leq \sum_{h=0}^{\kappa_-} (q-h) C^{h+1} \binom{q}{h} + \sum_{h=\kappa_-+1}^{\kappa} (q-h)(h-\kappa_-) C^h \binom{q}{h}$$

$$\kappa_- = \min\left\{\mathcal{S}(T^*) - C + \left\lfloor \frac{1-\mathcal{H}(T^*)}{\lambda} \right\rfloor, q\right\}$$

$$\kappa = \min\left\{\mathcal{S}(T^*) - 1 + \left\lfloor \frac{1-\mathcal{H}(T^*)}{\lambda} \right\rfloor, q\right\}$$

Proof Let $\tilde{\pi}$ be the Selection policy, i.e. for any unit-state l:

$$\tilde{\pi}(l) = \begin{cases} \overline{a} \text{ If } l \text{ has never been visited} \\ \text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a) \text{ Otherwise.} \end{cases}$$

As we saw in the proof of Lemma 16, a branch l can only be considered for Expansion if $l \in T^{\tilde{\pi}}$. Let us denote $T^{\tilde{\pi}} = \{l_1, \ldots, l_{T^{\tilde{\pi}}}\}$ in the order that these branches are considered for Expansion. From Lemma 16 and the proof of Theorem 10, l_1 can only be considered for Expansion if:

$$\mathcal{S}\left(T^{\tilde{\pi}}\right) \leq \mathcal{S}(T^*) - 1 + \frac{1-\mathcal{H}(T^*)}{\lambda}$$

However, since l_2 can only be considered for Expansion after l_1, the condition becomes tighter for l_2:

$$\mathcal{S}\left(T^{\tilde{\pi}}\right) \leq \mathcal{S}(T^*) - 2 + \frac{1-\mathcal{H}(T^*)}{\lambda}$$

And so on, $l_{|T^{\tilde{\pi}}|}$ can only be expanded if:

$$\mathcal{S}\left(T^{\tilde{\pi}}\right) \leq \mathcal{S}(T^*) - |T^{\tilde{\pi}}| + \frac{1-\mathcal{H}(T^*)}{\lambda}$$

In the proof of Theorem 10, we counted all branches of depth $\mathcal{S}(l) = \mathcal{S}(T^*) - 1 + \frac{1-\mathcal{H}(T^*)}{\lambda}$ as potential candidates for Expansion, however, here we will show that this number is much lower. Indeed, consider a branch l with:

$$\mathcal{S}(l) = \mathcal{S}(T^*) - 1 + \frac{1-\mathcal{H}(T^*)}{\lambda}$$

For l to be considered for Expansion, there has to be a DT T satisfying:

$$\mathcal{S}(T) \leq \mathcal{S}(T^*) - 1 + \frac{1-\mathcal{H}(T^*)}{\lambda}$$

Since $l \in T$ then $\mathcal{S}(l) \leq \mathcal{S}(T)$, and thus:

$$\mathcal{S}(T) = \mathcal{S}(T^*) - 1 + \frac{1-\mathcal{H}(T^*)}{\lambda}$$

Since:

$$\mathcal{S}(T) > \mathcal{S}(T^*) - 2 + \frac{1-\mathcal{H}(T^*)}{\lambda}$$

There can only be one branch that is expanded in T. Therefore, if l is expanded, non of its sibling branches (i.e. branches that differ from l by only one clause) can be expanded. This reduces the number of such branches l that can be considered for Expansion to one branch from each siblings set. Let us now count these branches of depth $h = \mathcal{S}(T^*) - 1 + \frac{1-\mathcal{H}(T^*)}{\lambda}$ fulfilling these conditions:

- There are $\binom{q}{h}$ choices for the features.
- For the first $h-1$ chosen features, there are C choices of values, but the last one can only take one value, otherwise a sibling branch will also be counted.
- Each Expansion incurs $q-h$ branch evaluations.

In fact, these new considerations are valid for all depths $h > \mathcal{S}(T^*) - C + \frac{1-\mathcal{H}(T^*)}{\lambda}$, let us generalise:

- There are $\binom{q}{h}$ choices for the features.
- The first $h-1$ features can take C values, the last one can only take $k = h - \mathcal{S}(T^*) + C - \left\lfloor \mathcal{S}(T^*) - 1 + \frac{1-\mathcal{H}(T^*)}{\lambda} \right\rfloor$ values.
- Each Expansion incurs $q-h$ branches

As for branches with depths $h \leq \mathcal{S}(T^*) - C + \frac{1-\mathcal{H}(T^*)}{\lambda}$, the count stays the same as in Theorem 10. Thus we deduce that:

$$\Gamma(q, C, \lambda) \leq \sum_{h=0}^{\kappa_-} (q-h) C^{h+1} \binom{q}{h} + \sum_{h=\kappa_-+1}^{\kappa} (q-h)(h-\kappa_-) C^h \binom{q}{h}$$

$$\kappa_- = \min\left\{ \mathcal{S}(T^*) - C + \left\lfloor \frac{1-\mathcal{H}(T^*)}{\lambda} \right\rfloor, q \right\}$$

$$\kappa = \min\left\{ \mathcal{S}(T^*) - 1 + \left\lfloor \frac{1-\mathcal{H}(T^*)}{\lambda} \right\rfloor, q \right\}$$

■

Corollary 13. *Let $\Gamma(q, \lambda, C)$ be defined as in Theorem 10, then we have:*

$$\Gamma(q, C, \lambda) \leq \sum_{h=0}^{\kappa_-} (q-h) C^{h+1} \binom{q}{h} + \sum_{h=\kappa_-+1}^{\kappa} (q-h) \left(h - \left\lfloor \frac{1}{K\lambda} \right\rfloor + C \right) C^h \binom{q}{h}$$

$$\kappa_- = \min\left\{ \left\lfloor \frac{1}{K\lambda} \right\rfloor - C, q \right\};\ \kappa = \min\left\{ \left\lfloor \frac{1}{K\lambda} \right\rfloor - 1, q \right\}$$

Proof The Corollary follows immediately in similar fashion to Corollary 11 by using:

$$\mathcal{S}(T^*) - 1 + \frac{1-\mathcal{H}(T^*)}{\lambda} \leq \frac{K-1}{K\lambda} - 1$$

■

Theorem 14. *Consider a classification problem where all features share the same number of categories C, i.e. $\mathcal{X} = \{1, \dots, C\}^q$. Performing a binary encoding on $\mathcal{X}$ yields the new feature space $\mathcal{X}' = \{0,1\}^{qC}$. We define an unnecessary branch l on $\mathcal{X}'$ as a branch that valuates to 0 for any input vector $X \in \mathcal{X}$:*

$$\forall X \in \mathcal{X} : l(X) = 0$$

The number of unnecessary branches introduced by binary encoding is:

$$\mathcal{U}(q, C) = \mathcal{A}(q, C) - \mathcal{B}(q, C) = 3^{qC} - [2C+1]^q$$

Proof The proof of this Theorem proceeds by counting the total number of branches possible on $\mathcal{X}'$ and subtracting the total number of branches that are not unnecessary.

Let us start with the total number of branches on $\mathcal{X}'$. Any branch on $\mathcal{X}'$ has the form:

$$l = \bigwedge_{v=1}^{w} \mathbb{1}\{X'^{(i_v)} = z_v\}$$

Where $X'^{(i_v)}$ are the features on the space $\mathcal{X}'$, $w \in \{0, \dots, qC\}, i_v \in \{1, \dots, qC\}, z_v \in \{0,1\}$. We note that $w = 0$ corresponds to $l = \Omega$ by definition.

- There are qC possibilities for choosing w.
- For each possible value w, there are $\binom{qC}{w}$ possible combinations $\{i_1, \dots, i_w\}$.
- For each combination $\{i_1, \dots, i_w\}$, there are 2^w possible assignments $(z_1, \dots, z_w)$

Therefore the total number of branches on $\mathcal{X}'$ is:

$$\mathcal{A}(q, C) = \sum_{w=0}^{qC} \binom{qC}{w} 2^w = 3^{qC} \tag{4.9}$$

Let us now count the number of non-unnecessary branches. To do this, we consider a slightly different notation of the features on $\mathcal{X}'$.

$$\forall i \in \{1, \dots, q\}, \forall j \in \{1, \dots, C\} : X'^{(i,j)} = \mathbb{1}\{X^{(i)} = j\}$$

A branch $l = \bigwedge_{v=1}^{w} \mathbb{1}\{X'^{(i_v, j_v)} = z_v\}$ is not unnecessary if and only if $w \in \{0, \dots, q\}, i_v \in \{1, \dots, q\}, j_v \in \{1, \dots, C\}, z_v \in \{0,1\}$.

- For each possible value $w \in \{0, \dots, q\}$, there are $\binom{q}{w}$ possible combinations $\{i_1, \dots, i_w\}$.
- For each combination $\{i_1, \dots, i_w\}$, there are C^w possible assignments $(j_1, \dots, j_w)$.
- For each assignment $(j_1, \dots, j_w)$, there are 2^w possible assignments $(z_1, \dots, z_w)$.

The total number of branches that are not unnecessary is therefore:

$$\mathcal{B}(q, C) = \sum_{w=0}^{q} \binom{q}{w} 2^w C^w = [2C+1]^q \tag{4.10}$$

From Eq. (4.9) and Eq. (4.10) we deduce that the total number of unnecessary branches is:

$$\mathcal{U}(q, C) = \mathcal{A}(q, C) - \mathcal{B}(q, C) = 3^{qC} - [2C+1]^q$$

■

4.12 Appendix: Auxiliary Procedures

procedure SELECT()
  $l \leftarrow \Omega$
  path $\leftarrow [l]$
  while l.expanded and (not l.complete) **do**
    $(\mathcal{Q}(l,i), \text{return\_complete}, i, \text{children\_incomplete}) \leftarrow l.\text{queue}[0]$
    $l \leftarrow \text{children\_incomplete}[0]$
    path.append(l)
  end while
  return (l, path)
end procedure
procedure EXPAND($l, \mathcal{D}$)
  l.expanded $\leftarrow True$
  for $i \in \mathcal{A}(l) \setminus \{\overline{a}\}$ **do**
    SPLIT($l, i, \mathcal{D}$)
    $\mathcal{V}(l) \leftarrow \max\{\mathcal{Q}(l,\overline{a}), l.\text{queue}[0][0]\}$ ▷ This update comes from Eq. (4.3)
  end for
  if $\mathcal{V}(l) = \mathcal{Q}(l,\overline{a})$ **then** ▷ In this case $\mathcal{V}^*(l) = \mathcal{Q}^*(l,\overline{a}) = \mathcal{H}(l)$
    l.complete $\leftarrow True$ ▷ $\mathcal{V}^*(l)$ is known
    l.terminal $\leftarrow True$ ▷ Label l terminal if the optimal action at l is $\pi^*(l) = \overline{a}$
  end if
end procedure
procedure BACKPROPAGATE(path)
  $N \leftarrow \text{length}(\text{path})$
  for $t = N-2$ to 0 **do**
    $l \leftarrow \text{path}[t]$
    $(\mathcal{Q}(l,i), \text{return\_complete}, i, \text{children\_incomplete}) \leftarrow l.\text{queue.pop}()$
    $\mathcal{Q}(l,i) \leftarrow \text{return\_complete}$ ▷ Initialise $\mathcal{Q}(l,i)$
    for $l' \in \text{children\_incomplete}$ **do**
      $\mathcal{Q}(l,i) \leftarrow \mathcal{Q}(l,i) + \mathcal{V}(l')$
      if l'.complete **then** ▷ Check if l' is complete now
        children_incomplete.discard(l') ▷ Delete l' from children_incomplete
      end if
    end for
    l.queue.push$((\mathcal{Q}(l,i), \text{return_complete}, i, \text{children_incomplete}))$
    $(\mathcal{Q}(l,i^*), \text{return\_complete}, i^*, \text{children\_incomplete}) \leftarrow l.\text{queue}[0][0]$
    $\mathcal{V}(l) \leftarrow \mathcal{Q}(l,i^*)$
    if $(\mathcal{V}(l) = \mathcal{Q}(l,\overline{a}))$ or (children_incomplete is empty) **then**
      l.complete $\leftarrow True$
      l.terminal $\leftarrow True$ ▷ Label l terminal if the optimal action at l is $\pi^*(l) = \overline{a}$
    end if
  end for
end procedure

```
56: procedure INFER()
57:     T ← []
58:     Q ← queue()
59:     Q.put(Ω)
60:     while Q is not empty do
61:         l ← Q.pop()
62:         if l.terminal then
63:             T.append(l)
64:         else
65:             (Q(l, i), return_complete, i, children_incomplete) ← l.queue[0]
66:             for l' ∈ l.children[i] do
67:                 Q.put(l')
68:             end for
69:         end if
70:     end while
71:     return T
72: end procedure
73: procedure INITIALISE(l, D)
74:     l.expanded ← False                          ▷ Label l as not expanded yet
75:     l.children ← dict()                         ▷ Initialise the dictionary of children
76:     l.queue ← queue([])                         ▷ Initialise the priority queue of l
77:     Q(l, ā) ← H(l)                              ▷ H(l) is calculated with D
78:     if A(l) = {ā} then
79:         l.terminal ← True                       ▷ Label l as terminal if it cannot be split
80:         l.complete ← True                       ▷ V*(l) is known
81:         V(l) ← Q(l, ā)                          ▷ In this case V*(l) = Q*(l, ā) = H(l)
82:     else
83:         l.terminal ← False
84:         Initialise V(l) according to Eq. (4.3) and Eq. (4.4)
85:         if V(l) = Q(l, ā) then
86:             l.complete ← True                   ▷ V*(l) is known, V*(l) = Q*(l, ā) = H(l)
87:             l.terminal ← True                   ▷ Label l terminal if the optimal action at l is π*(l) = ā
88:         else
89:             l.complete ← False                  ▷ V*(l) is still unknown
90:         end if
91:     end if
92:     memo.add(l)                                 ▷ Add the initialised branch to the memo
93: end procedure
```

94: **procedure** SPLIT$(l, i, \mathcal{D})$
95: l.children$[i] \leftarrow []$ ▷ Initialise the list of children that stem taking split action i in l
96: $\mathcal{Q}(l, i) \leftarrow -\lambda$ ▷ Initialise the Upper Bound $\mathcal{Q}(l, i)$
97: return_complete $\leftarrow -\lambda$ ▷ Initialise the return due to complete children
98: children_incomplete $\leftarrow []$ ▷ Initialise the list of incomplete children
99: **for** $j \in \{1, \ldots, C_i\}$ **do**
100: $l_{ij} \leftarrow l \wedge \mathbb{1}\{X^{(i)} = j\}$
101: **if** $l_{ij} \notin$ memo **then** ▷ Only initialise the branches that are not in the memo
102: INITIALISE$(l_{ij}, \mathcal{D})$
103: **end if**
104: l.children$[i]$.append(l_{ij})
105: $\mathcal{Q}(l, i) \leftarrow \mathcal{Q}(l, i) + \mathcal{V}(l_{ij})$ ▷ Update the Upper Bound $\mathcal{Q}(l, i)$
106: **if** l_{ij}.complete **then**
107: return_complete $\leftarrow$ return_complete $+ \mathcal{V}(l_{ij})$
108: **else**
109: children_incomplete.append(l_{ij})
110: **end if**
111: **end for**
112: l.queue.push$((\mathcal{Q}(l, i), \text{return_complete}, i, \text{children_incomplete}))$
113: **end procedure**

4.13 Appendix: Experiments on depth analysis

This appendix provides all the results for all the experiments conducted in Section 4.7.3.

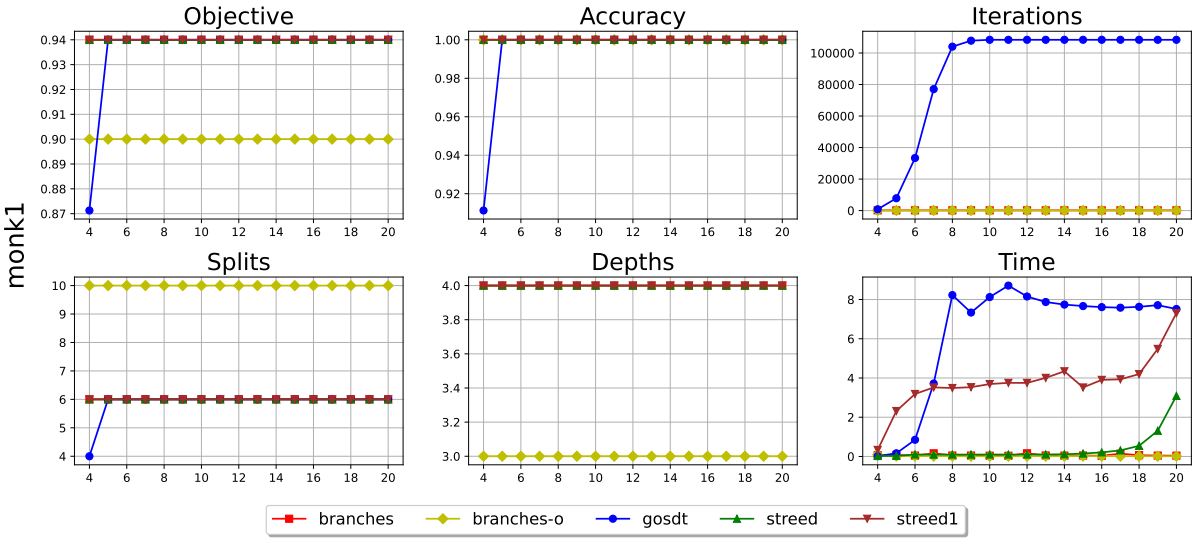


Figure 4.19: Depth analysis for monk1.

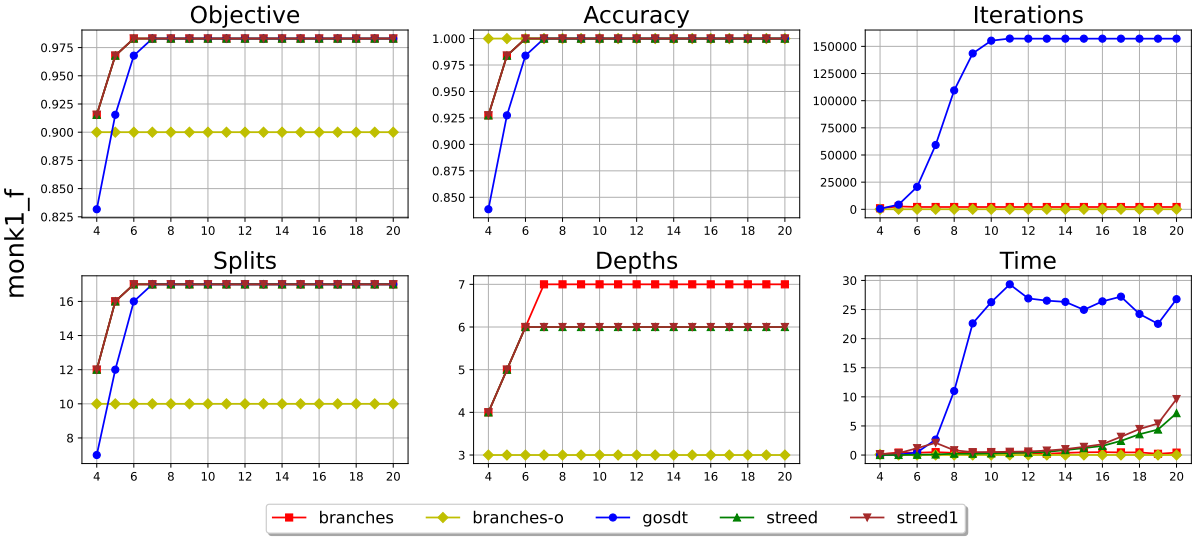


Figure 4.20: Depth analysis for monk1-f.

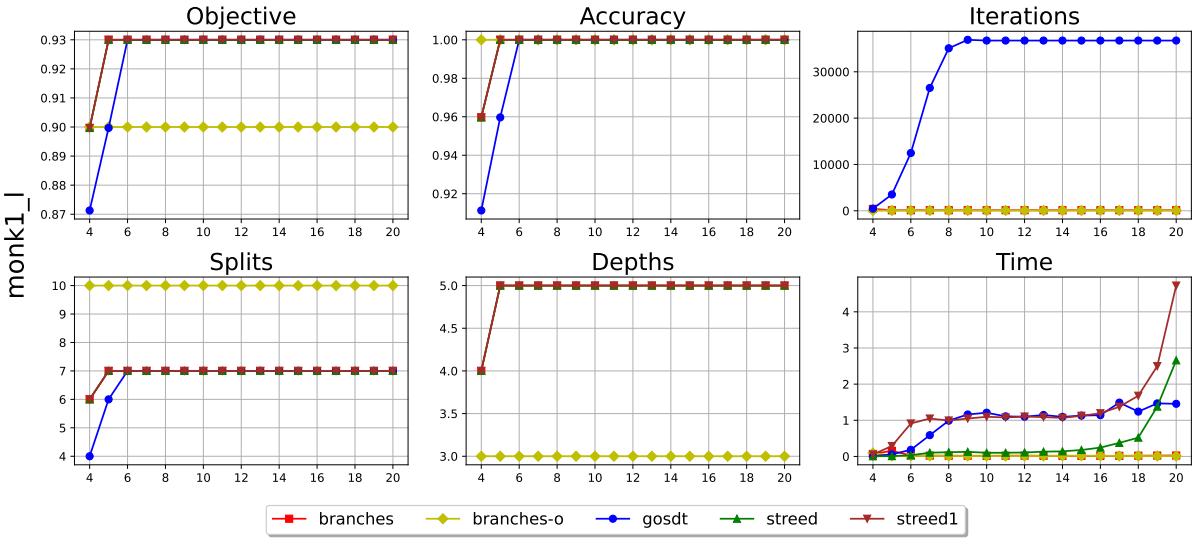


Figure 4.21: Depth analysis for monk1-l.

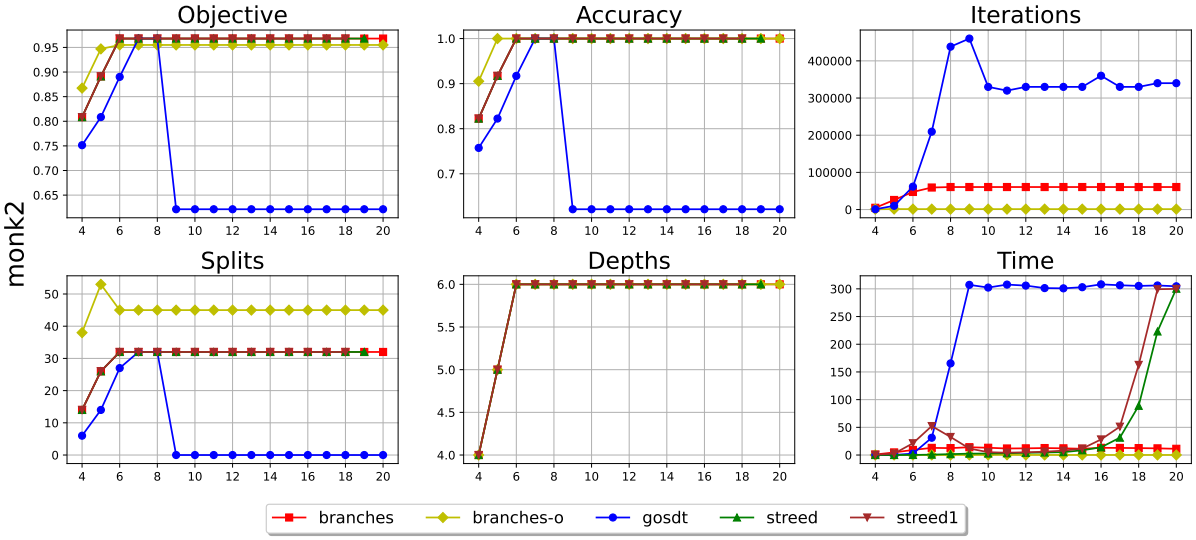


Figure 4.22: Depth analysis for monk2.

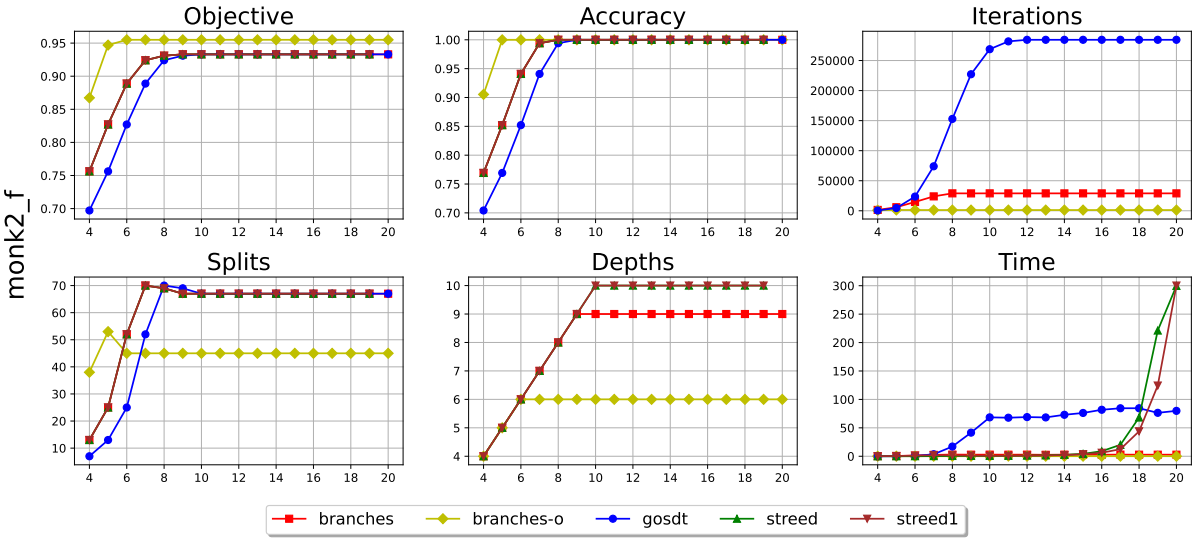


Figure 4.23: Depth analysis for monk2-f.

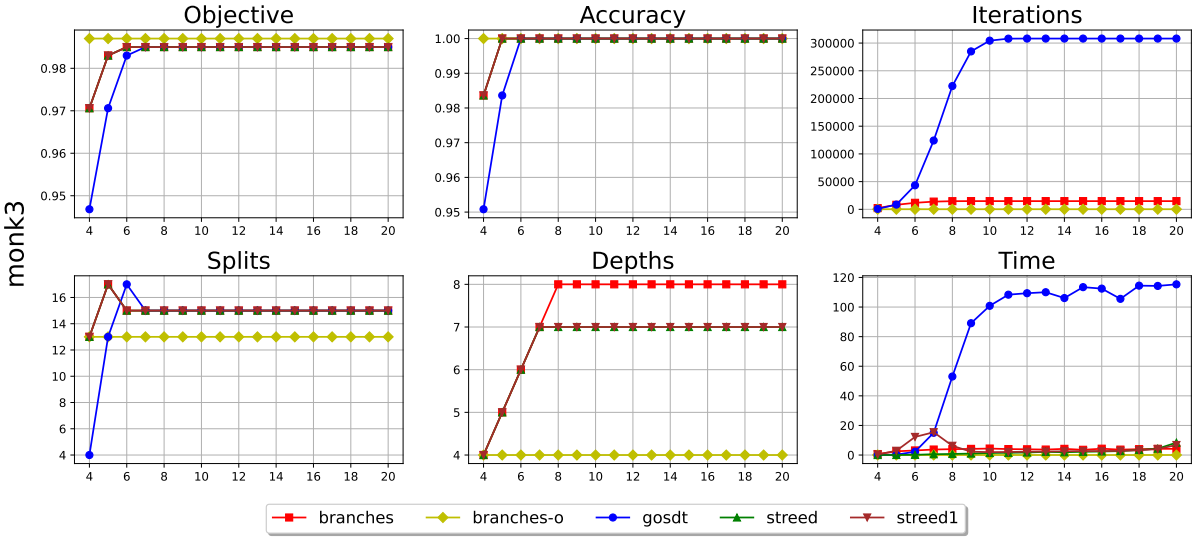


Figure 4.24: Depth analysis for monk3.

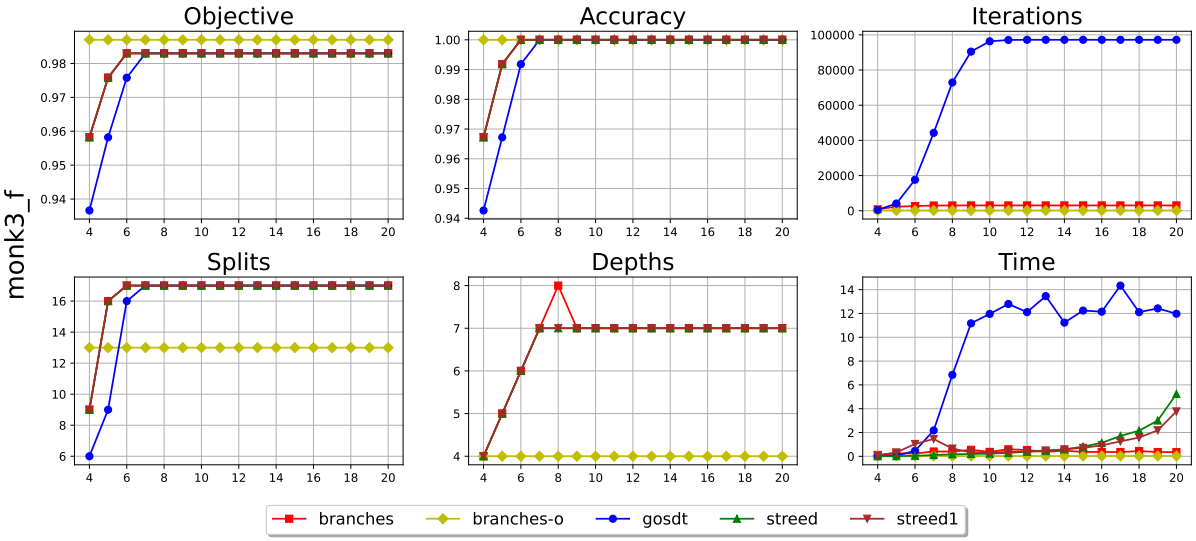


Figure 4.25: Depth analysis for monk3-f.

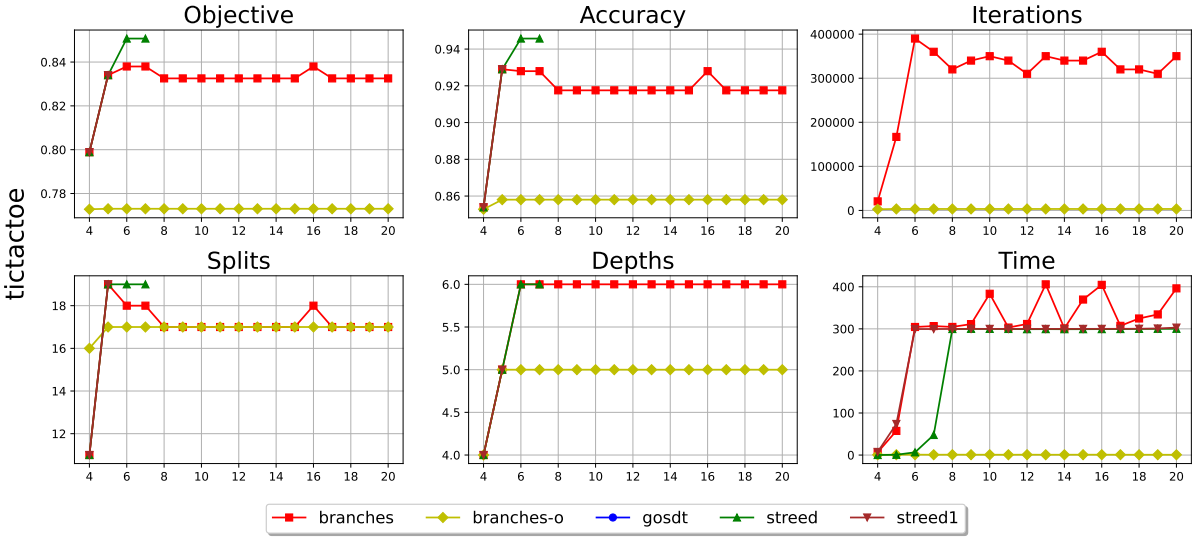


Figure 4.26: Depth analysis for tic-tac-toe.

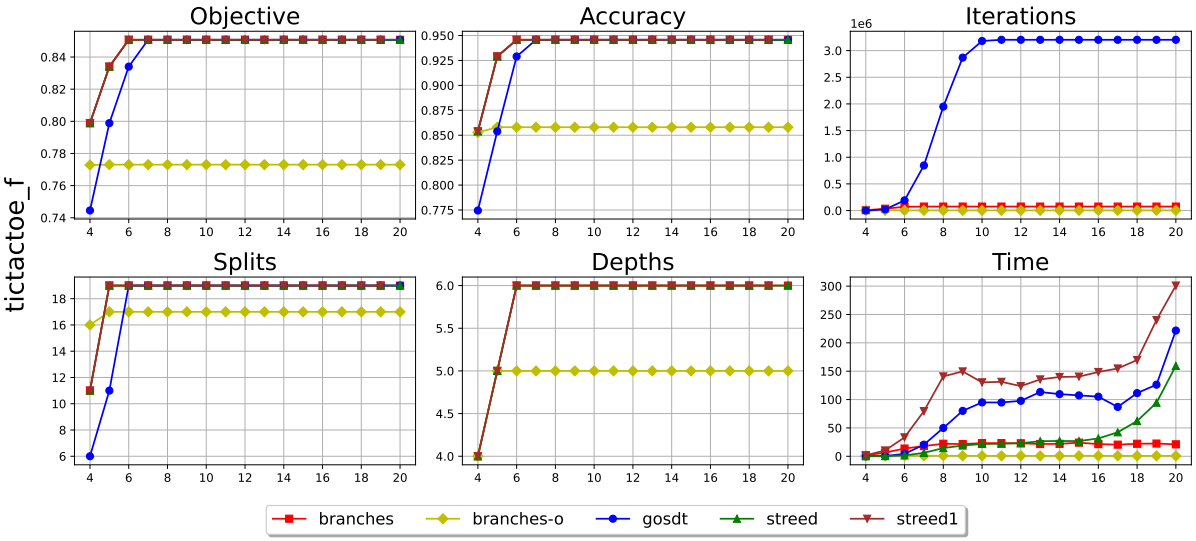


Figure 4.27: Depth analysis for tic-tac-toe-f.

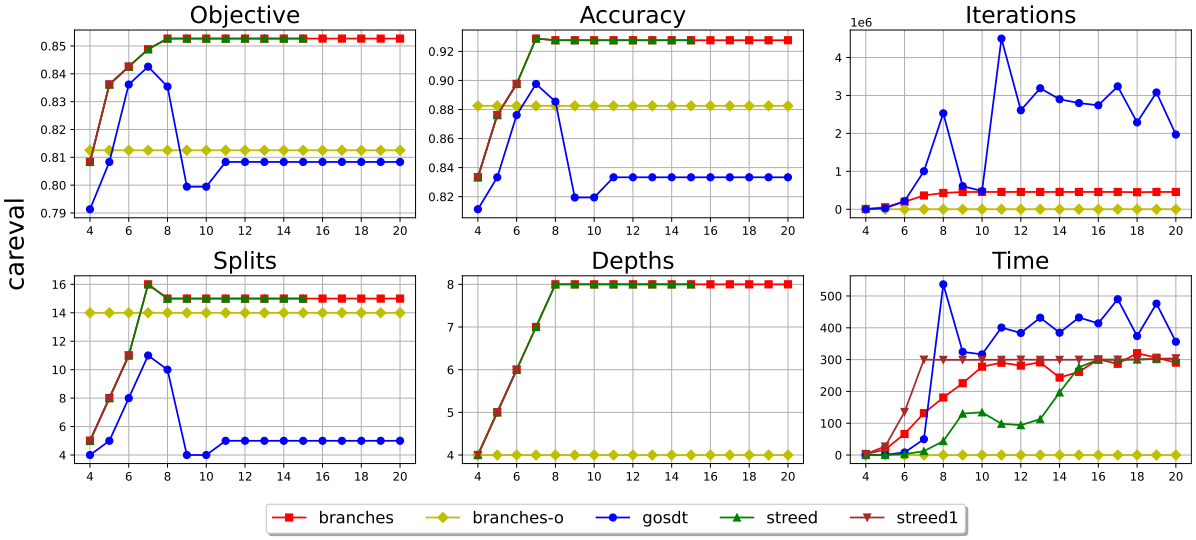


Figure 4.28: Depth analysis for car-eval.

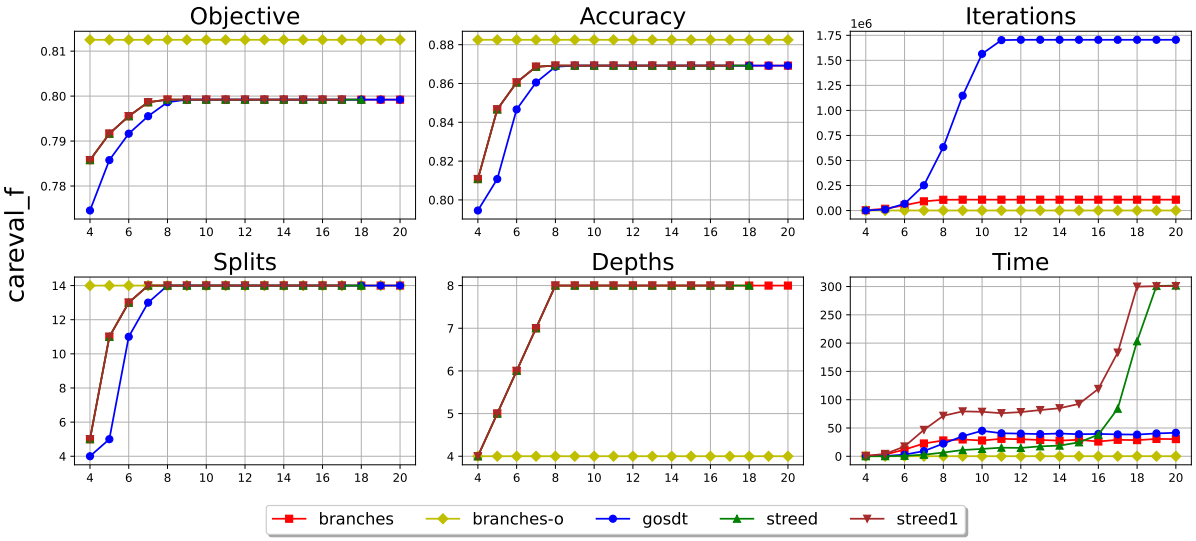


Figure 4.29: Depth analysis for car-eval-f.

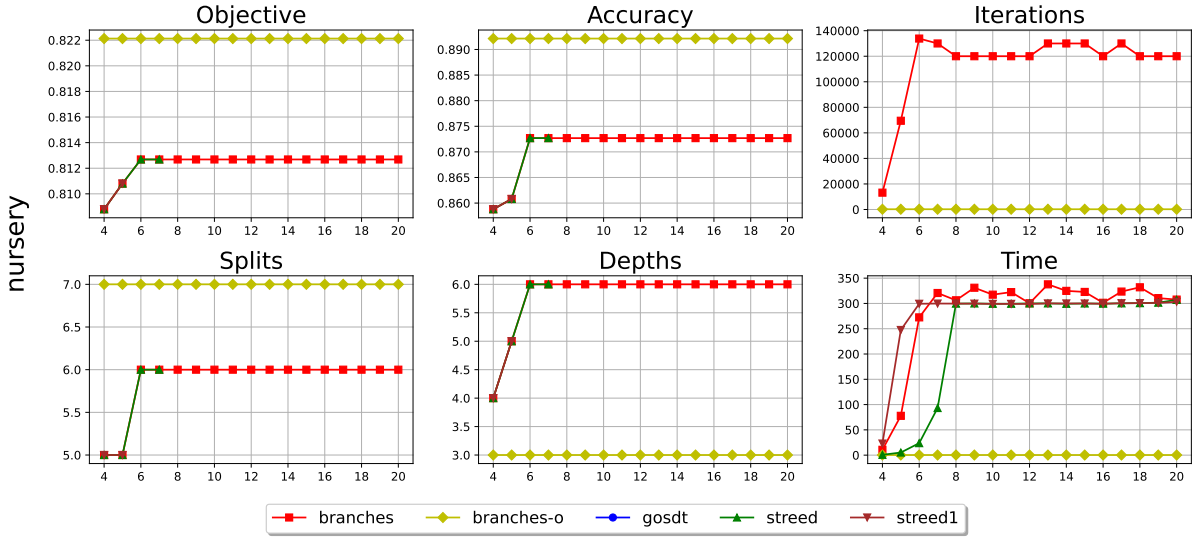


Figure 4.30: Depth analysis for nursery.

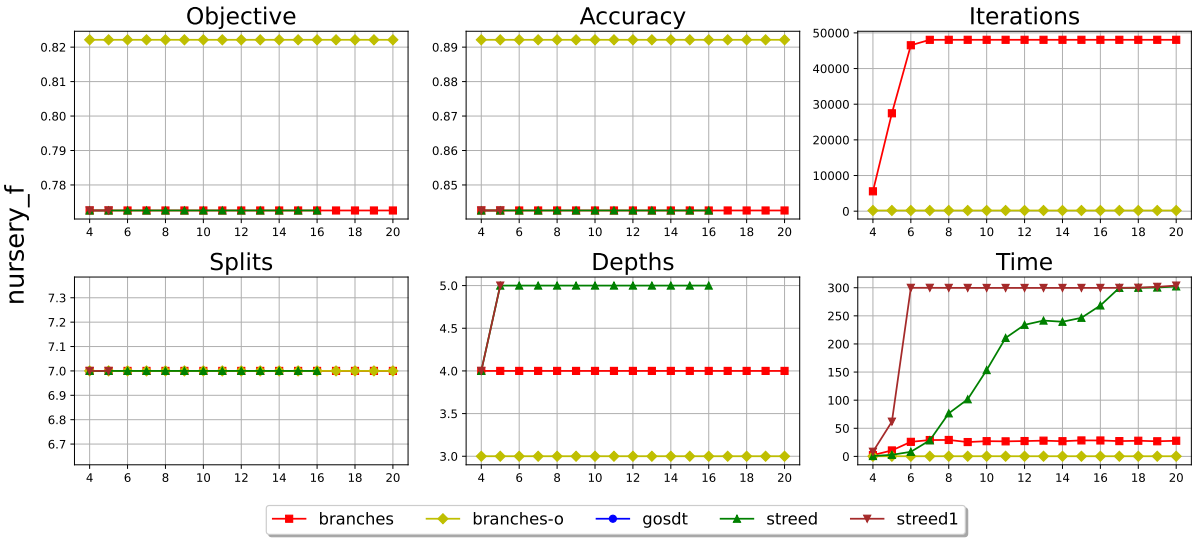


Figure 4.31: Depth analysis for nursery-f.

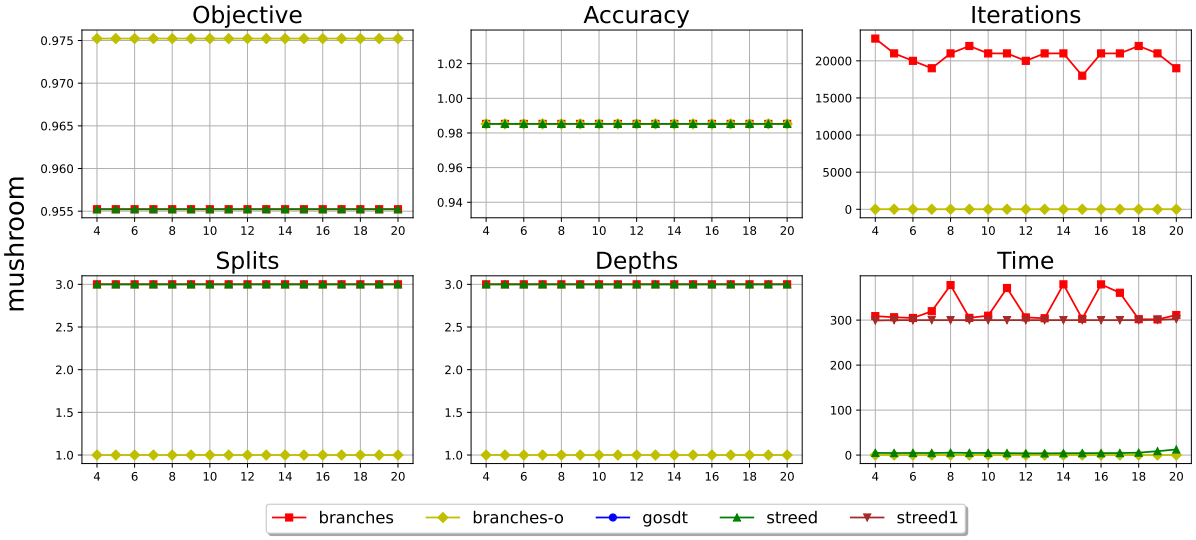


Figure 4.32: Depth analysis for mushroom.

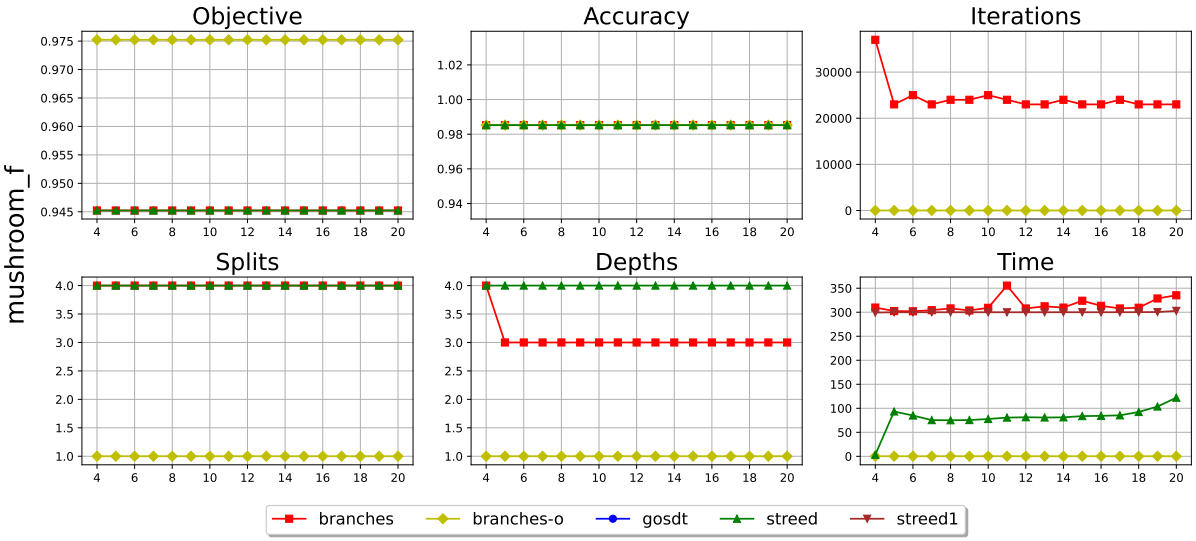


Figure 4.33: Depth analysis for mushroom-f.

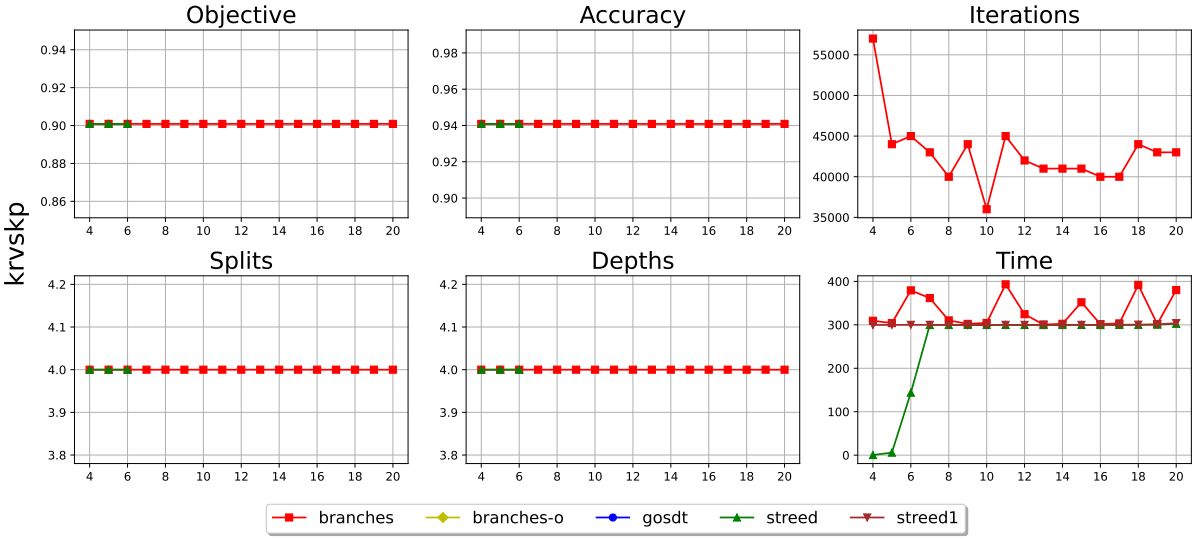


Figure 4.34: Depth analysis for kr-vs-kp.

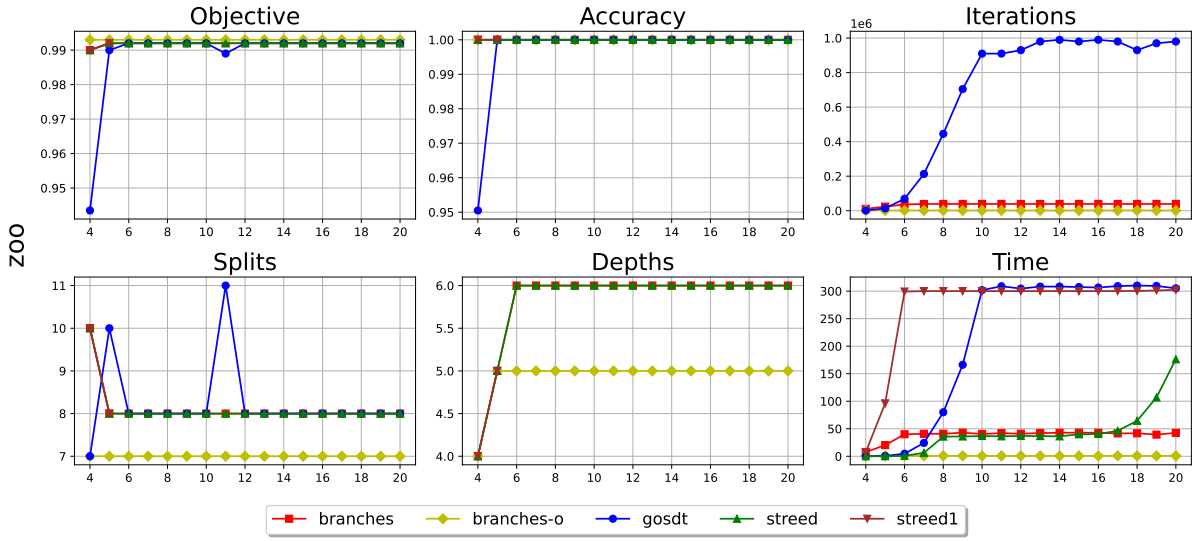


Figure 4.35: Depth analysis for zoo.

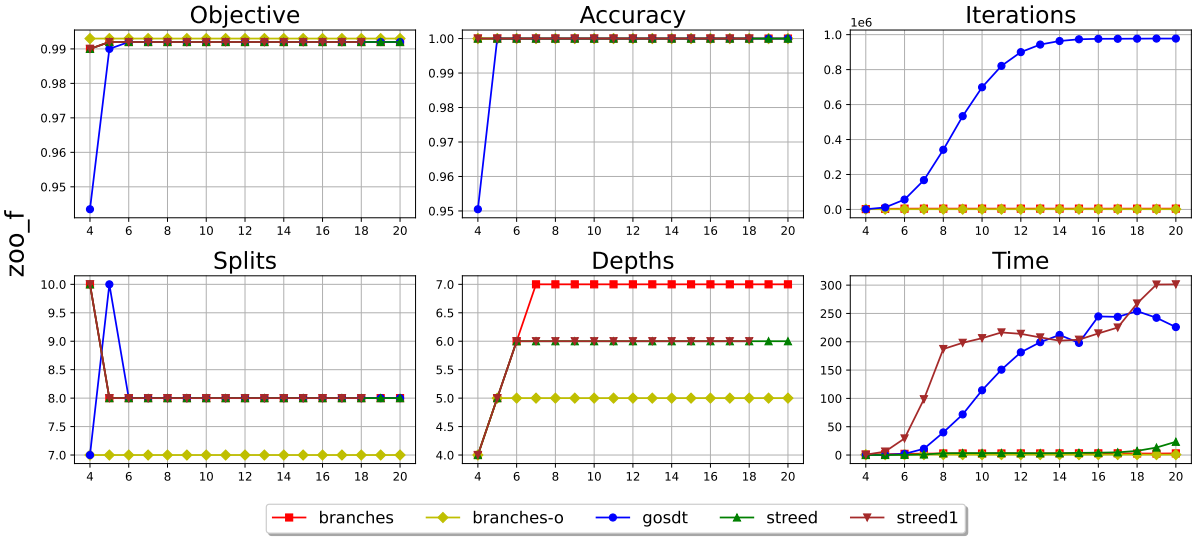


Figure 4.36: Depth analysis for zoo-f.

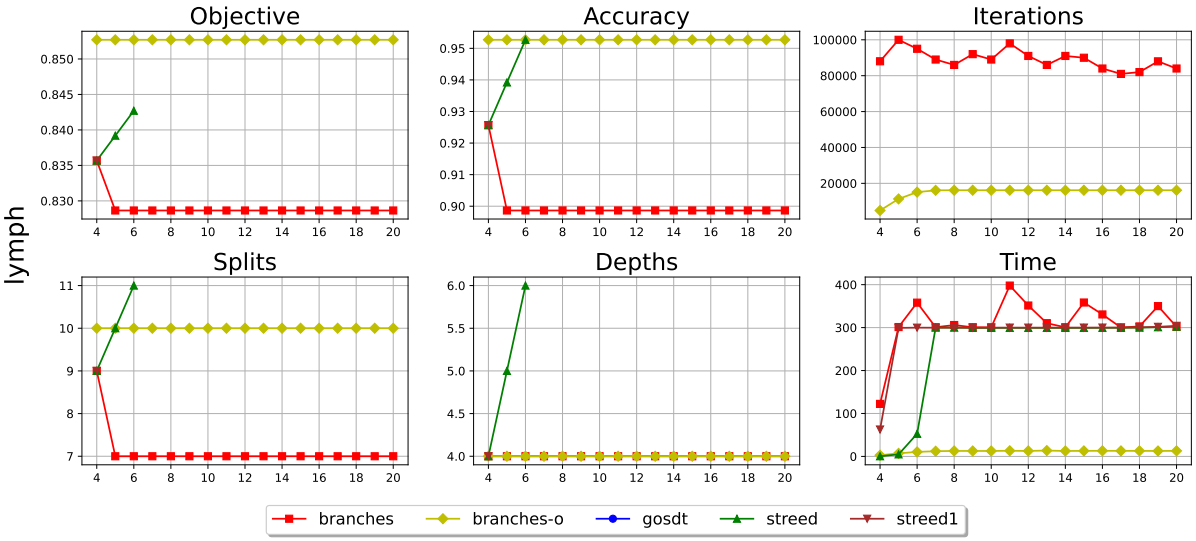


Figure 4.37: Depth analysis for lymph.

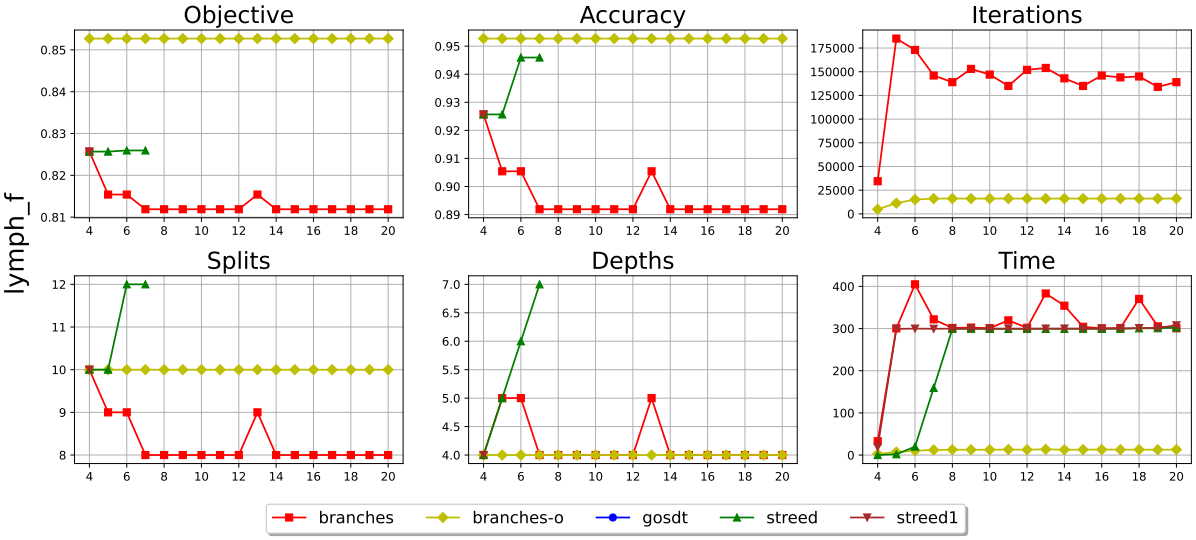


Figure 4.38: Depth analysis for lymph-f.

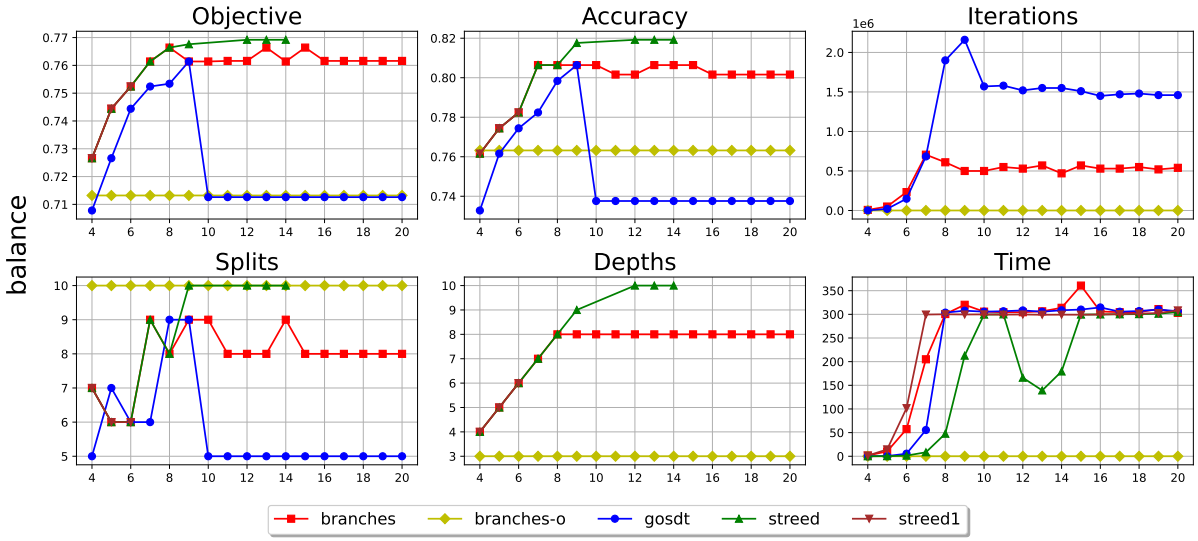


Figure 4.39: Depth analysis for balance.

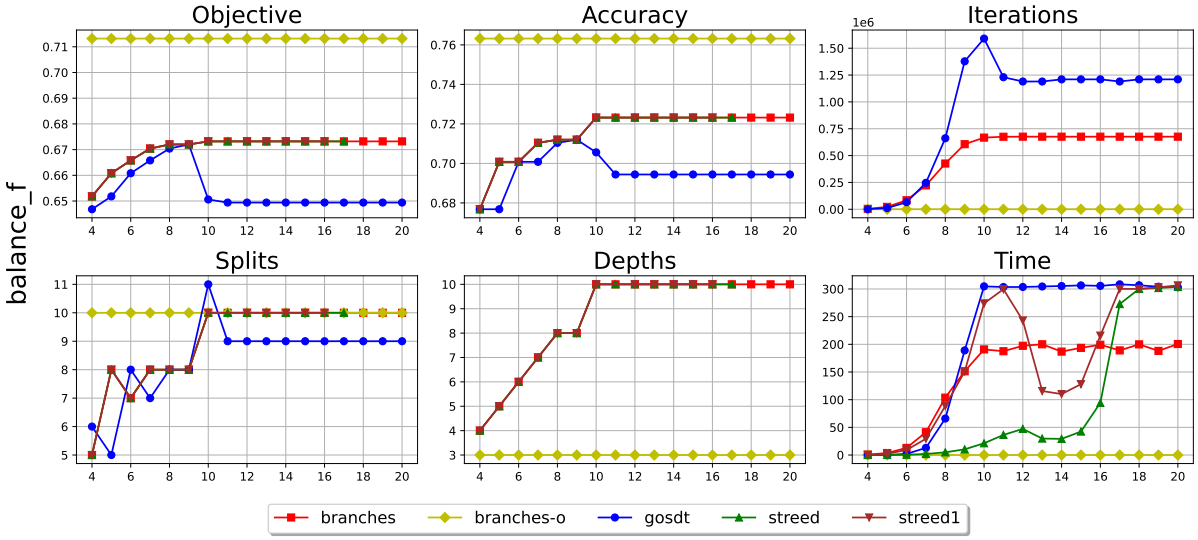


Figure 4.40: Depth analysis for balance-f.

Part II

Optimal Online Decision Trees

Chapter 5

Online Learning of Decision Trees with UCB and Epsilon-Greedy

Contents

5.1 Introduction . . . 129
5.2 Preliminaries . . . 130
5.3 The Algorithms: UCDT and EGDT . . . 134
5.4 Sample Efficiency and Weights Degeneracy . . . 137
5.5 Experiments . . . 140
5.6 Conclusion . . . 142
5.7 Appendix: Proofs . . . 146
5.8 Appendix: Pseudo-code . . . 151

5.1 Introduction

Part I of the thesis focused on the batch setting, where a fixed dataset is provided. In this second part, we turn our attention to the online setting where we rather observe a stream of data. This framework, also commonly known as Data Stream Mining, is becoming increasingly important in modern applications with continuous supplies of data. An in-depth discussion about the importance of Data-Stream Mining can be found in (Bifet and Kirkby, 2009).

All the benefits of Decision Trees, that we discussed for the batch setting, are still relevant for Online Learning. Thus, we naturally want to construct optimal sparse DTs based on data streams, but unfortunately the batch algorithms we discussed in Part I are not directly applicable to data streams. The greedy approaches such as CART and C4.5 decide which splits to perform based on impurity metrics they calculate on a fixed dataset. Likewise, the Mathematical Programming methods have rigid formulations of their mathematical programs with respect to a fixed dataset, and the Dynamic Programming (DP) and Branch & Bound (B&B) algorithms have caching and pruning procedures that highly depend on the provided dataset as well. This prompted a large body of research to investigate ways that are specifically tailored to constructing DTs from data streams. Nonetheless, most of these efforts focus on top-down greedy constructions in similar fashion to CART and C4.5, and as such, they also inherit their suboptimality issues.

The most popular algorithm of this sort is VFDT (Domingos and Hulten, 2000). It offers a straightforward way of generalising the top-down greedy construction of DTs to the online setting via statistical estimates of the impurity metrics. VFDT decides when to split each leaf and which feature to use for the split according to Hoeffding's inequality, hence the name Hoeffding Trees. This yielded a principled approach and laid the foundation for subsequent developments such as CVFDT (Hulten et al., 2001), HAT (Bifet and Gavalda, 2009) and EFDT (Manapragada et al., 2018). Nevertheless, Rutkowski et al. (2012) noticed an inadequacy in applying Hoeffding's inequality for the considered impurity metrics Entropy and Gini. Indeed, these metrics are not sums of independent random variables, and as such, they violate the assumptions of Hoeffding's inequality. Rutkowski et al. (2012) address this issue by employing McDiarmid's inequality instead, however, despite the theoretical soundness of these bounds, they are very conservative, i.e. they need to gather a lot of data before proceeding with a split, thus significantly slowing down the algorithm. Other statistical tests were also proposed in the literature, among which the Multivariate Delta Method (Jin and Agrawal, 2003) and the Gaussian approximation (Rutkowski et al., 2013), which is used in conjunction with a first order Taylor Expansion of the impurity metrics.

None of the methods cited in the previous paragraph address optimality (in terms of solving for sparsity), and it seems that this concern is rarely discussed in the online setting. It is certainly far less investigated than in the batch setting. The only work we are aware of that follows a different approach from the top-down construction in this context is (Garlapati et al., 2015). The authors frame the problem within a Reinforcement Learning framework and use Q-Learning (Watkins and Dayan, 1992) to search for the optimal policy, inducing the RLDT algorithm. However, Garlapati et al. (2015) do not provide theoretical guarantees for the optimality of RLDT and no implementation is supplied to the best of our knowledge. On this note, in the first year of this PhD, we explored improving the scalability of RLDT by using DQN (Mnih et al., 2013) and we proved some theoretical guarantees in the form of optimising the expected misclassification error, this work is presented in our paper (Chaouki et al., 2023).

Alas, the results were not satisfactory and the framework seems hard to analyse meaningfully from a theoretical standpoint as it is a general Reinforcement Learning approach that does not incorporate special properties of the DT Learning problem.

Following the approach we introduced in Part I, we formulate the problem of online DT optimisation within a RL framework as well, but this time given a data stream instead of a fixed dataset. The induced algorithms, Upper Confidence Bound Decision Trees (UCDT) and Epsilon-Greedy Decision Trees (EGDT) use the Monte Carlo Tree Search (MCTS) approach (Browne et al., 2012). The difference between the two algorithms pertains to the policies they use, UCB for UCDT and Epsilon-Greedy for EGDT. These policies are chosen for their ability to manage the Exploration-Exploitation dilemma efficiently, see (Auer et al., 2002) for more details and a regret analysis in the Multi-Armed Bandit setting (Lattimore and Szepesvári, 2020).

We prove that both UCDT and EGDT converge almost surely to the optimal policy as we observe more and more data from the stream. We validate our findings via a series of experiments comparing UCDT and EGDT with the state of the art implemented in River (Montiel et al., 2021), a recent popular Python library for online Machine Learning. The most important result in these findings is that UCDT and EGDT successfully find the optimal sparse DT in some settings where the state of the art are practically guaranteed to fail such as XOR.

5.2 Preliminaries

We consider the classification problem introduced in Section 2.1 within the online setting, and we keep the concepts of branches and DTs introduced in Section 3.2. We recall that, within this online setting, we observe data X_t, Y_t through a stream and we assume that they are i.i.d. drawn from a joint probability distribution. A main limitation of our work is that we do not consider concept drift where the underlying distribution is non-stationary. We focus on proofs of optimality for the i.i.d. assumption and we leave concept drift for future work.

5.2.1 Problem Reformulation

Due to the online assumption, there are few modifications of the problem formulation that we need to account for. Random variables X and Y no longer follow some joint empirical probability distribution induced by a dataset $\mathcal{D}$, but rather $\mathbb{P}_{X,Y}$ (introduced in Section 2.1). For any branch l, the majority class is:

$$k^*(l) = \text{Argmax}_{1 \leq k \leq K} \mathbb{P}[Y = k | l(X) = 1]$$

For mathematical soundness, we only define $k^*(l)$ for branches l satisfying $\mathbb{P}[l(X) = 1] > 0$[1], otherwise it is not necessary to define $k^*(l)$ because the leaf contains no data sample. Therefore Eq. (3.3) is now relegated to the status of estimator rather than the true definition of $k^*(l)$. The same remark holds for $\mathcal{H}(l)$, we write it as:

$$\mathcal{H}(l) = \mathbb{P}[l(X) = 1, Y = k^*(l)]$$

But we can no longer use Eq. (3.4) as the true probability value, but only as an estimate. For any DT T, the prediction $T(X)$, the accuracy $\mathcal{H}(T)$ and the regularised objective $\mathcal{H}_\lambda(T)$ are

[1] We denote $\mathbb{P}$ both the conditional probability distribution of Y given X and the marginal probability distribution of X, they are both defined uniquely from $\mathbb{P}_{X,Y}$. In fact, we will also drop the subscript X, Y even when considering the joint probability distribution to make the notation lighter.

still defined with Eq. (3.5), Eq. (3.6) and Eq. (3.7) respectively, which we recall below:

$$\forall T = \{l_1, \ldots, l_{|T|}\} : T(X) = \sum_{u=1}^{|T|} l_u(X) k^*(l_u) \in \{1, \ldots, K\}$$

$$\mathcal{H}(T) = \mathbb{P}[T(X) = Y] = \sum_{u=1}^{|T|} \mathcal{H}(l_u)$$

$$\mathcal{H}_\lambda(T) = -\lambda \mathcal{S}(T) + \mathcal{H}(T)$$

Our objective remains to find $T^* = \text{Argmax}_T \mathcal{H}_\lambda(T)$.

5.2.2 Markov Decision Process (MDP)

We use the same MDP as DT-Miner's (see Section 3.2.3). However, while the reward of the terminal action is still $r(T, \overline{a}) = \mathcal{H}(T)$, we can no longer calculate it exactly as we used to do with Eq. (3.6) and Eq. (3.4). That is because the probability measure in $\mathcal{H}(T) = \mathbb{P}[T(X) = Y]$ is the unknown joint probability distribution $\mathbb{P}_{X,Y}$ rather than the empirical probability distribution induced by a dataset, as was the case in the batch context. As a consequence, $r(T, \overline{a})$ is unknown and needs to be estimated. A first straightforward approach to implementing this idea is by observing some fixed number n of samples $\{(X_t, Y_t)\}_{t=1}^n$ from the stream and defining the estimate accordingly. We recall that:

$$r(T, \overline{a}) = \mathcal{H}(T) = \mathbb{P}[T(X) = Y] = \sum_{l \in T} \mathcal{H}(l) = \sum_{l \in T} \mathbb{P}[l(X) = 1, Y = k^*(l)]$$

Thus, to estimate $\mathcal{H}(T)$ it suffices to estimate $\mathbb{P}[l(X) = 1, Y = k^*(l)]$ for all $l \in T$. Let $l \in T$, from the definition of $k^*(l)$ we have:

$$\mathbb{P}[l(X) = 1, Y = k^*(l)] = \max_{1 \leq k \leq K} \mathbb{P}[l(X) = 1, Y = k]$$

For each $1 \leq k \leq K$, we estimate $\mathbb{P}[l(X) = 1, Y = k] = \mathbb{E}[l(X) \mathbb{1}\{Y = k\}]$ with the following Monte Carlo estimate:

$$\frac{1}{n} n_k(l) = \frac{1}{n} \sum_{t=1}^{m} l(X_t) \mathbb{1}\{Y_t = k\}$$

Which is just the average of the number of samples in $\{(X_t, Y_t)\}_{t=1}^n$ that are contained in branch l and that are of class k. This leads to estimating $\mathcal{H}(l) = \mathbb{P}[l(X) = 1, Y = k^*(l)]$ with:

$$\widehat{\mathcal{H}}(l) = \frac{1}{n} \max_{1 \leq k \leq K} n_k(l)$$

Now we can define the following estimate $\widehat{\mathcal{H}}(T)$ for $\mathcal{H}(T)$:

$$\widehat{\mathcal{H}}(T) = \sum_{l \in T} \widehat{\mathcal{H}}(l)$$

Unfortunately, this estimate presents a subtle issue due to its bias as we will see shortly. We recall that our objective is to find $T^* = \text{Argmax}_T \mathcal{H}_\lambda(T)$, and to this end our approach relies on the equivalence between solving the MDP and finding T^* as we explained in Section 3.2.3, we recall that equivalence below:

$$T^{\pi^*} = T^* = \text{Argmax}_T \mathcal{H}_\lambda(T), \pi^* = \text{Argmax}_\pi \mathcal{V}^\pi(\Omega)$$

Now let us consider a surrogate MDP with the modified rewards $\widehat{r}(T,a) = r(T,a)$ and $\widehat{r}(T,\overline{a}) = \widehat{\mathcal{H}_\lambda}(T)$. All the relevant quantities of this surrogate MDP will be denoted with a $\widehat{\ }$ symbol: $\widehat{r}, \widehat{\mathcal{R}}^\pi, \widehat{\mathcal{V}}^\pi, \widehat{\mathcal{Q}}^\pi$ are the reward, return, value and state-action value functions of this surrogate MDP respectively. Now suppose we naively apply an off-the shelf RL algorithm such as Q-Learning (Watkins and Dayan, 1992) to solve this MDP. Q-Learning would seek the policy maximising the value function from the initial state:

$$\begin{aligned}\widehat{\pi} &= \text{Argmax}_\pi \widehat{\mathcal{V}}^\pi(\Omega) = \text{Argmax}_\pi \mathbb{E}\left[\widehat{\mathcal{R}}^\pi(\Omega)\right] \\ &= \text{Argmax}_\pi \Big\{ -\lambda \mathcal{S}(T^\pi) + \mathbb{E}\left[\widehat{r}(T^\pi, \overline{a})\right] \Big\} \end{aligned} \tag{5.1}$$

The problem arises here from the fact that the estimates $\widehat{r}(T,\overline{a})$ are biased, we show this below:

$$\begin{aligned}\mathbb{E}\left[\widehat{r}(T,\overline{a})\right] &= \mathbb{E}\left[\sum_{l\in T} \widehat{\mathcal{H}}(l)\right] = \sum_{l\in T} \mathbb{E}\left[\widehat{\mathcal{H}}(l)\right] \\ &= \sum_{l\in T} \mathbb{E}\left[\max_{1\leq k\leq K} \frac{n_k(l)}{n}\right] \\ &\geq \sum_{l\in T} \max_{1\leq k\leq K} \mathbb{E}\left[\frac{n_k(l)}{n}\right] \\ &\geq \sum_{l\in T} \underbrace{\max_{1\leq k\leq K} \mathbb{P}\left[l(X)=1, Y=k\right]}_{\mathbb{P}[l(X)=1,Y=k^*(l)]=\mathcal{H}(l)} = \mathcal{H}(T)\end{aligned}$$

The inequality in the third line is due to the convexity of the max function and Jensen's inequality. Because of this bias, Eq. (5.1) does not guarantee that $\widehat{\pi}$ is equal to:

$$\pi^* = \text{Argmax}_\pi \mathcal{V}^\pi(\Omega) = \text{Argmax}_\pi \Big\{ \underbrace{-\lambda \mathcal{S}(T^\pi) + \mathcal{H}(T^\pi)}_{\neq -\lambda\mathcal{S}(T^\pi)+\mathbb{E}\left[\widehat{r}(T^\pi,\overline{a})\right]} \Big\}$$

This prompts us to look for a different estimate $\widehat{r}(T,\overline{a}) = \widehat{\mathcal{H}}(T)$. One solution to this problem could be to seek unbiased estimates of $\mathcal{H}(T)$ or at least to alleviate the issue due to the overestimation bias. A popular idea in this context is to decouple the estimation processes of the class probabilities $\mathbb{P}\left[Y=k|l(X)=1\right]$ and the majority class $k^*(l)$ by keeping two independent sets of samples for each process in a Double Learning fashion. This approach was used by Hasselt (2010) to alleviate a similar overestimation issue encountered by Q-Learning. Nonetheless, we chose to follow a different approach based on the fact that, despite being biased, the estimator $\widehat{\mathcal{H}}(T)$ is consistent:

$$\widehat{\mathcal{H}}(T) \xrightarrow[n\to\infty]{\text{a.s}} \mathcal{H}(T)$$

The consistency of $\widehat{\mathcal{H}}(T)$ implies the consistency of the value of any policy π. Indeed, let T be some state, $T_0 = T$ and $\forall t \geq 0 : T_t \xrightarrow{\pi(T_t)} T_{t+1}$, then the return satisfies:

$$\begin{aligned}\forall \pi : \widehat{\mathcal{R}}^\pi(T) &= \sum_{t=0}^{\infty} \widehat{r}(T_t, T_{t+1}) \\ &= -\lambda\left[\mathcal{S}(T^\pi(T)) - \mathcal{S}(T))\right] + \widehat{r}(T^\pi, \overline{a}) \\ &\xrightarrow[n\to\infty]{\text{a.s}} -\lambda\left[\mathcal{S}(T^\pi(T)) - \mathcal{S}(T))\right] + \mathcal{H}(T^\pi) = \mathcal{V}^\pi(T)\end{aligned}$$

This in turn trivially leads to the consistency of $\widehat{\mathcal{Q}}^\pi(T,a)$:

$$\forall T, a \in \mathcal{A}(T) : \widehat{\mathcal{Q}}^\pi(T,a) \xrightarrow[n\to\infty]{\text{a.s}} \mathcal{Q}^\pi(T,a)$$

One can verify this easily with the Bellman Equations in Proposition 1. To benefit from these consistency results, each time we take action $\overline{a}$ in a state T, we observe new m samples from the stream and accumulate them with the previously observed samples that were amassed during previous applications of $\overline{a}$ to T. The estimated values $\widehat{\mathcal{V}}^{\pi}$ are still biased, but we shift our focus from seeking the policy maximising the value $\widehat{\mathcal{V}}^{\pi}(\Omega) = \mathbb{E}\left[\widehat{\mathcal{R}}^{\pi}(\Omega)\right]$ to the policy maximising the return:

$$\widehat{\pi} = \text{Argmax}_{\pi}\widehat{\mathcal{R}}^{\pi}(\Omega)$$

and its asymptotic behaviour when the number of iterations grows to infinity. $\widehat{\pi}$ is random now by virtue of $\widehat{\mathcal{R}}^{\pi}(\Omega)$ being random itself. **What guarantees can we derive regarding the optimality of $\widehat{\pi}$?**

- These guarantees can come in the form of finite-time guarantees such as:

$$\mathbb{P}\left[\widehat{\pi} \neq \pi^*\right] \leq \mathcal{B}_1(N) \tag{5.2}$$

$$\left|\mathbb{E}\left[\mathcal{H}_{\lambda}\left(T^{\widehat{\pi}}\right)\right] - \mathcal{H}_{\lambda}(T^*)\right| \leq \mathcal{B}_2(N) \tag{5.3}$$

$$\mathbb{P}\left[\mathcal{H}_{\lambda}\left(T^{\widehat{\pi}}\right) \geq \mathcal{H}_{\lambda}(T^*) - \epsilon\right] \geq 1 - \mathcal{B}_3(\epsilon, N) \tag{5.4}$$

  Where $\mathcal{B}_1(N), \mathcal{B}_2(N), \mathcal{B}_3(\epsilon, N)$ are bounds that vanish to 0 as the number of iterations N of the algorithm increases. Eq. (5.2) states that the probability of $\widehat{\pi}$ being suboptimal decreases with N, while Eq. (5.3) and Eq. (5.4) are expected and PAC regret analyses respectively that would quantify how suboptimal $\widehat{\pi}$ is after running the algorithm for N iterations.

- These guarantees can also come in the form of an asymptotic convergence result:

$$\widehat{\pi} \xrightarrow[n\to\infty]{\text{a.s}} \pi^* \tag{5.5}$$

A rich theoretical analysis would include all of these guarantees. Unfortunately, the finite-time guarantees are very hard to derive in the context of MDPs, they are generally derived in the context of Multi-Armed Bandits (Lattimore and Szepesvári, 2020). The difficulty in the context of MDPs stems from the non-stationarity of the estimates and the lack of meaningful concentration inequalities at the level of each state as explained by Kocsis and Szepesvári (2006). A famous illustration of this difficulty is with regard to the analysis of the UCT algorithm, a Monte Carlo Tree Search algorithm that employs a UCB policy. In their original paper, (Kocsis and Szepesvári, 2006, Theorem 5) derives a finite-time guarantee in the form of (5.2), however, in a follow-up paper, (Kocsis et al., 2006, Theorem 6) modifies this result into an asymptotic convergence of the probability of suboptimality to 0 without providing a rate of convergence $\mathcal{B}_1(N)$. The authors further state in their proof:

Unfortunately, our methods are too crude to derive a meaningful convergence rate result on the failure probability. This is because we don't have sufficiently strong bounds for the concentration of $T_i(t)$ for suboptimal arms. However, we conjecture that the convergence rate is $(\log(t)/t)^{\kappa}$ for $0 < \kappa \leq 1$

Nevertheless, we will still attempt and derive a finite-time guarantee in the form of (5.2), but this will take a different version of our online algorithm, which we analyse in Chapter 7. In this work however, we will derive an asymptotic convergence guarantee for UCDT and EGDT in the form (5.5). While the finite-time guarantees are desirable properties, the asymptotic convergence is also important in its own right, especially since the finite-time guarantees do not necessarily imply asymptotic convergence.

5.3 The Algorithms: UCDT and EGDT

We design UCDT and EGDT as MCTS algorithms. The objective is to estimate the optimal state-action values $\mathcal{Q}^*(T,a) = \mathcal{Q}^{\pi^*}(T,a)$ since:

$$\forall T : \pi^*(T) = \text{Argmax}_{a \in \mathcal{A}(T)} \mathcal{Q}^*(T,a)$$

Two crucial questions arise now:

- **How shall we estimate $\mathcal{Q}^*(T,a)$?**
- **The space of state-action pairs is immense, hence we cannot estimate $\mathcal{Q}^*(T,a)$ for all possible states T and actions $a \in \mathcal{A}(T)$. Which pairs (T,a) should be chosen for these estimations?**

We answer these questions in the next two sections.

5.3.1 Estimating $\mathcal{Q}^*(T,a)$

Let T be a state. For the terminal action, $\mathcal{Q}^*(T,\overline{a}) = \mathcal{H}(T)$ and thus we can use the estimate:

$$\widehat{\mathcal{Q}}(T,\overline{a}) = \widehat{\mathcal{H}}(T) = \frac{1}{n}\sum_{l \in T} \max_{1 \le k \le K} n_k(l) = \frac{1}{n}\sum_{l \in T} \max_{1 \le k \le K} \sum_{t=1}^{n} l(X_t)\, \mathbb{1}\{Y_t = k\} \tag{5.6}$$

which is calculated based on all the accumulated observed samples $\{(X_t, Y_t)\}_{t=1}^n$ so far in T. Now consider a split action $a \in \mathcal{A}(T) \setminus \{\overline{a}\}$ and let $T \xrightarrow{a} T'$. Based on the Bellman Equations:

$$\begin{cases} \mathcal{Q}^*(T,a) & = -\lambda + \mathcal{V}^*(T') \\ \mathcal{V}^*(T') & = \max_{a' \in \mathcal{A}(T')} \mathcal{Q}^*(T',a') \end{cases}$$

We define our estimates accordingly via:

$$\begin{cases} \widehat{\mathcal{Q}}(T,a) & = -\lambda + \widehat{\mathcal{V}}(T') \\ \widehat{\mathcal{V}}(T') & = \max_{a' \in \mathcal{A}(T')} \widehat{\mathcal{Q}}(T',a') \end{cases} \tag{5.7}$$

This description is incomplete. If we stop here, then the recursive calculation of a single $\widehat{\mathcal{Q}}(T,a)$ would necessitate calculating $\widehat{\mathcal{Q}}(T^\pi(T),\overline{a})$ for all possible policies π and backpropagating them through Eq. (5.7). This is unfeasible due to the large number of such possible policies π. To alleviate this issue, we initialise the value estimate of all states to the state-action value of the terminal action taken at the same state:

$$\forall T\ \forall a \in \mathcal{A}(T) \setminus \{\overline{a}\} : \widehat{\mathcal{V}}(T) = \widehat{\mathcal{Q}}(T,\overline{a}) \tag{5.8}$$

There are two important things to note here. This initialisation is an arbitrary choice that serves as a heuristic kick-starting our search. The reason it is sound is that, in conjunction with an appropriate exploration-exploitation policy (as we will see in the next section), the asymptotic convergence result we are after (5.5) still holds. If we were after a fine-grained finite-time guarantee, then the choice behind this initialisation could be crucial. The second important remark is that, although the initialisation (5.8) is with regard to all states T, we do not have to manually perform it before running the algorithm as this would be unfeasible too because of the large size of the state space. Instead, whenever we encounter a new state, we initialise its value $\widehat{\mathcal{V}}(T)$ with Eq. (5.8), in this manner we avoid any unnecessary overhead due to this prior initialisation.

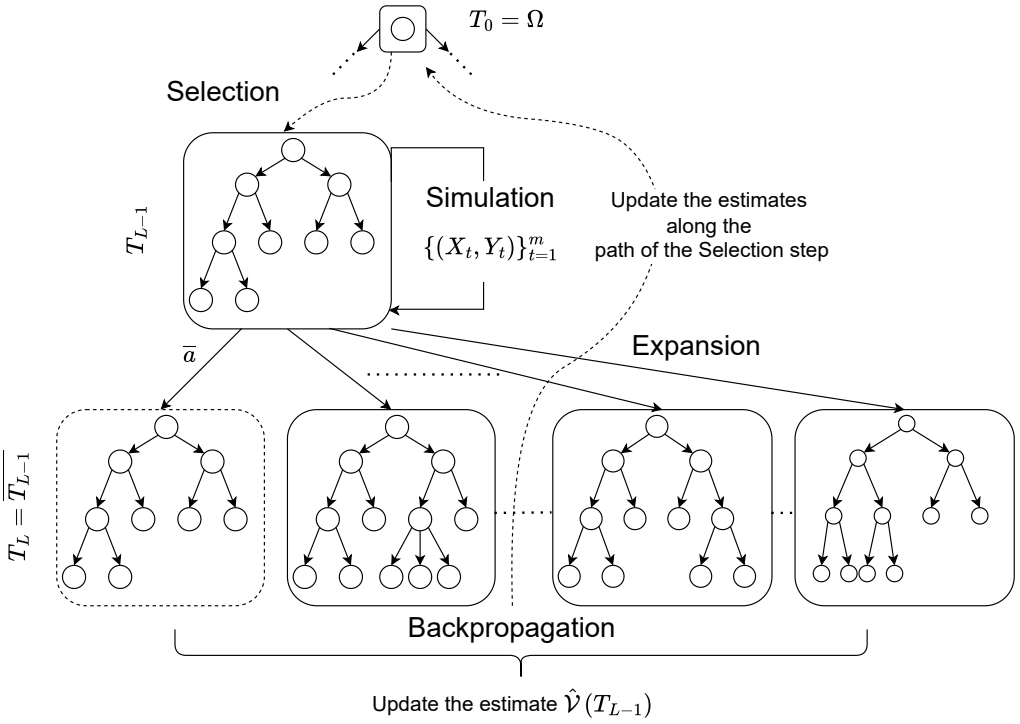


Figure 5.1: Algorithm 2 from the perspective of the Search Tree. The left-most child is drawn in dashed lines to emphasize that it is an absorbing state since it results from taking the terminal action $\overline{a}$ at T_{L-1}.

5.3.2 The Search strategy

We turn our attention now to the second question we asked in Section 5.3. Our approach can be viewed from the lens of Monte Carlo tree Search (MCTS) applied to the state space of Decision Trees, this means that the nodes of the Search Tree (our search space in graphical form) are Decision Trees themselves and its root is the root branch Ω. Essentially, we define a policy π that will serve as a guidance for our search proposing which states to use for our estimations; we will define π shortly for UCDT and EGDT independently. π is usually called a Tree policy in the MCTS literature. The general approach of UCDT and EGDT is described in Algorithm 2, we explain below the steps we perform at each iteration. Fig. 5.1 is a visual representation of Algorithm 2 from the graphical lens of the Search Tree.

- **Selection:** Line 4. Starting from Ω, follow π and get the trajectory:

$$\Omega = T_0 \xrightarrow{\pi(T_0)} T_1 \xrightarrow{\pi(T_1)} \dots \xrightarrow{\pi(T_{L-1})} T_L = \overline{T_{L-1}}$$

  From the Search Tree perspective, we start at Ω and we descend the Search Tree by choosing child nodes according to π until we encounter a leaf. *We emphasize that the notions of node and leaf employed in this context are with respect to the Search Tree graph, not any Decision Tree. In fact, the nodes of the Search Tree are themselves Decision Trees because they represent states of our MDP (See Fig. 5.1).*

- **Simulation:** Lines 6 and 10. In line 10, when T_{L-1} has never been visited (due to the else condition in line 9), we observe m new samples from the stream in T_{L-1}. Since T_{L-1}

is visited for the first time, the accumulated number of observed samples in T_{L-1} is m and we can denote these $\{(X_t, Y_t)\}_{t=1}^{m}$. These samples are used in initialising $\widehat{\mathcal{Q}}(T_{L-1}, a)$ for all actions $a \in \mathcal{A}(T_{L-1})$ as explained in lines 11 and 15. On the other hand, in line 6, since $v(T_{L-1}) \geq 2$ (if condition in line 5), then T_{L-1} has already been visited before, which means that $\widehat{\mathcal{Q}}(T_{L-1}, a)$ has also already been initialised for all actions, these estimates may have also been updated during some other iterations. The samples $\{(X_t, Y_t)\}_{t=1}^{m}$ can only update $\widehat{\mathcal{Q}}(T_{L-1}, \overline{a})$. Indeed, for any split action $a \in \mathcal{A}(T_{L-1}) \setminus \{\overline{a}\}$ with $T_{L-1} \xrightarrow{a} T$, the estimates $\widehat{\mathcal{Q}}(T_{L-1}, a)$ now depend on the estimates $\widehat{\mathcal{V}}(T')$ of descendants of T that we visited in previous iterations. *We emphasize here again that this notion of descendants is with respect to the Search Tree.* As a consequence of this, the samples $\{(X_t, Y_t)\}_{t=1}^{m}$ can only update $\widehat{\mathcal{Q}}(T_{L-1}, \overline{a})$. Since T_{L-1} has been visited before, then we have previously observed other samples in T_{L-1}, we accumulate the new samples with these to form the set of accumulated samples $\{(X_t, Y_t)\}_{t=1}^{n}$, which will be used to update $\widehat{\mathcal{Q}}(T_{L-1}, \overline{a})$ as explained in line 7.

- **Expansion:** Loop in line 12. This step only occurs when T_{L-1} has never been visited before. In this case, we need to initialise $\widehat{\mathcal{Q}}(T_{L-1}, a)$ for all actions $a \in \mathcal{A}(T_{L-1})$. For $\widehat{\mathcal{Q}}(T_{L-1}, \overline{a})$, this is achieved in line 11. As for the split actions $a \in \mathcal{A}(T_{L-1}) \setminus \{\overline{a}\}$, we expand T_{L-1} by considering its children (again in the context of the Search Tree) $T_{L-1} \xrightarrow{a} T$, for which we initialise $\widehat{\mathcal{V}}(T)$ according to line 14 and the observed data from the Simulation step. these estimates $\widehat{\mathcal{V}}(T)$ are in turn employed to initialise the estimates $\widehat{\mathcal{Q}}(T_{L-1}, a)$ as explained in line 15. All of the estimates $\widehat{\mathcal{Q}}(T_{L-1}, a)$ are then utilised to update $\widehat{\mathcal{V}}(T_{L-1})$ and the policy $\pi(T_{L-1})$ in lines 18 and 19 respectively.

- **Backpropagation:** Loop in line 21. The objective of this step is to use the estimate $\widehat{\mathcal{V}}(T_{L-1})$ in order to recursively update the estimates $\widehat{\mathcal{V}}(T_j), \widehat{\mathcal{Q}}(T_j, \pi(T_j))$ and the policy $\pi(T_j)$ along $j = L-1, \ldots, 1$. From the Search Tree perspective, $T_{L-2}, \ldots, T_0 = \Omega$ are the ancestors of T_{L-1} along the path we took during the Selection step. The updates we perform in this step can be seen as if we backpropagate $\widehat{\mathcal{V}}(T_{L-1})$ up the Search Tree to update the relevant estimates. Hence the name Backpropagation.

At the end of the M iterations, we propose a policy $\widehat{\pi}$ according to the estimates $\widehat{\mathcal{Q}}(T, a)$ we calculated:

$$\forall T : \widehat{\pi}(T) = \text{Argmax}_{a \in \mathcal{A}(T)} \widehat{\mathcal{Q}}(T, a)$$

Then we unroll $\widehat{\pi} : \Omega = T_0 \xrightarrow{\pi(T_0)} T_1 \xrightarrow{\pi(T_1)} \ldots \xrightarrow{\pi(T_{L-1})} T_L = \overline{T_{L-1}}$, in which case $T_{L-1} = T^{\widehat{\pi}}$ is our proposed solution.

The only step remaining to instantiate a concrete algorithm is to define the policy updates in Lines 19 and 24.

UCDT: UCB policy update:

$$\pi(T) = \text{Argmax}_{a \in \mathcal{A}(T)} \left\{ \widehat{\mathcal{Q}}(T, a) + \gamma \sqrt{\frac{\log v(T)}{v(T')}} \right\} \tag{5.9}$$

Where $\gamma > 0$ is a hyperparameter that controls the exploration-exploitation trade-off, and $v(T)$ is the number of times we visited state T. In the Argmax, we implicitly used $T \xrightarrow{a} T'$.

Proposition 17. *For any state T, UCDT satisfies the optimal convergence property:*

$$\widehat{\mathcal{V}}(T) \xrightarrow[M\to\infty]{a.s} \mathcal{V}^*(T), \ \widehat{\pi}(T) \xrightarrow[M\to\infty]{a.s} \pi^*(T)$$

EGDT: Epsilon-greedy policy update:

$$\begin{cases} \pi(a|T) = 1 - \frac{1}{v(T)^\gamma} + \frac{1}{v(T)^\gamma |\mathcal{A}(T)|} \text{ if } a = \text{Argmax}_{a' \in \mathcal{A}(T)} \left\{ \widehat{\mathcal{Q}}(T, a') \right\} \\ \pi(a|T) = \frac{1}{v(T)^\gamma |\mathcal{A}(T)|} \text{ otherwise} \end{cases} \tag{5.10}$$

$0 < \gamma \leq 1$ controls the exploration-exploitation trade-off. π is a stochastic policy here unlike all the policies we have used so far. For any state T and action $a \in \mathcal{A}(T)$, the term $\pi(a|T)$ is the probability of performing a in T according to π. (5.10) corresponds to choosing a random action with probability $\frac{1}{v(T)^\gamma}$ or otherwise choosing the optimal action with respect to the estimated state-action values $\widehat{\mathcal{Q}}(T, a)$.

Proposition 18. *For any state T, EGDT satisfies the optimal convergence property:*

$$\widehat{\mathcal{V}}(T) \xrightarrow[M\to\infty]{a.s} \mathcal{V}^*(T), \ \widehat{\pi}(T) \xrightarrow[M\to\infty]{a.s} \pi^*(T)$$

The condition $0 < \gamma \leq 1$ is crucial in our proof of Proposition 18. It stems from the fact that the series $\sum_n \frac{1}{n^\gamma}$ diverges. EGDT is not guaranteed to converge optimally if $\gamma > 1$, at least not in our proof, the intuition behind this remark is that its level of exploration could be insufficient due to the rapid decrease of $\frac{1}{v(T)^\gamma}$.

5.4 Sample Efficiency and Weights Degeneracy

So far we have introduced our algorithms with no Dynamic Programming memoisation component. While DP is not necessary for Proposition 17 and Proposition 18 to hold, neglecting it results in impractical algorithms with large computational costs. This section is dedicated to incorporating DP and discussing the issues that arise from this procedure.

The MDP we employed in this work is similar to DT-Miner's, for this reason we can use a similar caching procedure where branches are identified uniquely by their clauses. The objective is to accumulate data in branches whenever they are encountered regardless of which state (DT) is being considered for the Simulation step, this will accelerate convergence and render the method much more sample efficient. Indeed, consider two different DTs $T \neq T'$ that share some common branch $l \in T \cap T'$. When T is selected and simulated, we observe data from the stream in l, and the same thing happens when we select and simulate T'. However, when DP is not taken into consideration, we observe different sets of data in l in these different cases as opposed to accumulating them. It is as if $l \in T$ and $l \in T'$ are treated as different branches. Hence the sample inefficiency of this approach. Let us now introduce the full concrete procedure. For each branch $l = \bigwedge_{v=1}^{\mathcal{S}(l)} \mathbb{1}\{X^{(i_v)} = j_v\}$, we store the following running statistics:

$$\forall i \notin \{i_1, \ldots, i_{\mathcal{S}(l)}\} \ \forall j \in \{1, \ldots, C_i\} \ \forall k \in \{1, \ldots, K\} : n_{ijk}(l) = \sum_{t=1}^{n} l(X_t) \mathbb{1}\{X_t^{(i)} = j\} \mathbb{1}\{Y_t = k\} \tag{5.11}$$

where $\{(X_t, Y_t)\}_{t=1}^{n}$ are all the accumulated observed samples in l (which come from different simulation steps of different DTs that share l). We recall that $\{1, \dots, q\} \setminus \{i_1, \dots, i_{|\mathcal{S}(l)|}\}$ is the set of all features we can use for splitting l. $n_{ijk}(l)$ is the number of observed samples that are in l, for which the i^{th} feature is equal to j, and that are of class k. The reason behind defining $n_{ijk}(l)$ pertains to making the Expansion step more efficient as we will see shortly. These types of statistics are used in the literature of online Decision Trees, a popular case can be found in (Domingos and Hulten, 2000). Suppose now that we have just selected and simulated a DT T with $l \in T$. During Expansion, our first objective is to calculate $\widehat{\mathcal{Q}}(T, a)$ for all actions $a \in \mathcal{A}(T)$. For the terminal action, using Eq. (5.6), we have:

$$\widehat{\mathcal{Q}}(T, \overline{a}) = \frac{1}{N(T)} \sum_{l \in T} \max_{1 \leq k \leq K} n_k(l) = \frac{1}{N(T)} \sum_{l \in T} \max_{1 \leq k \leq K} \sum_{i \notin \{i_1, \dots, i_{\mathcal{S}(l)}\}} \sum_{j=1}^{C_i} n_{ijk}(l) \tag{5.12}$$

It is straightforward to verify that $n_k(l) = \sum_{i \in \mathcal{A}(l) \setminus \{\overline{a}\}} \sum_{j=1}^{C_i} n_{ijk}(l)$. Moreover, it is very important to note here that $N(T)$ is the total number of accumulated observed samples across all the branches in T, which translates to:

$$N(T) = \sum_{l \in T} \sum_{i \notin \{i_1, \dots, i_{\mathcal{S}(l)}\}} \sum_{j=1}^{C_i} \sum_{k=1}^{K} n_{ijk}(l)$$

Now let us consider a split action $(l, i) \in \mathcal{A}(T) \setminus \{\overline{a}\}$ and initialise $\widehat{\mathcal{Q}}(T, i)$. Using Eq. (5.7) and Eq. (5.8) we get:

$$\widehat{\mathcal{Q}}(T, (l, i)) = -\lambda + \widehat{\mathcal{Q}}(T(l, i), \overline{a}) = -\lambda + \frac{1}{N(T)} \sum_{l \in T(l,i)} \max_{1 \leq k \leq K} \sum_{i \in \mathcal{A}(l) \setminus \{\overline{a}\}} \sum_{j=1}^{C_i} n_{ijk}(l) \tag{5.13}$$

where $T(l, i)$ is the next state that stems from taking action (l, i) in T: $T \xrightarrow{(l,i)} T(l, i)$, it was defined in Section 3.2.3. With Eq. (5.12) and Eq. (5.13), we see how the $n_{ijk}(l)$ statistics (5.11) are sufficient to calculate $\widehat{\mathcal{Q}}(T, a)$ for all actions $a \in \mathcal{A}(T)$. Thus once we update the $n_{ijk}(l)$ statistics, which can be done efficiently in one time during the Simulation of T, we can straightforwardly calculate $\widehat{\mathcal{Q}}(T, a)$ for all $a \in \mathcal{A}(T)$, yielding an efficient Expansion step.

Employing the $n_{ijk}(l)$ statistics along with DP sounds very promising from the perspective of speeding up convergence. However, there is a subtle fatal flaw in this approach as it currently stands, it arises from the definition of $\widehat{\mathcal{Q}}(T, \overline{a})$ in Eq. (5.12), and more precisely from dividing by $N(T)$. As we shall explain shortly, this flaw leads to the inconsistency of $\widehat{\mathcal{Q}}(T, \overline{a})$:

$$\lim_{M \to \infty} \widehat{\mathcal{Q}}(T, \overline{a}) \neq \mathcal{Q}^*(T, \overline{a})$$

This means that as the algorithm runs more and more ($M \longrightarrow \infty$), the estimate $\widehat{\mathcal{Q}}(T, \overline{a})$ is no longer guaranteed to converge almost surely to the true optimal value $\mathcal{Q}^*(T, \overline{a})$. Let us investigate the reason behind this flaw. In order to show the consistency of $\widehat{\mathcal{Q}}(T, \overline{a})$, we write:

$$\widehat{\mathcal{Q}}(T, \overline{a}) = \frac{1}{N(T)} \sum_{l \in T} \max_{1 \leq k \leq K} n_k(l) = \sum_{l \in T} \frac{\max_{1 \leq k \leq K} n_k(l)}{N(T)}$$

and we show that $\frac{\max_{1 \leq k \leq K} n_k(l)}{N(T)} \xrightarrow[M \to \infty]{\text{a.s}} \mathbb{P}[l(X) = 1, k^*(l) = Y]$ for all the branches $l \in T$. To see how this is no longer guaranteed, let us consider the example in Fig. 5.2. Suppose

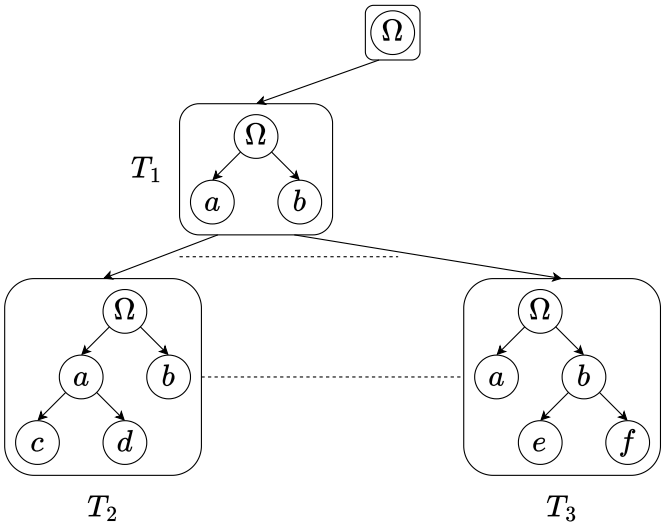


Figure 5.2: Illustration of the Weights Degeneracy issue.

we are simulating T_2, then we observe samples in branches c and d for the first time, however, this is not the case for b. Indeed, when simulating T_2, we are observing new samples in b that are accumulated with samples we previously observed in b when T_1 was simulated in a previous iteration of the algorithm. Therefore, we are observing more samples in b than it should be relatively to c and d, which skews the proportions and as a consequence the fractions $\frac{\max_{1\leq k\leq K} n_k(c)}{N(T)}, \frac{\max_{1\leq k\leq K} n_k(d)}{N(T)}$ underestimate $\mathbb{P}\left[c\left(X\right)=1, k^*\left(c\right)=Y\right]$ and $\mathbb{P}\left[d\left(X\right)=1, k^*\left(d\right)=Y\right]$ respectively while $\frac{\max_{1\leq k\leq K} n_k(b)}{N(T)}$ overestimates $\mathbb{P}\left[b\left(X\right)=1, k^*\left(b\right)=Y\right]$, and the consistency property of $\widehat{\mathcal{Q}}\left(T_2, \overline{a}\right)$ is therefore lost. We call this phenomenon **Weights Degeneracy**. Notice that this same phenomenon happens between a and e, f when simulating T_3. Moreover, in this case we also observe and accumulate additional samples in b while neglecting c and d, making the Weights Degeneracy effect even stronger. The important question to ask now is: **How can we avoid the Weights Degeneracy phenomenon?**

To answer this question, we go back to the original form of $\mathcal{Q}^*\left(T, \overline{a}\right)$ and rewrite it as follows:

$$\mathcal{Q}^*\left(T, \overline{a}\right) = \mathcal{H}\left(T\right) = \mathbb{P}\left[T\left(X\right)=Y\right] = \sum_{l\in T} \mathbb{P}\left[T\left(X\right)=Y | l\left(X\right)=1\right] \mathbb{P}\left[l\left(X\right)=1\right]$$

We estimate each of the two terms in the sum independently. Estimating $\mathbb{P}\left[T\left(X\right)=Y | l\left(X\right)=1\right]$ is straightforward via:

$$\frac{\max_{1\leq k\leq K} n_k\left(l\right)}{n\left(l\right)}$$

where we recall that $n\left(l\right)$ is the total number of observed samples in l and $n_k\left(l\right)$ is the total number of observed samples in l that are of class k. For the second term $\mathbb{P}\left[l\left(X\right)=1\right]$, we need to be careful as estimating it with $\frac{n(l)}{N(T)}$ gets us back to the Weights Degeneracy problem. Instead, we will define an alternative estimator based on the chain rule:

$$\mathbb{P}\left[l\left(X\right)=1\right] = \mathbb{P}\left[l\left(X\right)=1, P\left(l\right)\left(X\right)=1, \ldots, P^{\mathcal{S}(l)}\left(l\right)\left(X\right)=1\right]$$

where we define:

$$\begin{cases} P^0\left(l\right) = l \\ \forall 1 \leq i \leq \mathcal{S}\left(l\right) : P^i\left(l\right) = P\left(P^{i-1}\left(l\right)\right) \end{cases}$$

$P(l) = P^1(l)$ is the parent branch of l in T and $P^i(l)$ is the i^{th} ancestor of l in T. Hence $P^{\mathcal{S}(l)}(l) = \Omega$ and thus $\mathbb{P}\left[P^{\mathcal{S}(l)}(l)(X) = 1\right] = 1$. This leads to:

$$\mathbb{P}[l(X) = 1] = \prod_{i=0}^{\mathcal{S}(l)-1} \mathbb{P}\left[P^i(l)(X) = 1 | P^{i+1}(l)(X) = 1\right]$$

We estimate each term $\mathbb{P}\left[P^i(l) = 1 | P^{i+1}(l) = 1\right]$ with:

$$\widehat{p}\left(P^i(l) | P^{i+1}(l)\right) = \frac{n\left(P^i(l)\right)}{\sum_{\eta \in \text{Ch}(P^{i+1}(l), i)} n(\eta)}$$

$\text{Ch}\left(P^{i+1}(l), i\right)$ is the set of children branches that stem from splitting $P^{i+1}(l)$ with respect to the i^{th} feature, we defined in Section 3.2.3. Our new estimate of $\mathbb{P}[l(X) = 1]$ is therefore:

$$\widehat{p}(l) = \prod_{i=0}^{\mathcal{S}(l)-1} \widehat{p}\left(P^i(l) | P^{i+1}(l)\right) \tag{5.14}$$

The reason $\widehat{p}\left(P^i(l) | P^{i+1}(l)\right)$ is consistent, i.e.

$$\widehat{p}\left(P^i(l) | P^{i+1}(l)\right) \xrightarrow[M \to \infty]{\text{a.s}} \mathbb{P}\left[P^i(l)(X) = 1 | P^{i+1}(l)(X) = 1\right]$$

is because all the children branches $\eta \in \text{Ch}\left(P^{i+1}(l), i\right)$ are shared among exactly the same DTs across the whole Search Tree, thus whenever we simulate a DT that is containing one of these branches, we observe samples in all of the children $\text{Ch}\left(P^{i+1}(l), i\right)$ as well, thus avoiding Weights Degeneracy. As a consequence we get the consistency of $\widehat{p}(l)$:

$$\widehat{p}(l) \xrightarrow[M \to \infty]{\text{a.s}} \mathbb{P}[l(X) = 1]$$

Thus establishing the consistency of the new estimate of $\mathcal{Q}^*(T, \overline{a})$ as well:

$$\widehat{\mathcal{Q}}(T, \overline{a}) = \sum_{l \in T} \widehat{p}(l) \frac{\max_{1 \leq k \leq K} n_k(l)}{n(l)} \xrightarrow[M \to \infty]{\text{a.s}} \mathcal{Q}^*(T, \overline{a}) \tag{5.15}$$

This concludes the incorporation of Dynamic Programming in UCDT and EGDT. We note that, from a practical standpoint, employing DP without this new estimator led indeed to bad results that never retrieved the optimal solution. Utilising these new estimates in conjunction with DP solved this issue and led to the desired results in practice.

5.5 Experiments

We compare UCDT and EGDT with the state of the art on a series of data streams with categorical features. For this purpose, we use the recent and very popular River Python library (Montiel et al., 2021). River is the result of a merger between the prior online Machine Learning libraries creme (Halford et al., 2019) and scikit-multiflow (Montiel et al., 2018b). It contains implementations of several online DT classifiers: VFDT (Domingos and Hulten, 2000), HAT (Bifet and Gavalda, 2009), EFDT (Manapragada et al., 2018) and SGT (Gouk et al., 2019). River also provides many data streams and a module allowing easy implementations of user-defined streams. We compare UCDT and EGDT with all the algorithms cited above on all the data streams with categorical features in River. Furthermore, we also define four new problems where the true classification function is XOR on the first two features. Each one of

Table 5.1: Number of features q, number of categories per feature C, number of classes K and penalty parameter λ for the different data streams we used.

Stream	q	C	K	λ
led	7	2	10	0.01
led-irrelevant	24	2	10	0.01
random-tree-1	10	2	2	0.001
random-tree-2	10	4	2	0.001
random-tree-3	20	5	2	0.001
stagger	3	3	2	0.001
xor-1	50	2	2	0.01
xor-2	100	2	2	0.01
xor-3	200	2	2	0.01
xor-4	500	2	2	0.01

these handcrafted streams contains a different number of features q, and since only the first two features are relevant to this XOR function, the higher q is, the more challenging finding the true optimal DT becomes. Table 5.1 describes the experimental setup for each data stream. In all our experiments, we observe $m = 100$ samples per Simulation step, and we run each algorithm for 2×10^5 stream samples while limiting its execution time to 5 minutes. Setting a time limit is especially important for UCDT and EGDT because they are more memory consuming due to storing and updating the search tree structure. Thus, when run for too long they risk running out of memory. We note that the γ parameter controls the Exploration-Exploitation trade-off. It can be tuned through crossvalidation using subsets of data from the stream. Our preliminary experiments showed the value 0.1 to be practical and we settled for it in all the experiments. The reason for this small value is that we noticed UCDT and EGDT to be sensitive to high γ values, in which case they put a lot of weight on exploration, thus greatly slowing down their convergence. Nevertheless, accurately quantifying this sensitivity warrants further investigation that we leave for a future work. For reproducibility, we set a seed of 42 for all the experiments.

Table 5.2 provides the accuracies of the final solutions proposed by the different algorithms, along with their number of splits. In addition, we evaluate the algorithms via their prequential accuracies on a sliding window of size 100 samples. Upon the arrival of a new sample from the data stream, the sample is used to estimate the accuracy of the algorithm's current solution before being employed for training. A sliding window of 100 samples estimates the accuracy, called prequential accuracy, for each method. Furthermore, we also report the complexity, in terms of the number of splits, of the proposed solutions as a function of the number of observed samples so far. Table 5.2, Fig. 5.3 and Fig. 5.4 show the superiority of UCDT and EGDT. On LED and LED-irrelevant, they reach the optimal solution with 100% accuracy and 9 splits significantly faster, in terms of the number of consumed samples, than the other methods. In this regard, VFDT and HAT provide a satisfactory performance, reaching the true optimal solution, albeit much later than UCDT and EGDT. On the other hand, EFDT displays an unstable training. Although it reaches the optimal solution very fast, it deviates from it often and does not stabilise its solution even after all the 2×10^5 samples have been observed. This can be explained by EFDT's fast splitting process that does not make use of statistical tests unlike VFDT and HAT. However, once the splits are performed, EFDT starts revising and replacing them, hence inducing the observed instability. Lastly, SGT fails at recovering the optimal solution.

Table 5.2: Comparing UCDT and EGDT with the state of the art online DTs implemented in River. The evaluations are in term of the accuracy of the final DT proposed by the algorithms and its number of splits.

Stream	VFDT		EFDT		SGT		HAT		UCDT		EGDT	
	accuracy	splits	accuracy	splits	accuracy	splits	accuracy	splits	accuracy	splits	accuracy	splits
led	**1**	**9**	0.94	8	0.14	6	**1**	**9**	**1**	**9**	**1**	**9**
led-irrelevant	**1**	**9**	**1**	**9**	0.69	10	**1**	**9**	**1**	**9**	**1**	**9**
random-tree-1	**1**	35	0.93	27	0.13	598	**1**	35	0.873	13	0.881	13
random-tree-2	0.89	75	0.95	117	0.77	10	0.91	87	0.847	5	0.841	4
random-tree-3	0.84	115	0.85	115	0.65	20	0.79	5	0.760	4	0.787	6
stagger	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**
xor-1	**1**	**3**	0.45	7	0.57	21	0.61	64	**1**	**3**	**1**	**3**
xor-2	0.45	67	0.8	5	0.48	31	0.54	65	**1**	**3**	**1**	**3**
xor-3	0.50	61	0.55	3	0.53	32	0.51	63	**1**	**3**	**1**	**3**
xor-4	0.54	31	0.49	3	0.51	16	0.45	31	**1**	**3**	0.46	3

On the random tree generators, the results are inconclusive. While VFDT, HAT and EFDT exhibit better prequential accuracy than UCDT and EGDT on the long run, they also propose significantly more complex solutions, thus incurring large penalties with regards to sparsity. On the XOR experiments, as we explained in Section 1.2, the greedy top-down construction methods struggle with XOR, especially in the presence of many irrelevant features. Fig. 5.4 shows this issue clearly. In fact, this is not a limitation that can be solved with better statistical tests. Even if the true Entropy and Gini metrics were available, the problem would still remain as it pertains to the myopic property of these impurity metrics. It is therefore interesting to notice that UCDT and EGDT successfully retrieve the optimal sparse DT, that has only three splits, while also employing very few samples to achieve this. On the hardest challenge with 498 irrelevant features, XOR 4, EGDT fails to retrieve the optimal DT while UCDT succeeds. As such, UCDT might satisfy better finite-time guarantees than EGDT. Unfortunately, our asymptotic convergence guarantees Proposition 17 and Proposition 18 do not allow us to draw such conclusion. This warrants further investigation in the future.

As a final remark, observe that in many cases, the functions describing the prequential accuracies and splits of UCDT and EGDT abruptly stop well ahead of 2×10^5 samples. This is due to reaching the timeout limit of 5 minutes. On the other hand, the greedy methods rarely reached timeout, they can treat more samples faster than our algorithms. In cases where data arrives from a stream at very a fast pace, UCDT and EGDT might not be practical because they are significantly slower at treating each subset of data compared to the greedy methods.

5.6 Conclusion

The problem of seeking optimal sparse DTs in the online setting is yet to be thoroughly investigated. For this reason, we proposed in this chapter our first approach for tackling this problem via MCTS, resulting in two algorithms UCDT and EGDT. From a theoretical perspective, we showed that both UCDT and EGDT achieve optimal convergence asymptotically. This means

that given an infinite stream of data, our algorithms are guaranteed to find the optimal DT. We tested this quality empirically and we showed that UCDT and EGDT outperform the current state of the art in terms of the quality of the retrieved solutions. More importantly, we showed how the existing algorithms are practically guaranteed to be suboptimal in situations where our methods are still optimal. Nevertheless, this advantage might be irrelevant in cases where data arrive at a very fast pace because our MCTS methods treat data significantly slower than the greedy methods.

UCDT and EGDT employ frequentist policies. In the next chapter, we shall investigate a Bayesian approach using Thompson Sampling. We will show that such approach also satisfies asymptotic optimal convergence. Nonetheless, we are still unable to derive finite-time guarantees for these methods. The allure of these results is that they permit a more quantitative assessment of the algorithm's performance after a set number of iterations. In Chapter 7, we shall consider a different approach for optimal online DTs, for which we are able to derive finite-time guarantees.

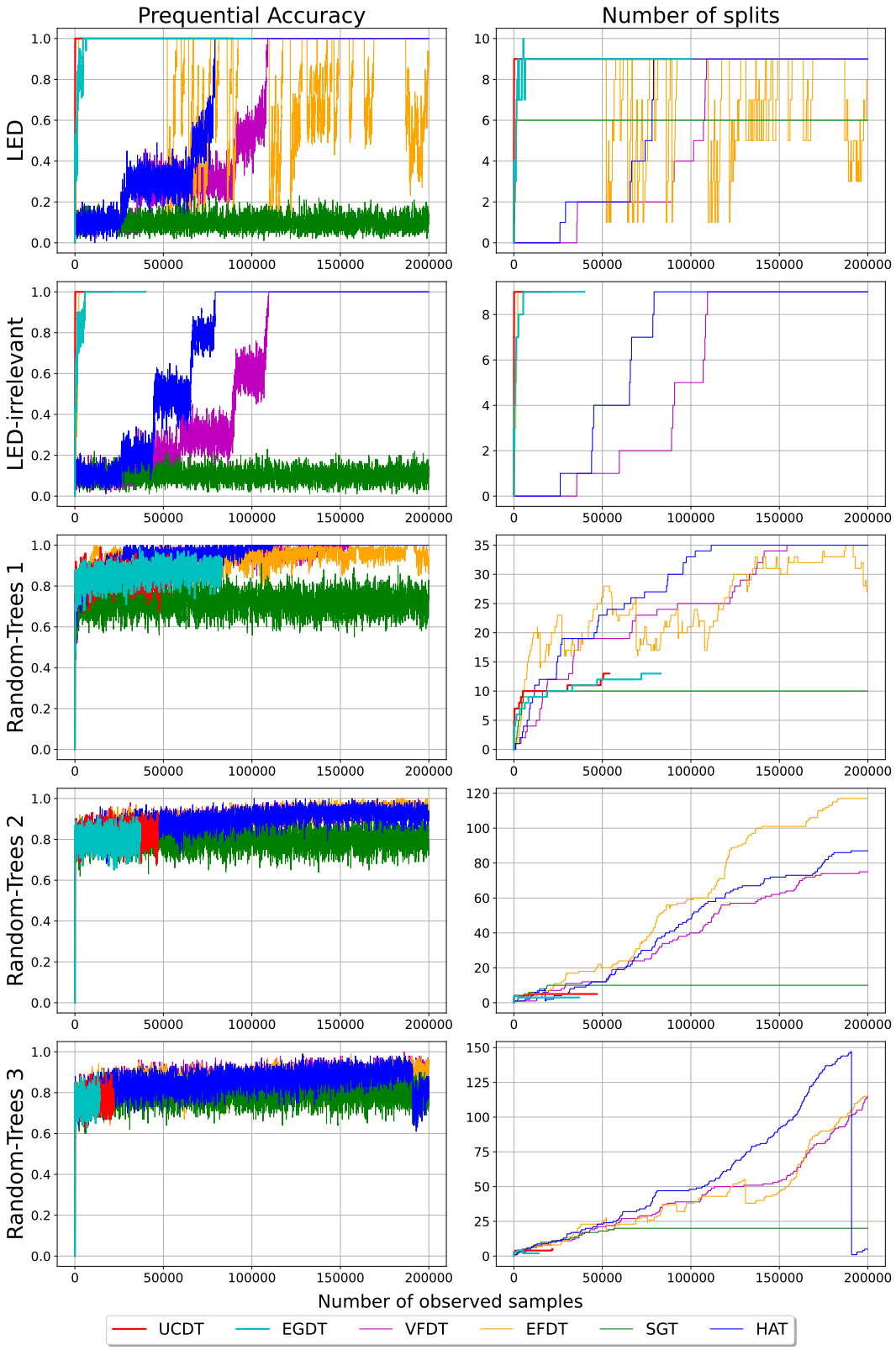


Figure 5.3: The left column reports the prequential accuracy of each method as a function of the number of observed samples so far. The right column reports the number of splits of the solution for each algorithm as a function of the number of observed samples so far.

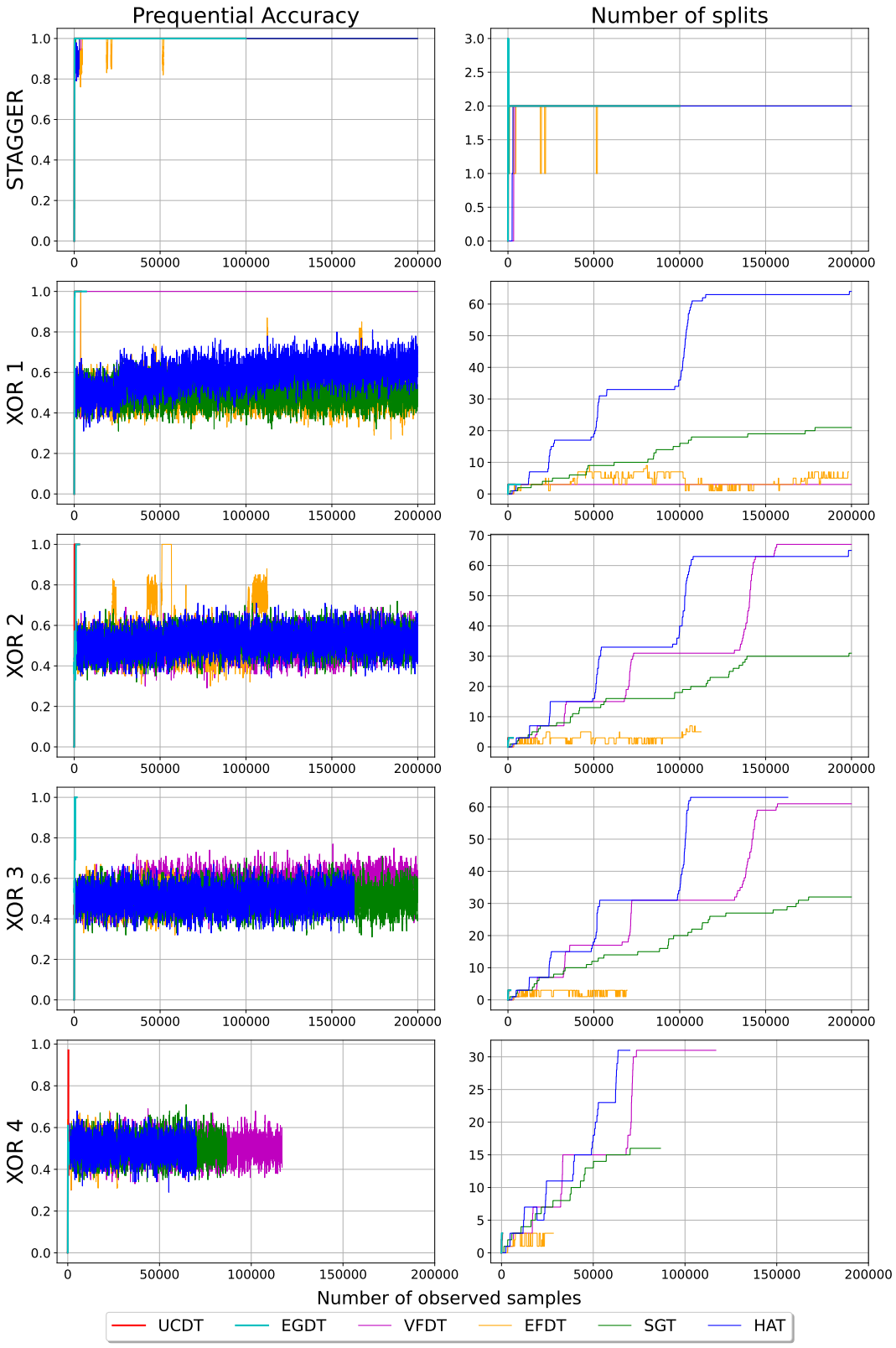


Figure 5.4: The remaining experimental results.

5.7 Appendix: Proofs

In this Appendix, our objective is to prove Proposition 17 and Proposition 18. We conduct this proof directly for the case where we employ Dynamic Programming, thus the estimates are the ones we defined in Section 5.4.

Lemma 19. *Let $\overline{T}$ be a terminal state. Then for all branches $l \in T$ with $p(l) > 0$, we have:*

$$\forall 1 \leq k \leq K : \widehat{p}_k(l) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} p_k(l)$$

where $\widehat{p}_k(l) = \frac{n_k(l)}{n(l)}, p_k(l) = \mathbb{P}[T(X) = Y | l(X) = 1]$ and $p(l) = \mathbb{P}[l(X) = 1]$

Proof Let $\epsilon > 0$, we want to show that:

$$\mathbb{P}\left[\exists n > 0, \forall v\left(\overline{T}\right) > n : |\widehat{p}_k(l) - p_k(l)| < \epsilon\right] = 1$$

We recall that $n(l)$ denotes the number of observed samples in branch l. By the Strong Law of Large Numbers, we have $\widehat{p}_k(l) \xrightarrow[n(l)\to\infty]{\text{a.s}} p_k(l)$, i.e.

$$\mathbb{P}\left[\exists n_l > 0, \forall n(l) > n_l : |\widehat{p}_k(l) - p_k(l)| < \epsilon\right] = 1$$

On the other hand, since $p(l) > 0$, we have $n(l) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} \infty$, i.e.

$$\mathbb{P}\left[\forall n_l > 0, \exists n > 0, \forall v\left(\overline{T}\right) > n : n(l) > n_l\right] = 1$$

Therefore, we deduce that:

$$\mathbb{P}\left[\exists n > 0, \forall v\left(\overline{T}\right) > n : |\widehat{p}_k(l) - p_k(l)| < \epsilon\right] = 1 \tag{5.16}$$

Furthermore, we write:

$$\begin{aligned}
&\mathbb{P}\left[\forall n > 0, \exists n_0 > 0, \forall v\left(\overline{T}\right) > n_0 : |\widehat{p}_k(l) - p_k(l)| \leq \frac{1}{n}\right] \\
&\geq 1 - \sum_{n=1} \underbrace{\mathbb{P}\left[\forall n_0 > 0, \exists v\left(\overline{T}\right) > n_0 : |\widehat{p}_k(l) - p_k(l)| > \frac{1}{n}\right]}_{=0 \text{ Because of Eq. (5.16)}}
\end{aligned}$$

Therefore $\mathbb{P}\left[\forall n > 0, \exists n_0 > 0, \forall v\left(\overline{T}\right) > n_0 : |\widehat{p}_k(l) - p_k(l)| \leq \frac{1}{n}\right] = 1$ concluding the proof. ■

Lemma 20. *For any terminal state $\overline{T}$, we have the following:*

$$\widehat{\mathcal{V}}\left(\overline{T}\right) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} \mathcal{V}^*\left(\overline{T}\right)$$

Proof

$$\widehat{\mathcal{V}}\left(\overline{T}\right) = \sum_{l\in T} \widehat{p}(l) \max_{1\leq k\leq K} \widehat{p}_k(l)$$

Consider a branch $l \in T$, if $p(l) = 0$, then trivially $\widehat{p}(l) = 0$. Otherwise, the Strong Law of Large Numbers stipulates that:

$$\widehat{p}(l) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} p(l)$$

On the other hand, Lemma 19 states that for all branches $l \in T$ with $p(l) > 0$ satisfy:

$$\forall 1 \leq k \leq K : \widehat{p}_k(l) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} p_k(l)$$

Thus we have:

$$\max_{1\leq k\leq K} \widehat{p}_k(l) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} \max_{1\leq k\leq K} p_k(l)$$

Therefore, we deduce that:

$$\widehat{\mathcal{V}}\left(\overline{T}\right) = \sum_{l\in\mathcal{L}(T)} \widehat{p}(l) \max_{1\leq k\leq K} \widehat{p}_k(l) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} \sum_{l\in\mathcal{L}(T)} p(l) \max_{1\leq k\leq K} p_k(l) = \mathcal{V}^*\left(\overline{T}\right)$$

■

Lemma 21. *Let T be a state, then for any action $a \in \mathcal{A}(T)$ with $T \xrightarrow{a} T'$ we have:*

$$\widehat{\mathcal{Q}}(T,a) \xrightarrow[v(T)\to\infty]{\text{a.s}} \mathcal{Q}^*(T,a)$$
$$\widehat{\mathcal{V}}(T) \xrightarrow[v(T)\to\infty]{\text{a.s}} \mathcal{V}^*(T)$$
$$v(T') \xrightarrow[v(T)\to\infty]{\text{a.s}} \infty$$

Proof We conduct this proof by Backward Induction on $\mathcal{S}(T)$. If $\mathcal{S}(T)$ is the maximum number of splits possible, then $\mathcal{A}(T) = \{\overline{a}\}$ and thus:

$$\forall a \in \mathcal{A}(T) : \widehat{\mathcal{V}}(T) = \widehat{\mathcal{Q}}(T,a) = \widehat{\mathcal{Q}}(T,\overline{a}) = \widehat{\mathcal{V}}\left(\overline{T}\right)$$

On the other hand, according to Lemma 20 we have:

$$\widehat{\mathcal{V}}\left(\overline{T}\right) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} \mathcal{V}^*\left(\overline{T}\right) = \mathcal{V}^*(T) = \mathcal{Q}^*(T,\overline{a})$$

Moreover, since $\mathcal{A}(T) = \{\overline{a}\}$, whenever we visit T we also visit $\overline{T}$, which means that $v(T) = v\left(\overline{T}\right)$ and therefore:

$$\widehat{\mathcal{Q}}(T,\overline{a}) = \widehat{\mathcal{V}}\left(\overline{T}\right) \xrightarrow[v(T)\to\infty]{\text{a.s}} \mathcal{V}^*(T) = \mathcal{Q}^*(T,\overline{a})$$

Now suppose that this is satisfied for all numbers of splits $\geq n$ where n is an integer smaller than the maximum number of splits. For any split action $a \in \mathcal{A}(T)$ with $T \xrightarrow{a} T'$, $\mathcal{S}(T') \geq n$ and the induction hypothesis yields:

$$\widehat{\mathcal{V}}(T') \xrightarrow[v(T')\to\infty]{\text{a.s}} \mathcal{V}^*(T') \tag{5.17}$$

As for the terminal action, Lemma 20 yields:

$$\widehat{\mathcal{V}}\left(\overline{T}\right) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} \mathcal{V}^*\left(\overline{T}\right) \tag{5.18}$$

Now it suffices to show that for any action $a \in \mathcal{A}(T)$ with $T \xrightarrow{a} T'$ we have:

$$v(T') \xrightarrow[v(T)\to\infty]{\text{a.s}} \infty$$

Proof by contraction: As a matter of convenience, let us denote $\mathcal{A}\left(a_1,\dots,a_{|\mathcal{A}(T)|}\right)$ and:

$$\forall u\in\{1,\dots,|\mathcal{A}(T)|\}:T\xrightarrow{a_u}T_u$$

Without loss of generality, suppose that T_1 is visited a finite number of times as $v(T)\to\infty$. This is equivalently formulated with:

$$\exists N>0,\forall v(T)>0:v(T_1)\leq N$$

Let $v(T)>|\mathcal{A}(T)|-1+N$ be some large number. Since $v(T_1)\leq N$, then we have:

$$\exists 2\leq j\leq|\mathcal{A}(T)|:v(T_j)\geq\frac{v(T)-N}{|\mathcal{A}(T)|-1}$$

This comes from the fact that $v(T)=\sum_{i=1}^{|\mathcal{A}(T)|}v(T_i)$. Therefore, $\exists\frac{v(T)-N}{|\mathcal{A}(T)|-1}\leq v\leq v(T)$ such that T_j has been chosen, for the last time, at the v^{th} visit of T. Thus, at the v^{th} visit of T, we necessarily have:

$$\begin{aligned}&\widehat{\mathcal{Q}}(T,a_1)+C\sqrt{\frac{\log v}{v'(T_1)}}\leq\widehat{\mathcal{Q}}(T,a_j)+C\sqrt{\frac{\log v}{v'(T_j)}}\\\implies&-\lambda\mathbb{1}\{T_1\neq\overline{T}\}+\widehat{\mathcal{V}}(T_1)+C\sqrt{\frac{\log v}{v'(T_1)}}\leq-\lambda\mathbb{1}\{T_j\neq\overline{T}\}+\widehat{\mathcal{V}}(T_j)+C\sqrt{\frac{\log v}{v'(T_j)}}\end{aligned}$$

Where $v'(T_1)$ and $v'(T_j)$ are the number of visits of T_1 and T_j, respectively, at the time of the v^{th} visit of T. We have $v'(T_1)\leq v(T_1)\leq N$ and $v'(T_j)=v(T_j)\geq\frac{v(T)-N}{|\mathcal{A}(T)|-1}$ (because at the v^{th} visit of T, T_j was chosen for the last time). This yields:

$$-\lambda\mathbb{1}\{T_1\neq\overline{T}\}+\widehat{\mathcal{V}}(T_1)+C\sqrt{\frac{\log v}{N}}\leq-\lambda\mathbb{1}\{T_j\neq\overline{T}\}+\widehat{\mathcal{V}}(T_j)+C\sqrt{\frac{(|\mathcal{A}(T)|-1)\log v}{v(T)-N}}\tag{5.19}$$

We recall that $\frac{v(T)-N}{|\mathcal{A}(T)|-1}\leq v\leq v(T)$. Therefore we get:

$$-\lambda\mathbb{1}\{T_1\neq\overline{T}\}+\widehat{\mathcal{V}}(T_1)+C\sqrt{\frac{\log v}{N}}\xrightarrow[v(T)\to\infty]{\text{a.s}}\infty\tag{5.20}$$

On the other hand, since $v(T_j)\geq\frac{v(T)-N}{|\mathcal{A}(T)|-1}$ then $v(T_j)\xrightarrow[v(T)\to\infty]{\text{a.s}}\infty$ and hence $\widehat{\mathcal{V}}(T_j)\xrightarrow[v(T)\to\infty]{\text{a.s}}\mathcal{V}^*(T_j)$, and we deduce that:

$$-\lambda\mathbb{1}\{T_j\neq\overline{T}\}+\widehat{\mathcal{V}}(T_j)+C\sqrt{\frac{(|\mathcal{A}(T)|-1)\log v}{v(T)-N}}\xrightarrow[v(T)\to\infty]{\text{a.s}}-\lambda\mathbb{1}\{T_j\neq\overline{T}\}+\mathcal{V}^*(T_j)\tag{5.21}$$

Eq. (5.20) and Eq. (5.21) show that Eq. (5.19) does not hold when $v(T)\to\infty$, which yields our contradiction. Therefore we deduce that:

$$\forall u\in\{1,\dots,|\mathcal{A}(T)|\}:v(T_u)\xrightarrow[v(T)\to\infty]{\text{a.s}}\infty$$

Now going back to Eq. (5.17) and Eq. (5.18), we get that:

$$\begin{aligned}&\forall u\in\{1,\dots,|\mathcal{A}(T)|\}:\widehat{\mathcal{V}}(T_u)\xrightarrow[v(T)\to\infty]{\text{a.s}}\mathcal{V}^*(T_u)\\\implies&\forall u\in\{1,\dots,|\mathcal{A}(T)|\}:\widehat{\mathcal{Q}}(T,a_u)\xrightarrow[v(T)\to\infty]{\text{a.s}}\mathcal{Q}^*(T,a_u)\\\implies&\widehat{\mathcal{V}}(T)\xrightarrow[v(T)\to\infty]{\text{a.s}}\mathcal{V}^*(T)\end{aligned}$$

This concludes our proof. ■

Proposition 17. *For any state T, UCDT satisfies the optimal convergence property:*

$$\widehat{\mathcal{V}}(T) \xrightarrow[M\to\infty]{a.s} \mathcal{V}^*(T), \ \widehat{\pi}(T) \xrightarrow[M\to\infty]{a.s} \pi^*(T)$$

Proof The Theorem now follows straightforwardly as a Corollary from Lemma 21. Indeed, the number of iterations M is also the number of visits of the root state Ω. Lemma 21 implies that for any state T:

$$\begin{aligned} v(T) &\xrightarrow[M\to\infty]{\text{a.s}} \infty \\ \forall a \in \mathcal{A}(T) : \widehat{\mathcal{Q}}(T,a) &\xrightarrow[M\to\infty]{\text{a.s}} \mathcal{Q}^*(T,a) \end{aligned}$$

On the other hand:

$$\begin{aligned} \widehat{\pi}(T) &= \text{Argmax}_{a\in\mathcal{A}(T)} \widehat{\mathcal{Q}}(T,a) \\ &\xrightarrow[M\to\infty]{\text{a.s}} \text{Argmax}_{T'\in\mathcal{A}(T)} \mathcal{Q}(T,a) \\ &\xrightarrow[M\to\infty]{\text{a.s}} \pi^*(T) \end{aligned}$$

■

Proposition 18. *For any state T, EGDT satisfies the optimal convergence property:*

$$\widehat{\mathcal{V}}(T) \xrightarrow[M\to\infty]{a.s} \mathcal{V}^*(T), \ \widehat{\pi}(T) \xrightarrow[M\to\infty]{a.s} \pi^*(T)$$

Proof The proof of Proposition 18 follows the same steps as the proof of Proposition 17. The only result we should ensure for our inductive reasoning to work is that for all actions $a \in \mathcal{A}(T)$ with $T \xrightarrow{a} T'$ we have:

$$v(T') \xrightarrow[v(T)\to\infty]{\text{a.s}} \infty$$

To do so, we shall prove that

$$\mathbb{P}\left[\exists \tau > 0, \forall v \geq \tau : T' \text{ is not chosen at the } v^{th} \text{ visit of } T\right] = 0$$

We know that, at the v^{th} visit of T, the probability of choosing T' is larger than $\frac{1}{v^\gamma |\mathcal{A}(T)|}$. Therefore:

$$\mathbb{P}\left[\exists \tau > 0, \forall v \geq \tau : T' \text{ is not chosen at the } v^{\text{th}} \text{ visit of } T\right] \leq \sum_{\tau>0} \prod_{v\geq\tau} \left(1 - \frac{1}{v^\gamma |\mathcal{A}(T)|}\right)$$

Do we have $\forall \tau > 0 : \prod_{v\geq\tau} \left(1 - \frac{1}{v^\gamma |\mathcal{A}(T)|}\right) = 0$?

To verify this, we investigate whether, for any $\tau > 0$, the series $\sum_{v\geq\tau} \log\left(1 - \frac{1}{v^\gamma |\mathcal{A}(T)|}\right)$ diverges

to $-\infty$. We have the following:

$$\begin{aligned}\log\left(1-\frac{1}{v^{\gamma}|\mathcal{A}\left(T\right)|}\right) &= \log\left(\frac{v^{\gamma}-\frac{1}{|\mathcal{A}(T)|}}{v^{\gamma}}\right)\\ &= \log\left(v^{\gamma}-\frac{1}{|\mathcal{A}\left(T\right)|}\right)-\log\left(v^{\gamma}\right)\\ &= \int_{v^{\gamma}}^{v^{\gamma}-\frac{1}{|\mathcal{A}(T)|}}\frac{du}{u}\\ &\leq -\frac{1}{v^{\gamma}}\end{aligned}$$

Since the series $\sum_{v\geq\tau}\frac{1}{v^{\gamma}}$ diverges for any $\tau>0$ (because $0<\gamma<1$), we deduce that $\sum_{v\geq\tau}\log\left(1-\frac{1}{v^{\gamma}|\mathrm{Ch}(T)|}\right)=-\infty$ and therefore $\forall\tau>0:\prod_{v\geq\tau}\left(1-\frac{1}{v^{\gamma}|\mathrm{Ch}(T)|}\right)=0$ This yields:

$$\mathbb{P}\left[\exists\tau>0,\forall v\geq\tau:T'\text{ is not chosen at the }v^{th}\text{ visit of }T\right]=0$$

Which concludes our proof. ■

5.8 Appendix: Pseudo-code

Algorithm 2 UCDT and EGDT

Input: M number of iterations, m number of observed samples per Simulation, $\lambda \geq 0$.
Initialise $\pi(T) = \overline{a}$ and the number of visits $v(T) = 1$ for all states.
for $t = 1$ to M **do**
  Unroll $\pi : \Omega = T_0 \xrightarrow{\pi(T_0)} T_1 \xrightarrow{\pi(T_1)} \dots \xrightarrow{\pi(T_{L-1})} T_L = \overline{T_{L-1}}$ ▷ Selection
  if $v(T_{L-1}) \geq 2$ **then**
    Observe new m samples in T_L ▷ Simulation
    Update $\widehat{\mathcal{Q}}(T_{L-1}, \overline{a}) = \widehat{\mathcal{V}}(T_L)$ as per (5.6)
    Update the number of visits of T_L with $v(T_L) = v(T_L) + 1$
  else
    Observe m samples in T_{L-1} ▷ Simulation
    Update $\widehat{\mathcal{Q}}(T_{L-1}, \overline{a}) = \widehat{\mathcal{V}}(T_L)$ as per (5.6)
    for $a \in \mathcal{A}(T_{L-1}) \setminus \{\overline{a}\}$ **do** ▷ Expansion
      Set $T_{L-1} \xrightarrow{a} T$
      Initialise $\widehat{\mathcal{V}}(T)$ with (5.8)
      Update $\widehat{\mathcal{Q}}(T_{L-1}, a) = -\lambda + \widehat{\mathcal{V}}(T')$ as per (5.7)
    end for
  end if
  Update $\widehat{\mathcal{V}}(T_{L-1}) = \max_{a \in \mathcal{A}(T_{L-1})} \widehat{\mathcal{Q}}(T_{L-1}, a)$ as per (5.7)
  Update the policy value $\pi(T_{L-1})$ as per (5.9) or (5.10)
  Update the number of visits of T_{L-1} with $v(T_{L-1}) = v(T_{L-1}) + 1$
  for $j = L - 1$ to 1 **do** ▷ Backpropagation
    Update $\widehat{\mathcal{Q}}(T_{j-1}, \pi(T_{j-1})) = -\lambda + \widehat{\mathcal{V}}(T_j)$ as per (5.7)
    Update $\widehat{\mathcal{V}}(T_{j-1}) = \max_{a \in \mathcal{A}(T_{j-1})} \widehat{\mathcal{Q}}(T_{j-1}, a)$ as per (5.7)
    Update the policy value $\pi(T_{j-1})$ as per (5.9) or (5.10)
    Update the number of visits of T_{j-1} with $v(T_{j-1}) = v(T_{j-1}) + 1$
  end for
end for
Define the solution $\widehat{\pi}(T) = \text{Argmax}_{a \in \mathcal{A}(T)} \widehat{\mathcal{Q}}(T, a)$ for all states T
Unroll $\widehat{\pi} : \Omega = T_0 \xrightarrow{\pi(T_0)} T_1 \xrightarrow{\pi(T_1)} \dots \xrightarrow{\pi(T_{L-1})} T_L = \overline{T_{L-1}}$
return T_{L-1}

Chapter 6

Online Learning of Decision Trees with Thompson Sampling

Contents

6.1 Introduction . . . 153
6.2 The Algorithm: TSDT . . . 153
6.3 Experiments . . . 160
6.4 Conclusion . . . 161
6.5 Appendix: Table of Notation . . . 164
6.6 Appendix: Proofs . . . 165
6.7 Appendix: Pseudo-code . . . 173

6.1 Introduction

In Chapter 5, we introduced UCDT and EGDT, two MCTS algorithms that successfully find optimal sparse DTs given data streams. This encourages us to explore other policies that have been substantially studied in the literature, as they might induce better theoretical or empirical results. The policy we consider in this work is Thompson Sampling (Thompson, 1933).

Thompson Sampling (TS) is fundamentally different from UCB and Epsilon-Greedy by virtue of being a Bayesian policy. It defines a prior distribution on the value of each action. This value can be the expected reward in a bandit context (Lattimore and Szepesvári, 2020), or the optimal expected return in a MDP context. TS chooses an action according to the probability of its prior being larger than the priors of the other actions, then upon observing a new sample of this action's reward, its prior is updated into a posterior distribution. TS has been substantially studied in the literature, mainly in the bandit setting, and proven to satisfy optimality guarantees (Agrawal and Goyal, 2012; Kaufmann et al., 2012; Agrawal and Goyal, 2013; Russo and Van Roy, 2016). In the context of MDPs however, TS has been rarely considered, but there are works such as (Bai et al., 2013, 2014). In (Bai et al., 2013, Section 3.5), the authors state:

We conclude that DNG-MCTS can find the optimal policy for the root node if unbounded computational resources are given.

where DNG-MCTS is the authors' algorithm. (Bai et al., 2013, Section 3.5) justifies the optimal convergence of MCTS based on a TS policy, but unfortunately no formal proof was provided. Nevertheless, we also believe this claim to be true, and we will substantiate it with a proof in our context.

In this chapter, we introduce TSDT, an MCTS algorithm seeking optimal sparse DTs given a data stream through the use of a TS policy. We show that TSDT retrieves the optimal sparse DT almost surely as we observe more and more data from the stream, and we compare it empirically with the state of the art from the River library. In similar fashion to UCDT and EGDT, TSDT also solves the hard challenges where the state of the art fail. However, we will see that TSDT can be even slower than UCDT and EGDT due to its posterior distribution updates. Nevertheless, it remains an interesting approach that could yield better performing algorithms in the future.

6.2 The Algorithm: TSDT

As is custom in this thesis now, we need to estimate the optimal state-action values $\mathcal{Q}^*(T, a) = \mathcal{Q}^{\pi^*}(T, a)$. However, instead of using frequentist point estimates as was the case for UCDT and EGDT, the Bayesian nature of TSDT rather necessitates the assignment of prior distributions on $\mathcal{Q}^*(T, a)$. These prior distributions are updated into posterior distributions as we observe more and more samples. Our objective now is to define these priors and posterior updates adequately. This replaces the first question we asked in Section 5.3; the second question however, remains the same.

6.2.1 Defining a Prior on $\mathcal{Q}^*(T, a)$

Let T be a state. Starting with the terminal action, we seek an adequate prior distribution on $\mathcal{Q}^*(T, \overline{a}) = \mathcal{H}(T)$. This time however, we consider incorporating Dynamic Programming in TSDT from the beginning, and we are aware of the Weights Degeneracy phenomenon that

could arise as a consequence. As we have discussed in Section 5.4, we consider writing $\mathcal{Q}^*(T, \overline{a})$ again in the following form:

$$\mathcal{Q}^*(T, \overline{a}) = \mathcal{H}(T) = \mathbb{P}[T(X) = Y] = \sum_{l \in T} \mathbb{P}[T(X) = Y | l(X) = 1] \mathbb{P}[l(X) = 1]$$

Let $\theta(T, \overline{a})$ denote a random variable that is distributed according to a prior on $\mathcal{Q}^*(T, \overline{a})$. We chose this notation to distinguish it from the point estimates $\widehat{\mathcal{Q}}(T, \overline{a})$ that we used in the context of UCDT and EGDT. The above expression of $\mathcal{Q}^*(T, \overline{a})$ suggests defining $\theta(T, \overline{a})$ as follows:

$$\theta(T, \overline{a}) = \sum_{l \in T} \widehat{p}(l) \theta(l) \tag{6.1}$$

where $\widehat{p}(l)$ are the point estimates of $\mathbb{P}[l(X) = 1]$ we defined in Eq. (5.14), and $\theta(l)$ are random variables distributed according to a prior distribution on $\mathbb{P}[T(X) = Y | l(X) = 1]$. In the following, we investigate ways of defining $\theta(l)$ and their consequences. We first notice that $\mathbb{P}[T(X) = Y | l(X) = 1]$ is the expectation of a Bernoulli variable:

$$\mathbb{P}[T(X) = Y | l(X) = 1] = \mathbb{E}[\mathbb{1}\{T(X) = Y\} | l(X) = 1]$$

Immediately, we are tempted to define $\theta(l)$ according to a Beta prior since it is conjugate to the Bernoulli likelihood. Accordingly, given $\{(X_t, Y_t)\}_{t=1}^n$ the accumulated observed samples during the simulations of all DTs that share l, $\theta(l)$ would be defined as follows:

$$\begin{aligned}
\theta(l) &\sim \text{Beta}(\alpha(l), \beta(l)) \\
\alpha(l) &= 1 + \sum_{t=1}^{n} l(X_t) \mathbb{1}\{T(X_t) = Y_t\} \\
\beta(l) &= 1 + \underbrace{\sum_{t=1}^{n} l(X_t)}_{n(l)} - \sum_{t=1}^{n} l(X_t) \mathbb{1}\{T(X_t) = Y_t\}
\end{aligned}$$

An immediate issue arises from these updates. For X_t in l, the value of the prediction $T(X_t) = k^*(l)$ is unknown. We solve this problem by estimating $T(X_t)$ itself and then plugging it back in the posterior updates. Notice that:

$$\begin{aligned}
\forall 1 \leq t \leq n : l(X_t) = 1 &\implies T(X_t) = k^*(l) \\
&\implies T(X_t) = \text{Argmax}_{1 \leq k \leq K} \mathbb{P}[Y = k | l(X) = 1]
\end{aligned}$$

This suggests estimating $T(X_t)$ (when $l(X_t) = 1$) with:

$$\widehat{T}(X_t) = \text{Argmax}_{1 \leq k \leq K} n_k(l) \tag{6.2}$$

Therefore resulting in:

$$\begin{aligned}
\theta(l) &\sim \text{Beta}(\alpha(l), \beta(l)) \\
\alpha(l) &= 1 + \sum_{t=1}^{n} l(X_t) \mathbb{1}\{\widehat{T}(X_t) = Y_t\} \\
\beta(l) &= 1 + n(l) - \sum_{t=1}^{n} l(X_t) \mathbb{1}\{\widehat{T}(X_t) = Y_t\}
\end{aligned}$$

This is a posterior update that we can certainly consider. However, let us imagine that we have just run a new simulation step of a DT containing l, and that we have at our disposal new data

$\{(X_t, Y_t)\}_{t=n+1}^{n+m}$ that we can accumulate with the previous data to update $\alpha(l)$ and $\beta(l)$. In this case, we would first update the estimated predictions $\widehat{T}(X_t)$ according to Eq. (6.2) by updating $n_k(l)$ as follows:

$$n_k(l) = \sum_{t=1}^{n+m} l(X_t) \mathbb{1}\{Y_t = k\}$$

The resulting new estimate $\widehat{T}(X_t)$ might be different from the older one and as such, when plugging $\widehat{T}(X_t)$ in the expressions of $\alpha(l)$ and $\beta(l)$, we would need to recalculate the entire sums to perform the update. This becomes more and more costly as we observe more and more data. Hence why we propose a different approach that is incremental and has a constant computational cost per update. To do this, we make slight changes to our notation:

$$\forall 1 \leq \tau \leq n : n_k^{(\tau)}(l) = \sum_{t=1}^{\tau} l(X_t) \mathbb{1}\{Y_t = k\}$$

$n_k^{(\tau)}(l)$ is the number of observed samples in l from the subset $\{(X_t, Y_t)\}_{t=1}^{\tau}$. In similar fashion, we use Eq. (6.2) to define the estimated prediction according to the subset $\{(X_t, Y_t)\}_{t=1}^{\tau}$:

$$\forall 1 \leq \tau \leq n : \widehat{T}^{(\tau)}(X_t) = \text{Argmax}_{1 \leq k \leq K} n_k^{(\tau)}(l) \tag{6.3}$$

With this new formulation, we define the posterior update as follows:

$$\theta(l) \sim \text{Beta}(\alpha(l), \beta(l)) \tag{6.4}$$

$$\alpha(l) = 1 + \sum_{t=1}^{n} l(X_t) \mathbb{1}\{\widehat{T}^{(t-1)}(X_t) = Y_t\} \tag{6.5}$$

$$\beta(l) = 1 + n(l) - \sum_{t=1}^{n} l(X_t) \mathbb{1}\{\widehat{T}^{(t-1)}(X_t) = Y_t\} \tag{6.6}$$

Now if we have new data $\{(X_t, Y_t)\}_{t=n+1}^{n+m}$ at our disposal, we can directly increment the values of $\alpha(l)$ and $\beta(l)$ with:

$$\theta(l) \sim \text{Beta}(\alpha(l), \beta(l))$$

$$\alpha(l) = \underbrace{1 + \sum_{t=1}^{n} l(X_t) \mathbb{1}\{\widehat{T}^{(t-1)}(X_t) = Y_t\}}_{\text{old value of } \alpha(l)} + \sum_{t=n+1}^{n+m} l(X_t) \mathbb{1}\{\widehat{T}^{(t-1)}(X_t) = Y_t\}$$

$$\beta(l) = \underbrace{1 + n(l) - \sum_{t=1}^{n} l(X_t) \mathbb{1}\{\widehat{T}^{(t-1)}(X_t) = Y_t\}}_{\text{old value of } \beta(l)} - \sum_{t=n+1}^{n+m} l(X_t) \mathbb{1}\{\widehat{T}^{(t-1)}(X_t) = Y_t\}$$

Before moving to the next steps of our posterior update, there are few important points we should note.

- $\widehat{T}^{(0)}$ is not defined by Eq. (6.3), it is an arbitrary initialisation of the prediction estimate.
- We use the terms $\widehat{T}^{(t-1)}$ in Eq. (6.5) and Eq. (6.6) instead of $\widehat{T}^{(t)}$. The reason behind this choice is that $\widehat{T}^{(t)}$ is defined in Eq. (6.3) partly based on (X_t, Y_t). Thus $\mathbb{1}\{\widehat{T}^{(t)}(X_t) = Y_t\}$ is skewed, biased and not even guaranteed to be consistent. On the other hand, only $\{(X_{t'}, Y_{t'})\}_{t'=1}^{t-1}$ are involved in the expression of $\widehat{T}^{(t-1)}(X_t)$, and since the data is i.i.d., an assumption we made when formulating the online problem in Section 2.1, $\widehat{T}^{(t-1)}(X_t)$ is also independent of (X_t, Y_t), thus yielding a better behaved estimator that is consistent as we shall prove later.

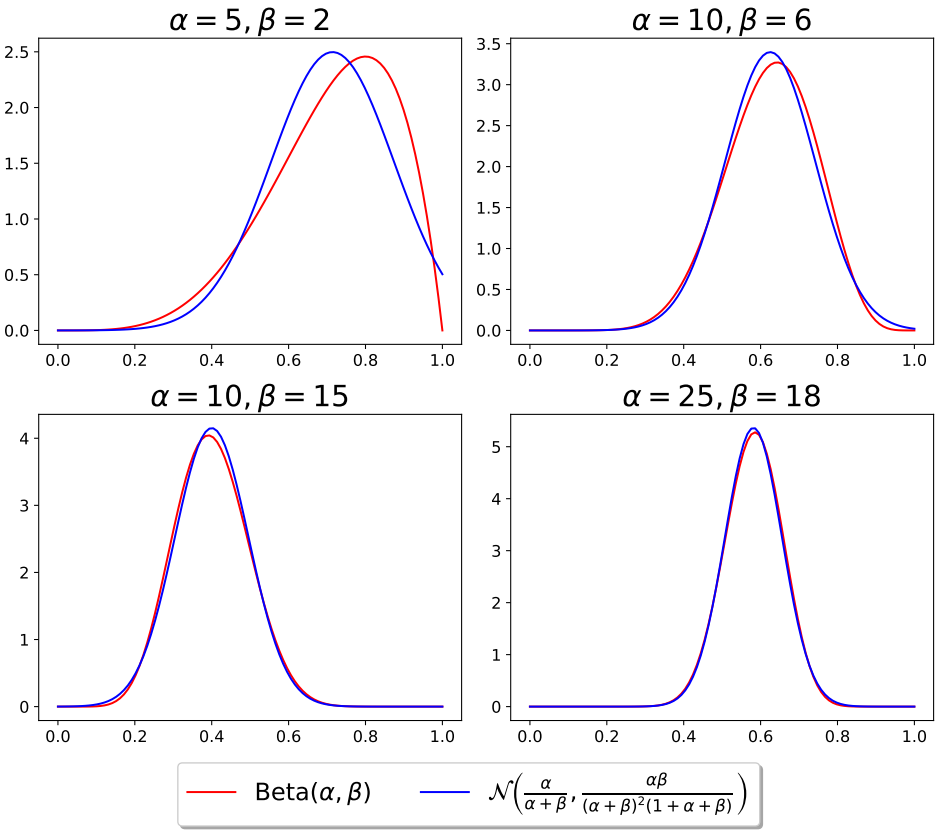


Figure 6.1: Probability density functions of a Beta distribution and its Normal approximation for different parameters α and β.

- One might think that since we do not update the prediction estimate in all the terms of the sums involved in Eq. (6.5) and Eq. (6.6), that this would cause issues for the posterior distribution of $\theta(l)$. As we will see shortly, $\theta(l)$ still concentrates around the true values they estimate $\mathbb{P}[T(X) = Y | l(X) = 1]$ almost surely as the number of observed samples n grows to infinity. However, this scheme of updates described in Eq. (6.5) and Eq. (6.6) might indeed slow down this concentration, but this is an analysis that we have not performed, as in similar fashion to Chapter 5, we focused on the asymptotic convergence guarantees and we did not analyse the rates of convergence.

There is still one more issue behind our posterior updates. We recall that defining $\theta(l)$ is not our end goal but rather defining $\theta(T, \overline{a})$. Therefore, let us use Eq. (6.4) in Eq. (6.1) and ask: **what is the distribution of $\boldsymbol{\theta(T, \overline{a})}$**? Answering this question is not straightforward, Eq. (6.1) describes $\theta(T, \overline{a})$ as a linear combination of the Beta distributed variables $\theta(l)$, hence its distribution is not easy to infer. To solve this problem, we employ a common approximation of Beta variables with Normal variables that share the same mean and variance. We redefine our posterior updates as follows:

$$\begin{cases} \theta(l) & \sim \mathcal{N}\left(\mu(l), \sigma(l)^2\right) \\ \mu(l) & = \frac{\alpha(l)}{\alpha(l)+\beta(l)} \\ \sigma(l)^2 & = \frac{\alpha(l)\beta(l)}{(\alpha(l)+\beta(l))^2(1+\alpha(l)+\beta(l))} \end{cases} \tag{6.7}$$

$\alpha(l)$ and $\beta(l)$ are still defined according to Eq. (6.5) and Eq. (6.6) respectively. $\mu(l)$ and $\sigma(l)^2$ are the mean and variance of Beta$(\alpha(l), \beta(l))$ respectively. Fig. 6.1 illustrates that the

probability density functions of a Beta distribution and its Normal approximation get closer to each other quickly as the α and β parameters grow. This means that, as we observe more and more samples, the Normal approximation Eq. (6.7) becomes quickly a very good approximation of Eq. (6.4). Now $\theta(T, \overline{a})$, in Eq. (6.1), is a linear combination of Normal variables, thus yielding:

$$\begin{cases} \theta(T, \overline{a}) & \sim \mathcal{N}\left(\mu(T, \overline{a}), \sigma(T, \overline{a})^2\right) \\ \mu(T, \overline{a}) & = \sum_{l \in T} \widehat{p}(l)\, \mu(l) \\ \sigma(T, \overline{a})^2 & = \sum_{l \in T} \widehat{p}(l)^2\, \sigma(l)^2 \end{cases} \tag{6.8}$$

This finally concludes the definition of the prior $\theta(T, \overline{a})$ on $\mathcal{Q}^*(T, \overline{a})$. The next step is to define the prior and posterior updates on $\mathcal{Q}^*(T, a)$ for split actions $a \in \mathcal{A}(T) \setminus \{\overline{a}\}$. Before moving to this step, let us make an important remark about the Normal prior in Eq. (6.8). Note that, even though $\theta(T, \overline{a})$ is a linear combination of the Normally distributed variables $\theta(l)$ according to Eq. (6.1), the coefficients $\widehat{p}(l)$ are not constant, they are themselves random variables estimating $\mathbb{P}[l(X) = 1]$, **then how did we still infer a Normal distribution for $\boldsymbol{\theta(T, \overline{a})}$**? The answer is that the priors we define are conditioned on the observed data. If $\{(X_t, Y_t)\}_{t=1}^n$ is the set of accumulated observed samples in all the branches $l \in T$, then the prior defined in Eq. (6.8) is conditioned on $\{(X_t, Y_t)\}_{t=1}^n$. Moreover, the estimates $\widehat{p}(l)$ are completely determined by $\{(X_t, Y_t)\}_{t=1}^n$, which means that, conditionally on $\{(X_t, Y_t)\}_{t=1}^n$, the coefficients $\widehat{p}(l)$ in Eq. (6.1) are constants, thus allowing the inference of the conditional Normal distribution in Eq. (6.8).

We turn our attention now to defining the prior on $\mathcal{Q}^*(T, a)$ for a split action $a \in \mathcal{A}(T) \setminus \{\overline{a}\}$. In similar fashion to Section 5.3.1, let $T \xrightarrow{a} T'$ and consider the Bellman Equations:

$$\begin{cases} \mathcal{Q}^*(T, a) & = -\lambda + \mathcal{V}^*(T') \\ \mathcal{V}^*(T') & = \max_{a' \in \mathcal{A}(T')} \mathcal{Q}^*(T', a') \end{cases}$$

It is thus natural to define our priors as follows:

$$\begin{cases} \theta(T, a) & = -\lambda + \theta(T') \\ \theta(T') & = \max_{a' \in \mathcal{A}(T')} \theta(T', a') \end{cases} \tag{6.9}$$

where $\theta(T')$ is a prior on $\mathcal{V}^*(T')$, and since $\mathcal{V}^*(T') = \mathcal{Q}^*(T', \overline{a})$, we set $\theta(T') = \theta(T', \overline{a})$. Moreover, if T' has never been visited, we initialise $\theta(T', a')$ to have the same distribution as $\theta(T', \overline{a})$ for all actions $a' \in \mathcal{A}(T')$. Thompson Sampling is a stochastic policy that chooses an action according to its probability of maximising the prior, i.e.

$$\forall a \in \mathcal{A}(T) : \pi(a|T) = \mathbb{P}\left[\forall a' \in \mathcal{A}(T) : \theta(T, a) \geq \theta(T, a')\right] \tag{6.10}$$

where $\pi(a|T)$ denotes the probability of taking action a at state T according to the stochastic policy π. In practice, in order to use Thompson Sampling, we first sample all the random variables $\theta(T, a)$ for all $a \in \mathcal{A}(T)$, then we take the action with maximum sample:

$$\text{Argmax}_{a \in \mathcal{A}(T)} \theta(T, a)$$

Therefore, going back to the update Eq. (6.9), we need to know the distribution of $\theta(T, a)$ and we need to ensure that we can sample from this distribution. Consider the term:

$$\theta(T') = \max_{a' \in \mathcal{A}(T')} \theta(T', a') \tag{6.11}$$

and suppose that $\forall a' \in \mathcal{A}(T') : \theta(T', a') \sim \mathcal{N}\left(\mu(T', a'), \sigma(T', a')^2\right)$. This supposition is motivated by an inductive reasoning, since the priors are all initially Normal, then if we show that

backpropagating Normal distributions yields a Normal distribution, then we would deduce that all priors are Normal at all times and we will retrieve their posterior update rules. On account of this: **Is $\theta(T')$ Normally distributed?**

Unfortunately, the maximum of Normal variables is not Normal, as documented in (Clark, 1961; Sinha et al., 2007). To solve this issue, a first approach is to use a Normal approximation of the distribution of the maximum in Eq. (6.11), this is achieved by recursively applying Clark's mean and variance formula for the maximum of two Normal variables. Indeed, consider the case with two independent Gaussians $\theta_1 \sim \mathcal{N}(\mu_1, \sigma_1^2), \theta_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$. Let $\theta = \max\{\theta_1, \theta_2\}$, then the mean μ and variance σ^2 of θ satisfy:

$$\mu = \mu_1 \Phi(\alpha) + \mu_2 \Phi(-\alpha) + \phi(\alpha)\sigma_m \tag{6.12}$$

$$\sigma^2 = \left(\mu_1^2 + \sigma_1^2\right)\Phi(\alpha) + \left(\mu_2^2 + \sigma_2^2\right)\Phi(-\alpha) + (\mu_1 + \mu_2)\sigma_m \phi(\alpha) - \mu^2 \tag{6.13}$$

where $\sigma_m = \sigma_1^2 + \sigma_2^2, \alpha = \frac{\mu_1 - \mu_2}{\sigma_m}$, and ϕ and Φ are respectively the probability density function and the cumulative distribution function of $\mathcal{N}(0,1)$, see (Clark, 1961; Tesauro et al., 2012). In similar fashion to our Normal approximation of a Beta distribution, the distribution of θ can be approximated with a Normal distribution by matching their first two moments, for short we call it Clark's approximation. This motivates rewriting $\theta(T')$ as nested pair-wise maximums:

$$\theta(T') = \max\left\{\theta(T', a_1), \max\left\{\theta(T', a_2), \ldots, \max\left\{\theta\left(T', a_{|\mathcal{A}(T')|-1}\right), \theta\left(T', a_{|\mathcal{A}(T')|}\right)\right\}\right\}\right\} \tag{6.14}$$

where we wrote $\mathcal{A}(T') = \{a_1, \ldots, a_{|\mathcal{A}(T')|}\}$ for convenience. Starting with the right-most term $\max\left\{\theta\left(T', a_{|\mathcal{A}(T')|-1}\right), \theta\left(T', a_{|\mathcal{A}(T')|}\right)\right\}$, we approximate its distribution with Clark's approximation, then we recursively apply Clark's approximation in similar fashion to the induced pair-wise maximums of Normal variables until we get an overall Normal approximation of $\theta(T')$. Such approximation scheme was used by Tesauro et al. (2012) in the context of MCTS with a Bayesian approach, but the policy the authors considered is not Thompson Sampling but rather a form of UCB employing the quantiles of the posterior distributions for the UCB index. Nevertheless, this approximation incurs a substantial computational cost as the number of actions $|\mathcal{A}(T')|$ increases. Moreover, as outlined by Sinha et al. (2007), the order in which the recursive approximations are executed may significantly affect the quality of the overall approximation. For these reasons, we introduce an alternative, more straightforward approach to Backpropagating the posterior distributions $\theta(T', a')$ to $\theta(T')$. Specifically, we assign to $\theta(T')$ the posterior distribution of the action with maximum posterior mean, which also yields a Normal prior for $\theta(T, a)$ according to Eq. (6.9):

$$\begin{cases} a^* & = \text{Argmax}_{a' \in \mathcal{A}(T')} \mu(T', a') \\ \mu(T') & = \mu(T', a^*) \\ \sigma(T') & = \sigma(T', a^*) \\ \theta(T') & \sim \mathcal{N}\left(\mu(T'), \sigma(T')^2\right) \\ \theta(T, a) & \sim \mathcal{N}\left(-\lambda + \mu(T'), \sigma(T')^2\right) \end{cases} \tag{6.15}$$

The posterior update Eq. (6.15) exhibits better computational efficiency than the scheme with the recursive applications of Clark's approximation. While in general, the prior of $\theta(T, a)$ may

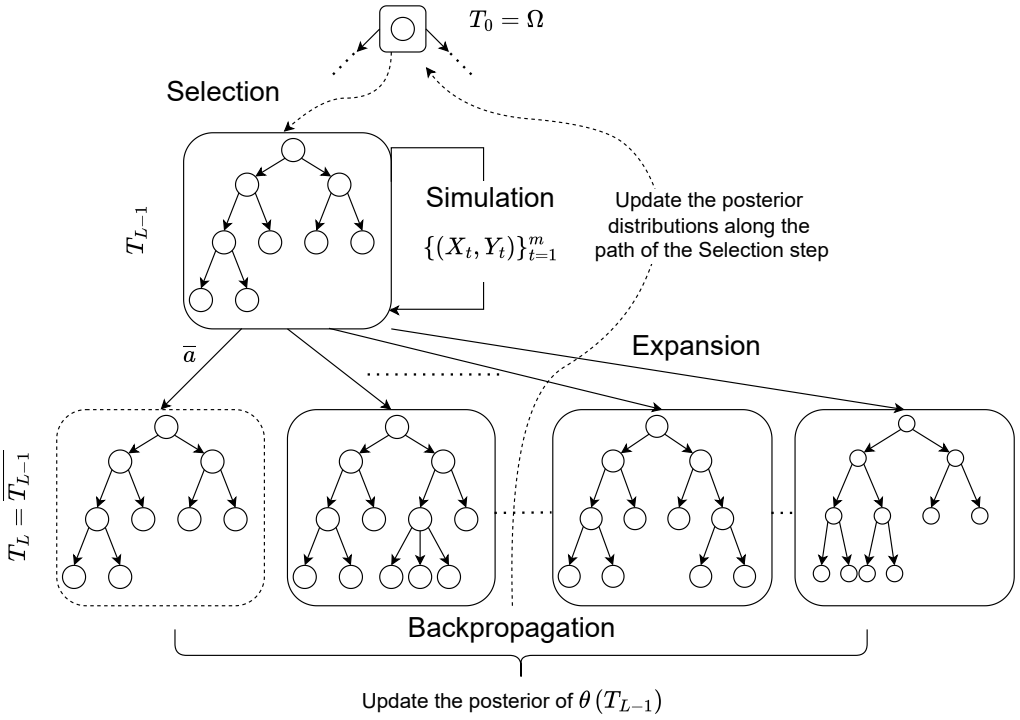


Figure 6.2: Algorithm 3 from the perspective of the Search Tree. The left-most child is drawn in dashed lines to emphasize that it is an absorbing state since it results from taking the terminal action at T_{L-1}.

not serve as an accurate approximation for the distribution of the maximum in Eq. (6.9), it progressively improves as $\theta(T', a')$, for actions $a' \in \mathcal{A}(T')$, concentrate around their means. This concentration occurs as more data is accumulated within T'. We have implemented both update schemes and we noticed in practice that Eq. (6.15) displays a substantial gain in computational efficiency and also in performance compared to Clark's approximation.

In practice, the posterior distributions $\theta(T) \sim \mathcal{N}\left(\mu(T), \sigma(T)^2\right)$ could concentrate very fast around some value, thus hindering exploration. For this reason, we introduce an exponent $0 < \gamma \leq 1$ to the variances. Its effect is to widen the posterior distribution around $\mu(T)$ and slow the collapse of it variance, thus allowing for a management between Exploration and Exploitation. This γ hyperparameter plays a similar role to the γ we introduced in Chapter 5 in the context of UCDT and TSDT.

6.2.2 The Search Strategy

TSDT's search strategy is similar to that of UCDT and EGDT. The difference pertains to using a Thompson Sampling Tree policy that is based on updating posterior distributions on the optimal state-action values. TSDT's algorithmic description is outlined in Algorithm 3, it follows the same steps (Selection, Simulation, Expansion and Backpropagation) as the ones we described in Section 5.3.2, except that instead of updating point estimates during Expansion and Backpropagation, we update posterior distributions. Fig. 6.2 is a visual representation of Algorithm 3 from the Search Tree perspective. At the end of the M iterations, we propose a

Table 6.1: Comparing TSDT with VFDT, HAT, EFDT and SGT on several datastreams with categorical features from River.

Stream	VFDT		EFDT		SGT		HAT		TSDT	
	accuracy	splits	accuracy	splits	accuracy	splits	accuracy	splits	accuracy	splits
led	**1**	**9**	0.94	8	0.14	6	**1**	**9**	**1**	**9**
led-irrelevant	**1**	**9**	**1**	**9**	0.69	10	**1**	**9**	**1**	**9**
random-tree-1	**1**	35	0.93	27	0.13	598	**1**	35	0.960	23
random-tree-2	0.89	75	0.95	117	0.77	10	0.91	87	0.850	12
random-tree-3	0.84	115	0.85	115	0.65	20	0.79	5	0.740	4
stagger	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**
xor-1	**1**	**3**	0.45	7	0.57	21	0.61	64	**1**	**3**
xor-2	0.45	67	0.8	5	0.48	31	0.54	65	**1**	**3**
xor-3	0.50	61	0.55	3	0.53	32	0.51	63	**1**	**3**
xor-4	0.54	31	0.49	3	0.51	16	0.45	31	0.440	3

policy $\widehat{\pi}$ according to the means of the posterior distributions $\mu(T, a)$:

$$\forall T : \widehat{\pi}(T) = \text{Argmax}_{a \in \mathcal{A}(T)} \mu(T, a)$$

Then we unroll $\widehat{\pi} : \Omega = T_0 \xrightarrow{\pi(T_0)} T_1 \xrightarrow{\pi(T_1)} \dots \xrightarrow{\pi(T_{L-1})} T_L = \overline{T_{L-1}}$, in which case $T_{L-1} = T^{\widehat{\pi}}$ is our proposed solution. The reason we base this solution on the means is that we expect the means $\mu(T, a)$ to have concentrated around the true optimal values $\mathcal{Q}^*(T, a)$ when the algorithm has run for a large number of iterations M.

Proposition 22 proves the concentration of $\theta(T) \sim \mathcal{N}\left(\mu(T), \sigma(T)^2\right)$ around the optimal values $\mathcal{V}^*(T)$. By virtue of the Bellman equations, this also leads to the concentration of the posteriors $\theta(T, a)$ around $\mathcal{Q}^*(T, a)$, thus resulting in the optimal convergence of $\widehat{\pi}(T)$ to $\pi^*(T)$ for all states T.

Proposition 22. *For any state T, TSDT satisfies the optimal convergence property:*

$$\mu(T) \xrightarrow[M \to \infty]{\text{a.s}} \mathcal{V}^*(T), \ \sigma(T)^2 \xrightarrow[M \to \infty]{\text{a.s}} 0, \ \widehat{\pi}(T) \xrightarrow[M \to \infty]{a.s} \pi^*(T)$$

6.3 Experiments

We use the same experimental setup introduced in Section 5.5 and Table 5.1. At the end of Section 6.2.1, we introduced the exponent γ to the variances of our Normal posterior distributions. The reason being to control the Exploration-Exploitation dilemma. Without this exponent, we noticed that the variances collapsed to 0 too quickly, which hinders TSDT's exploration capacity. Theoretically the algorithm will still converge optimally in the limit, however, the rate of this convergence will be greatly slowed down without an adequate γ value. In similar fashion to Chapter 5, we did not tune this γ but rather set it heuristically to $\gamma = 0.8$ for all experiments because we noticed that this value strikes a good balance between Exploration and Exploitation.

Table 6.1, Fig. 6.3 and Fig. 6.4 reveal similar remarks regarding the performance of TSDT to those of UCDT and EGDT discussed in Section 5.5. There are however, few interesting points that pertain exclusively to TSDT. It outperforms UCDT and EGDT on Random-Trees 1. Similarly to EGDT but unlike UCDT, it is suboptimal on XOR 4. Moreover, TSDT reaches timeout with fewer samples than UCDT and EGDT, which indicates that it is a slower algorithm. This is expected because, during the Selection step, TSDT needs to sample from the posterior distributions at the level of each state in order to choose the action to take next. This is contrast with UCDT, which only updates the action estimates based on the new number of visits of the state, and even though EGDT samples from a categorical distribution according to its epsilon-greedy policy, this is less computationally costly than TSDT's sampling procedure. Furthermore, TSDT updates the prediction estimates $\widehat{T}^{(t)}(X)$ via Eq. (6.3) after each Simulation step, thus incurring an additional computation cost.

Overall, TSDT is slower than UCDT and EGDT, but performs similarly or even better in some cases such as Random-Trees 1. A finite-time analysis could shed light on the differences between these algorithms. Moreover, there might better Thompson Sampling formulations of TSDT that do not require updating the prediction estimates $\widehat{T}(X)$. This could yield a faster and better performing algorithm, we leave this idea for a future work.

6.4 Conclusion

This chapter is a natural follow-up to Chapter 5. We investigated the use of Thompson Sampling as a Tree policy for our MCTS approach. The resulting algorithm, called TSDT, satisfies an optimal asymptotic convergence guarantee, in similar fashion to UCDT and TSDT. However, from an empirical point of view, while TSDT achieves similar performance to UCDT and EGDT, thus outperforming the state of the art, it displays slower convergence and fails at retrieving the optimal DT for XOR 4 within the allocated 5 minutes.

At this stage, we would recommend the use of UCDT or EGDT over TSDT. TSDT has a higher computational cost due to the continuing estimations of the predictions $\widehat{T}(X)$. Furthermore, despite the soundness of the employed approximations, in the sense that they allow optimal asymptotic convergence, their impact on finite-time performance is unknown and could be detrimental. Nevertheless, TSDT's Bayesian approach could be refined in the future through better prior formulations and more efficient Backpropagation schemes, especially in conjunction with a new MDP that resembles more Branches' than DT-Miner's.

In the next and final chapter of this thesis, we consider an online version of Branches, our main contribution in Part I. We will see that, contrary to our online methods introduced so far, Online-Branches is amenable to a finite-time analysis. Moreover, it outperforms UCDT, EGDT and TSDT from an empirical stand point.

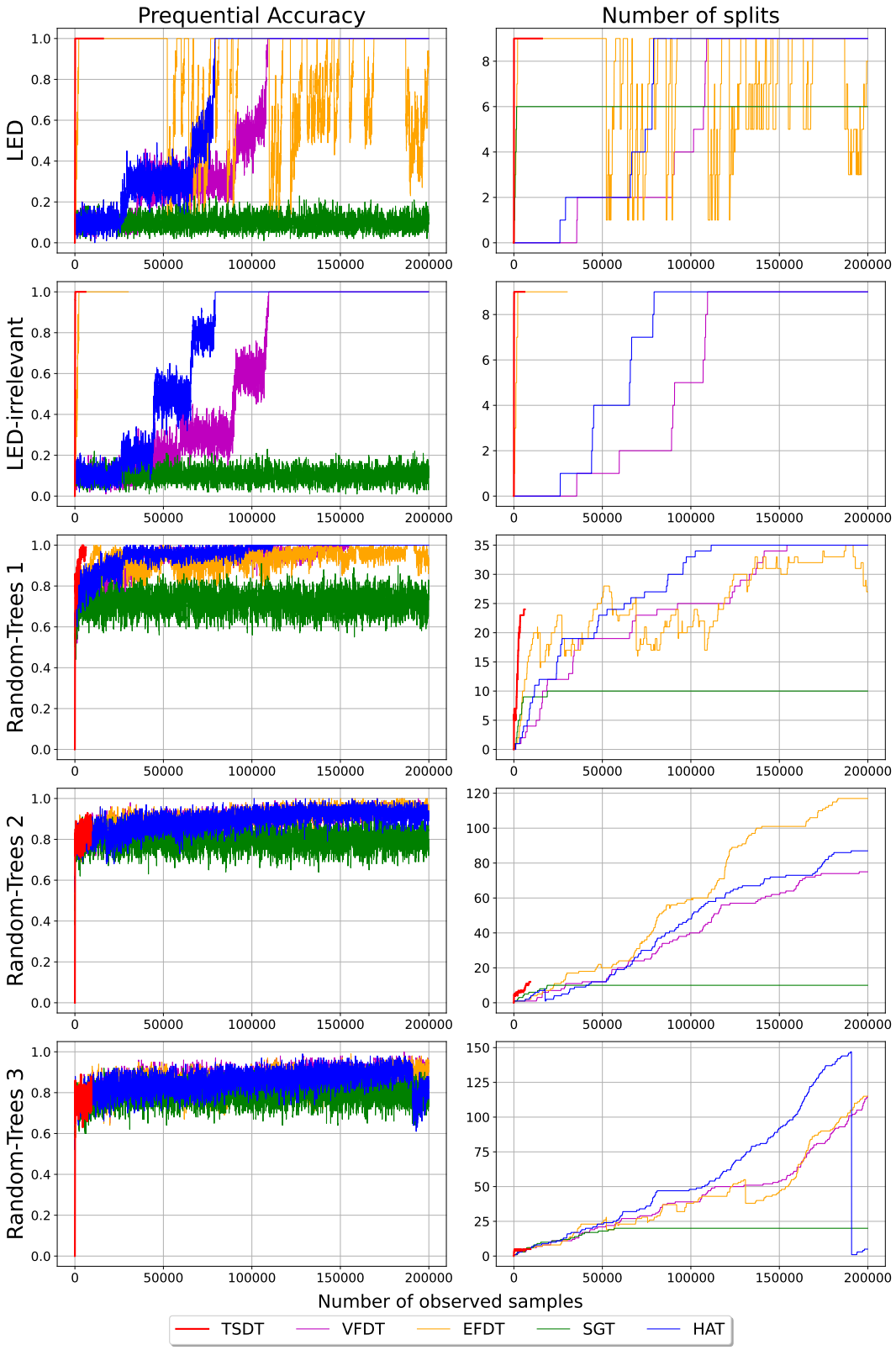


Figure 6.3: The left column reports the prequential accuracy of each method as a function of the number of observed samples so far. The right column reports the number of splits of the solution for each algorithm as a function of the number of observed samples so far.

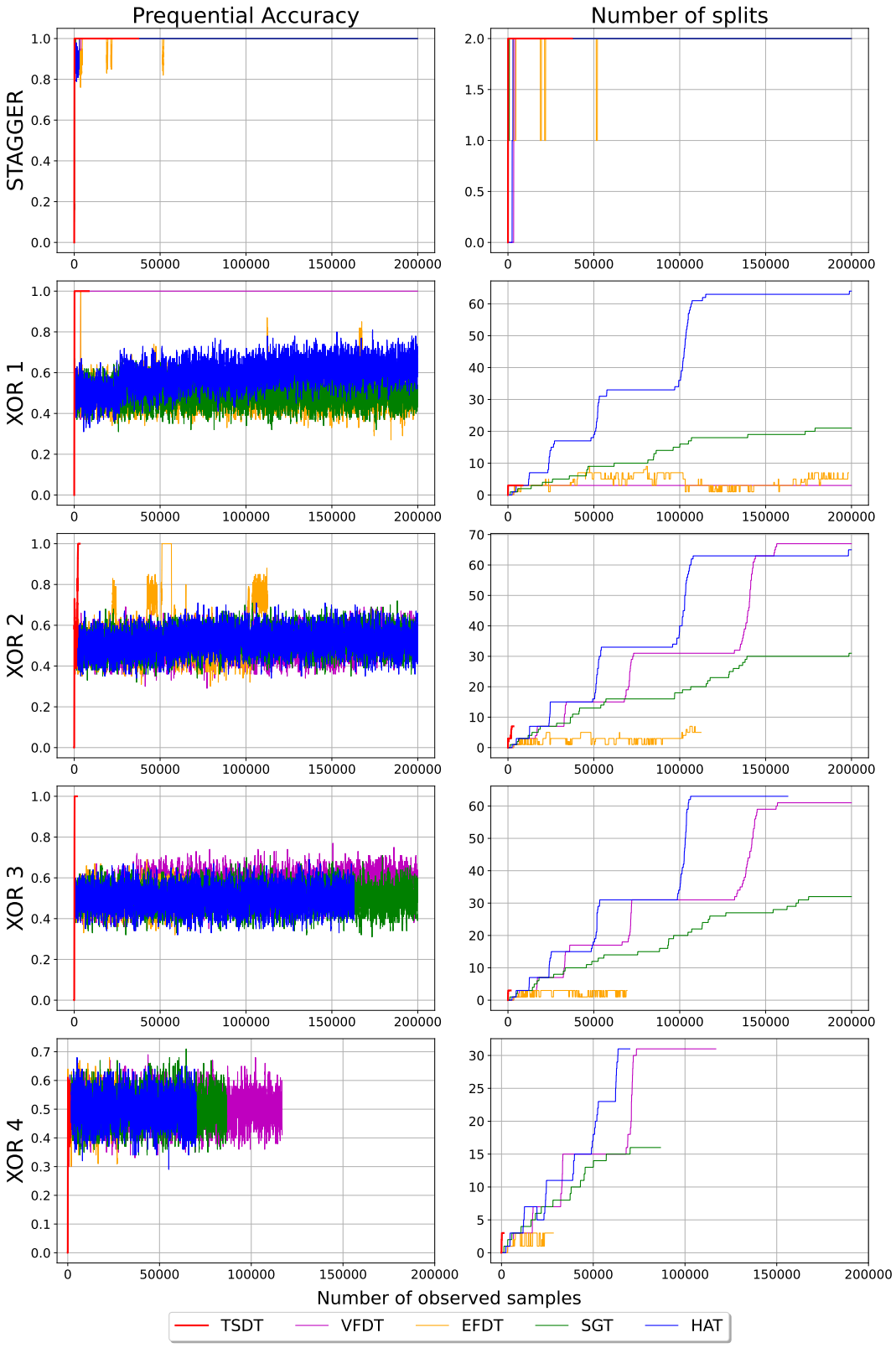


Figure 6.4: The remaining experimental results.

6.5 Appendix: Table of Notation

Table 6.2: Table of Notation

$n(l)$	$\triangleq$	Number of examples in l.
$n_k(l)$	$\triangleq$	Number of examples in l of class k.
$n_k^{(\tau)}(l)$	$\triangleq$	Number of examples in l of class k amongst data $\{(X_t, Y_t)\}_{t=1}^{\tau}$.
$k^*(l)$	$=$	$\text{Argmax}_{1\leq k\leq K}\{n_k(l)\}$, majority class in l.
$\mathbb{P}[l(X)=1]$	$\triangleq$	Empirical probability that X is in l.
$\widehat{p}(l)$	$\triangleq$	Point estimate of $\mathbb{P}[l(X)=1]$.
$\theta(l)$	$\triangleq$	Prior distribution on $\mathbb{P}[T(X)=Y\|l(X)=1]$.
$\theta(l)$	$\sim$	$\mathcal{N}\left(\mu(l), \sigma(l)^2\right)$.
$\theta(T,a)$	$\triangleq$	Prior distribution on $\mathcal{Q}^*(T,a)$.
$\widehat{T}(X)$	$=$	$\text{Argmax}_{1\leq k\leq K} n_k(l)$ estimate of $T(X)$.
$\widehat{T}^{(\tau)}(X)$	$=$	$\text{Argmax}_{1\leq k\leq K} n_k^{(\tau)}(l)$ estimate of $T(X)$ based on dataset $\{(X_t, Y_t)\}_{t=1}^{\tau}$.
$\widehat{\pi}(T)$	$=$	$\text{Argmax}_{a\in\mathcal{A}(T)}\mu(T,a)$.
$\widehat{\pi}$	$\triangleq$	Proposed policy solution.

6.6 Appendix: Proofs

Lemma 23. *For any terminal state $\overline{T}$, we have the following:*

$$\mu\left(\overline{T}\right) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} \mathcal{V}^*\left(\overline{T}\right), \sigma\left(\overline{T}\right)^2 \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} 0$$

Proof Let us start with the variance as it is a simpler result to prove. From Eq. (6.8) we have:

$$\sigma\left(\overline{T}\right)^2 = \sigma\left(T, \overline{a}\right)^2 = \sum_{l\in T} \widehat{p}\left(l\right)^2 \sigma\left(l\right)^2$$

Consider a branch $l \in T$. If $\mathbb{P}\left[l\left(X\right)=1\right]=0$ then $\widehat{p\left(l\right)}=0$. Otherwise, when $\mathbb{P}\left[l\left(X\right)=1\right]>0$, we have:

$$n\left(l\right) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} \infty \tag{6.16}$$

where we recall that $n\left(l\right)$ is the number of observed samples in l. On the other hand, according to Eq. (6.7) we have:

$$\sigma\left(l\right)^2 = \frac{\alpha\left(l\right)\beta\left(l\right)}{\left(\alpha\left(l\right)+\beta\left(l\right)\right)^2\left(1+\alpha\left(l\right)+\beta\left(l\right)\right)}$$

From Eq. (6.5) and Eq. (6.6) we have the following:

$$\alpha\left(l\right)+\beta\left(l\right) = n\left(l\right)+1$$

Replacing this in the expression of $\sigma\left(l\right)^2$, we get:

$$\begin{aligned}
\sigma\left(l\right)^2 &= \frac{\alpha\left(l\right)\beta\left(l\right)}{\left(n\left(l\right)+1\right)^2\left(n\left(l\right)+2\right)} \\
&\leq \frac{\left(n\left(l\right)+1\right)^2}{\left(n\left(l\right)+1\right)^2\left(n\left(l\right)+2\right)} \quad \text{Because } \alpha\left(l\right), \beta\left(l\right) \leq n\left(l\right)+1 \\
&\leq \frac{1}{n\left(l\right)+2} \xrightarrow[n(l)\to\infty]{\text{a.s}} 0
\end{aligned}$$

From Eq. (6.16) we get:

$$\sigma\left(l\right)^2 \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} 0$$

Therefore we deduce that:

$$\sigma\left(\overline{T}\right)^2 \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} 0$$

Let us show the result for the mean. From Eq. (6.8), we have:

$$\mu\left(\overline{T}\right) = \mu\left(T, \overline{a}\right) = \sum_{l\in T} \widehat{p}\left(l\right)\mu\left(l\right)$$

Again we consider a branch $l \in T$ such that $\mathbb{P}\left[l\left(X\right)=1\right]>0$. From Eq. (6.7), we have:

$$\begin{aligned}
\mu\left(l\right) &= \frac{\alpha\left(l\right)}{\alpha\left(l\right)+\beta\left(l\right)} \\
&= \frac{\alpha\left(l\right)}{2+n\left(l\right)} \\
&= \frac{1}{2+n\left(l\right)} + \frac{1}{2+n\left(l\right)} \sum_{t=1}^{n} l\left(X_t\right) \mathbb{1}\{\widehat{T}^{(t-1)}\left(X_t\right) = Y_t\}
\end{aligned}$$

Here $\{(X_t, Y_t)\}_{t=1}^{n}$ are the accumulated observed samples when simulating the states that share branch l. Our objective now is to show that:

$$\mu(l) \xrightarrow[n\to\infty]{\text{a.s}} \mathbb{P}[T(X) = Y | l(X) = 1] = p_k(l)$$

Since $\mathbb{P}[l(X) = 1] > 0$, then by the Strong Law of Large numbers we have:

$$\frac{n(l)}{n} \xrightarrow[n\to\infty]{\text{a.s}} \mathbb{P}[l(X) = 1] > 0$$

which also implies:

$$n(l) \xrightarrow[n\to\infty]{\text{a.s}} \infty \tag{6.17}$$

On the other hand we have:

$$\forall k \in \{1, \dots, K\} : \widehat{p}_k^{(t)}(l) \xrightarrow[t\to\infty]{\text{a.s}} p_k(l)$$

which means that:

$$\mathbb{P}\left[\forall \epsilon > 0, \exists \tau > 0, \forall t \geq \tau, \forall k \in \{1, \dots, K\} : \left|\widehat{p}_k^{(t)}(l) - p_k(l)\right| \leq \epsilon\right] = 1 \tag{6.18}$$

Take $\epsilon = \frac{1}{2}\min_{k \neq k'}\left\{\left|p_k(l) - p_{k'}(l)\right|\right\} > 0$. In light of Eq. (6.18) we have:

$$\begin{aligned}
&\mathbb{P}\left[\exists \tau > 0, \forall t \geq \tau, \forall k \in \{1, \dots, K\} : \left|\widehat{p}_k^{(t)}(l) - p_k(l)\right| \leq \epsilon\right] = 1 \\
\implies &\mathbb{P}\left[\exists \tau > 0, \forall t \geq \tau : \text{Argmax}_k\{p_k(l)\} = \text{Argmax}_k\{\widehat{p}_k^{(t)}(l)\}\right] = 1 \\
\implies &\mathbb{P}\left[\exists \tau > 0, \forall t \geq \tau : \widehat{T}^{(t-1)}(l) = T(l)\right] = 1
\end{aligned} \tag{6.19}$$

Note that $T(l)$ is the predicted class of T at branch l. Define $\tau > 0$ to be the random time such that:

$$\forall t \geq \tau : \widehat{T}^{(t-1)}(l) = T(l)$$

Eq. (6.19) implies that $\mathbb{P}[\tau < \infty] = 1$. Now consider $0 < \epsilon' < \epsilon$, we want to show that:

$$\mathbb{P}\left[\exists \tau' > 0, \forall n > \tau' : \left|\mu(l) - p_k(l)\right| \leq \epsilon'\right] = 1$$

For any $t' > 0$ we have the following:

$$\mathbb{P}\left[\exists \tau' > 0, \forall n > \tau' : \left|\mu(l) - p_k(l)\right| \leq \epsilon'\right] \geq \mathbb{P}\left[\exists \tau' > 0, \forall n > \tau' : \left|\mu(l) - p_k(l)\right| \leq \epsilon', \tau \leq t'\right] \tag{6.20}$$

When the event $\{\tau \leq t'\}$ happens, we have that for $n > \tau' > t'$:

$$\begin{aligned}
\left|\mu(l) - p_k(l)\right| &\leq \left|\frac{1}{2 + n(l)} + \frac{1}{2 + n(l)}\sum_{t=1}^{n} l(X_t)\, \mathbb{1}\{\widehat{T}^{(t-1)}(X_t) = Y_t\} - p_k(l)\right| \\
&\leq \left|\frac{1}{2 + n(l)} + \frac{1}{2 + n(l)}\sum_{t=1}^{t'} l(X_t)\, \mathbb{1}\{\widehat{T}^{(t-1)}(X_t) = Y_t\}\right| \\
&+ \left|\frac{1}{2 + n(l)}\sum_{t=t'+1}^{n} l(X_t)\, \mathbb{1}\{\widehat{T}^{(t-1)}(X_t) = Y_t\} - p_k(l)\right|
\end{aligned}$$

Since $\tau \leq t'$ then from the definition of τ we have:

$$\forall t \geq t' : \widehat{T}^{(t-1)}(l) = T(l)$$

Therefore we have:

$$\begin{aligned}\left|\mu(l) - p_k(l)\right| \leq & \left|\frac{1}{2+n(l)} + \frac{1}{2+n(l)} \sum_{t=1}^{t'} l(X_t)\, \mathbb{1}\{\widehat{T}^{(t-1)}(X_t) = Y_t\}\right| \\ & + \left|\frac{1}{2+n(l)} \sum_{t=t'+1}^{n} l(X_t)\, \mathbb{1}\{\widehat{T}(X_t) = Y_t\} - p_k(l)\right|\end{aligned}$$

For convenience, let us denote the terms:

$$\begin{aligned}A(t') &= \left|\frac{1}{2+n(l)} + \frac{1}{2+n(l)} \sum_{t=1}^{t'} l(X_t)\, \mathbb{1}\{\widehat{T}^{(t-1)}(X_t) = Y_t\}\right| \\ B(t') &= \left|\frac{1}{2+n(l)} \sum_{t=t'+1}^{n} l(X_t)\, \mathbb{1}\{\widehat{T}(X_t) = Y_t\} - p_k(l)\right|\end{aligned}$$

This yields:

$$A(T') + B(T') \leq \epsilon' \implies \left|\mu(l) - p_k(l)\right| \leq \epsilon'$$

Therefore, when we go back to Eq. (6.20) we get:

$$\begin{aligned}\mathbb{P}\left[\exists \tau' > 0, \forall n > \tau' : \left|\mu(l) - p_k(l)\right| \leq \epsilon'\right] &\geq \mathbb{P}\left[\exists \tau' > 0, \forall n > \tau' : A(t') + B(t') \leq \epsilon', \tau \leq t'\right] \\ &\geq \mathbb{P}\left[\exists \tau' > 0, \forall n > \tau' : A(t') + B(t') \leq \epsilon'\right] \\ &\quad - \mathbb{P}\left[\tau > t'\right] \qquad (6.21)\end{aligned}$$

Let us analyse the term $\mathbb{P}\left[\exists \tau' > 0, \forall n > \tau' : A(t') + B(t') \leq \epsilon'\right]$:

$$\begin{aligned}A(t') \leq \left|\frac{1+t'}{2+n(l)}\right| &\xrightarrow[n\to\infty]{\text{a.s}} 0 \;\text{ Because of Eq. (6.17)} \\ \implies \mathbb{P}\left[\exists \tau_1' > 0, \forall n > \tau_1' : A(t') \leq \frac{\epsilon'}{3}\right] &= 1 \qquad (6.22)\end{aligned}$$

On the other hand we have:

$$\begin{aligned}B(t') &\leq \left|\frac{1}{n(l) - t'} \sum_{t=t'+1}^{n} l(X_t)\, \mathbb{1}\{\widehat{T}(X_t) = Y_t\} - p_k(l)\right| \\ &\quad + \left|\left(\frac{1}{2+n(l)} - \frac{1}{n(l)-t'}\right) \sum_{t=t'+1}^{n} l(X_t)\, \mathbb{1}\{\widehat{T}(X_t) = Y_t\}\right| \\ &\leq \left|\frac{1}{n(l) - t'} \sum_{t=t'+1}^{n} l(X_t)\, \mathbb{1}\{\widehat{T}(X_t) = Y_t\} - p_k(l)\right| \\ &\quad + \frac{t'+2}{(2+n(l))(n(l)-t')} \sum_{t=t'+1}^{n} l(X_t)\, \mathbb{1}\{\widehat{T}(X_t) = Y_t\} \\ &\leq \left|\frac{1}{n(l) - t'} \sum_{t=t'+1}^{n} l(X_t)\, \mathbb{1}\{\widehat{T}(X_t) = Y_t\} - p_k(l)\right| \\ &\quad + \frac{(t'+2)(n-t')}{(2+n(l))(n(l)-t')}\end{aligned}$$

Eq. (6.17) yields:

$$\frac{(t'+2)(n-t')}{(2+n(l))(n(l)-t')} \xrightarrow[n\to\infty]{\text{a.s}} 0$$
$$\Longrightarrow \mathbb{P}\left[\exists \tau_2' > 0, \forall n > \tau_2' : \frac{(t'+2)(n-t')}{(2+n(l))(n(l)-t')} \leq \frac{\epsilon'}{3}\right] = 1 \tag{6.23}$$

As for the remaining term, the Strong Law of Large Numbers yields:

$$\frac{1}{n(l)-t'} \sum_{t=t'+1}^{n} l(X_t) \mathbb{1}\{\widehat{T}(X_t) = Y_t\} \xrightarrow[n\to\infty]{\text{a.s}} p_k(l)$$
$$\Longrightarrow \mathbb{P}\left[\exists \tau_3' > 0, \forall n > \tau_3' : \left|\frac{1}{n(l)-t'} \sum_{t=t'+1}^{n} l(X_t) \mathbb{1}\{\widehat{T}(X_t) = Y_t\} - p_k(l)\right| \leq \frac{\epsilon'}{3}\right] = 1 \tag{6.24}$$

From Eq. (6.22), Eq. (6.23) and Eq. (6.24), we get by using $\tau' = \max\{\tau_1', \tau_2', \tau_3'\}$:

$$\mathbb{P}\left[\exists \tau' > 0, \forall n > \tau' : A(t') + B(t') \leq \epsilon'\right] = 1$$

Going back to Eq. (6.21) yields:

$$\mathbb{P}\left[\exists \tau' > 0, \forall n > \tau' : \left|\mu(l) - p_k(l)\right| \leq \epsilon'\right] \geq 1 - \mathbb{P}\left[\tau > t'\right]$$

The key point now is that this is satisfied for all $t' > 0$, and since $\mathbb{P}[\tau = \infty] = 0$ we get:

$$\mathbb{P}\left[\exists \tau' > 0, \forall n > \tau' : \left|\mu(l) - p_k(l)\right| \leq \epsilon'\right] \geq 1$$

This means that:

$$\mu(l) \xrightarrow[n\to\infty]{\text{a.s}} p_k(l)$$

On the other hand, we have the consistency of the estimator $\widehat{p}(l)$:

$$\widehat{p}(l) \xrightarrow[n\to\infty]{\text{a.s}} \mathbb{P}[l(X) = 1]$$

and also that:

$$n \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} \infty$$

Using all of these results, we get:

$$\mu\left(\overline{T}\right) = \mu(T, \overline{a}) = \sum_{l \in T} \widehat{p}(l)\mu(l) \xrightarrow[v(\overline{T})\to\infty]{\text{a.s}} \sum_{l \in T} \mathbb{P}[l(X) = 1]\mathbb{P}[T(X) = Y | l(X) = 1] = \mathcal{V}^*\left(\overline{T}\right)$$

■

Lemma 24. *For any state T and action $a \in \mathcal{A}(T)$ such that $T \xrightarrow{a} T'$ we have:*

$$v(T') \xrightarrow[v(T)\to\infty]{\text{a.s}} \infty$$

Proof For convenience, let us denote $\mathcal{A}(T) = \{a_1, \ldots, a_{|\mathcal{A}(T)|}\}$ and:

$$\forall u \in \{1, \ldots, |\mathcal{A}(T)|\} : T \xrightarrow{a_u} T_u$$

Without loss of generality, let us show that:

$$v\left(T_1\right) \xrightarrow[v(T)\to\infty]{\text{a.s}} \infty$$

Equivalently, we will prove the following result:

$$\mathbb{P}\left[\exists V_1 > 0, \forall v\left(T\right) > 0 : v\left(T_1\right) < V_1\right] = 0$$

which translates to "The probability of visiting T_1 a finite number of times is 0".

$$\exists V_1 > 0, \forall v\left(T\right) > 0 : v\left(T_1\right) < V_1 \iff \exists V > 0, \forall v\left(T\right) > V : \theta\left(T, a_1\right) < \max_{2\leq u\leq |\mathcal{A}(T)|} \theta\left(T, a_u\right)$$

This equivalence just means that, there exists a number of visits of T after which action a_1 (and consequently next state T_1) are never visited again. Now we get:

$$\begin{aligned}\mathbb{P}\left[\exists V_1 > 0, \forall v\left(T\right) > 0 : v\left(T_1\right) < V_1\right] &= \mathbb{P}\left[\exists V > 0, \forall v\left(T\right) > V : \theta\left(T, a_1\right) < \max_{2\leq u\leq |\mathcal{A}(T)|} \theta\left(T, a_u\right)\right] \\ &\leq \sum_{V=1}^{\infty} \mathbb{P}\left[\forall v\left(T\right) > V : \theta\left(T, a_1\right) < \max_{2\leq u\leq |\mathcal{A}(T)|} \theta\left(T, a_u\right)\right]\end{aligned} \tag{6.25}$$

Now it suffices to show that:

$$\forall V > 0 : \mathbb{P}\left[\forall v\left(T\right) > V : \theta\left(T, a_1\right) < \max_{2\leq u\leq |\mathcal{A}(T)|} \theta\left(T, a_u\right)\right] = 0 \tag{6.26}$$

To do this, we first augment our notation by considering $\theta^v\left(T, a_u\right) \sim \mathcal{N}\left(\mu^v\left(T, a_u\right), \sigma^v\left(T, a_u\right)^2\right)$ for all $1 \leq u \leq |\mathcal{A}\left(T\right)|$. $\theta^v\left(T, a_u\right)$ is the posterior on $\mathcal{Q}^*\left(T, a_u\right)$ when $v\left(T\right) = v$. Moreover, we define $H\left(v\right)$ the filtration spanned by the history $\left\{\mu^{v'}\left(T, a_u\right), \sigma^{v'}\left(T, a_u\right)\right\}$ for all $u \in \{1, \ldots, |\mathcal{A}\left(T\right)|\}$ and all $V + 1 \leq v' \leq v$.

$$\begin{aligned}&\mathbb{P}\left[\forall v\left(T\right) > V : \theta\left(T, a_1\right) < \max_{2\leq u\leq |\mathcal{A}(T)|} \theta\left(T, a_u\right)\right] \\ &= \mathbb{E}\left[\prod_{v=V+1}^{\infty} \mathbb{1}\left\{\theta^v\left(T, a_1\right) < \max_{2\leq u\leq |\mathcal{A}(T)|} \theta^v\left(T, a_u\right)\right\}\right] \\ &= \mathbb{E}\left[\mathbb{E}\left[\prod_{v=V+1}^{\infty} \mathbb{1}\left\{\theta^v\left(T, a_1\right) < \max_{2\leq u\leq |\mathcal{A}(T)|} \theta^v\left(T, a_u\right)\right\} \middle| \bigcup_{v=V+1}^{\infty} H\left(v\right)\right]\right] \\ &= \prod_{v=V+1}^{\infty} \mathbb{E}\left[\mathbb{E}\left[\mathbb{1}\left\{\theta^v\left(T, a_1\right) < \max_{2\leq u\leq |\mathcal{A}(T)|} \theta^v\left(T, a_u\right)\right\} \middle| H\left(v\right)\right]\right]\end{aligned} \tag{6.27}$$

We can take the product outside the expectation because of condition independence on the histories. Now it suffices to show that:

$$\forall v \geq V + 1 : \mathbb{E}\left[\mathbb{E}\left[\mathbb{1}\left\{\theta^v\left(T, a_1\right) < \max_{2\leq u\leq |\mathcal{A}(T)|} \theta^v\left(T, a_u\right)\right\} \middle| H\left(v\right)\right]\right] < 1$$

We have the following:

$$\mathbb{E}\left[\mathbb{1}\left\{\theta^v\left(T, a_1\right) < \max_{2\leq u\leq |\mathcal{A}(T)|} \theta^v\left(T, a_u\right)\right\} \middle| H\left(v\right)\right] = \mathbb{P}\left[\theta^v\left(T, a_1\right) < \max_{2\leq u\leq |\mathcal{A}(T)|} \theta^v\left(T, a_u\right) \middle| H\left(v\right)\right]$$

To how that:

$$\mathbb{P}\left[\theta^v\left(T,a_1\right) < \max_{2\leq u\leq|\mathcal{A}(T)|}\theta^v\left(T,a_u\right)\middle|H\left(v\right)\right] < 1 \tag{6.28}$$

It suffices to show that:

$$\mathbb{P}\left[\theta^v\left(T,a_1\right) \geq \max_{2\leq u\leq|\mathcal{A}(T)|}\theta^v\left(T,a_u\right)\middle|H\left(v\right)\right] > 0 \tag{6.29}$$

Given the history $H\left(v\right)$, we have:

$$\forall 1\leq u\leq\left|\mathcal{A}\left(T\right)\right| : \theta^v\left(T,a_u\right)\sim\mathcal{N}\left(\mu^v\left(T,a_u\right),\sigma^v\left(T,a_u\right)^2\right)$$

Now we use the following remark:

$$\theta^v\left(T,a_1\right)\geq\mu^v\left(T,a_1\right),\mu^v\left(T,a_1\right)\geq\max_{2\leq u\leq|\mathcal{A}(T)|}\theta^v\left(T,a_u\right)\implies\theta^v\left(T,a_1\right)\geq\max_{2\leq u\leq|\mathcal{A}(T)|}\theta^v\left(T,a_u\right)$$

which yields:

$$\mathbb{P}\left[\theta^v\left(T,a_1\right)\geq\mu^v\left(T,a_1\right),\mu^v\left(T,a_1\right)\geq\max_{2\leq u\leq|\mathcal{A}(T)|}\theta^v\left(T,a_u\right)\right]\leq\mathbb{P}\left[\theta^v\left(T,a_1\right)\geq\max_{2\leq u\leq|\mathcal{A}(T)|}\theta^v\left(T,a_u\right)\right]$$

These probabilities are conditioned on the history $H\left(v\right)$, we just omitted writing again to avoid overloading the expressions. Let us analyse the L.H.S.

$$\begin{aligned}
&\mathbb{P}\left[\theta^v\left(T,a_1\right)\geq\mu^v\left(T,a_1\right),\mu^v\left(T,a_1\right)\geq\max_{2\leq u\leq|\mathcal{A}(T)|}\theta^v\left(T,a_u\right)\right]\\
&=\underbrace{\mathbb{P}\left[\theta^v\left(T,a_1\right)\geq\mu^v\left(T,a_1\right)\right]}_{=\frac{1}{2}}\mathbb{P}\left[\exists 2\leq u\leq\left|\mathcal{A}\left(T\right)\right| : \mu^v\left(T,a_1\right)\geq\theta^v\left(T,a_u\right)\right]\\
&=\frac{1}{2}\sum_{u=2}^{|\mathcal{A}(T)|}\mathbb{P}\left[\theta^v\left(T,a_u\right)\leq\mu\right]\\
&=\frac{1}{2}\sum_{u=2}^{|\mathcal{A}(T)|}\Phi\left(\frac{\mu^v\left(T,a_1\right)-\mu^v\left(T,a_u\right)}{\sigma^v\left(T,a_u\right)}\right)>0
\end{aligned}$$

where Φ is the cumulative distribution function of the standard Normal distribution. Note that the second line stems from the independence of the $\left\{\theta^v\left(T,a_u\right)\right\}_{u=1}^{|\mathcal{A}(T)|}$. We deduce Eq. (6.29) and therefore Eq. (6.28). Then from Eq. (6.27), Eq. (6.26) and Eq. (6.25) we deduce that:

$$\mathbb{P}\left[\exists V_1>0,\forall v\left(T\right)>0 : v\left(T_1\right)<V_1\right]=0$$

which concludes our proof. ■

Corollary 25. *Every state T is visited infinitely often as the number of iterations of TSDT grows to infinity, i.e.*

$$v\left(T\right)\xrightarrow[M\to\infty]{\text{a.s}}\infty$$

Proof Corollary 25 is straightforward to prove by Induction using Lemma 24 and the fact that M is the number of visits of the root Ω. ■

Proposition 22. *For any state T, TSDT satisfies the optimal convergence property:*

$$\mu(T) \xrightarrow[M\to\infty]{\text{a.s}} \mathcal{V}^*(T),\ \sigma(T)^2 \xrightarrow[M\to\infty]{\text{a.s}} 0,\ \widehat{\pi}(T) \xrightarrow[M\to\infty]{a.s} \pi^*(T)$$

Proof By Corollary 25, it suffices to show that for any state T we have:

$$\begin{aligned}\mu(T) &\xrightarrow[v(T)\to\infty]{\text{a.s}} \mathcal{V}^*(T)\\ \sigma(T)^2 &\xrightarrow[v(T)\to\infty]{\text{a.s}} 0\\ \widehat{\pi}(T) &\xrightarrow[v(T)\to\infty]{\text{a.s}} \pi^*(T)\end{aligned}$$

By induction on $\mathcal{S}(T)$. If $\mathcal{S}(T)$ is the maximum number of possible splits, then $T = \overline{T}$ is a terminal state and the result stems from Lemma 23. On the other hand, suppose the result is satisfied for all states with a number of splits larger than $n \geq 1$ where n is an integer smaller than the maximum number of splits. Now consider a state T with $\mathcal{S}(T) = n-1$. For convenience, let us denote $\mathcal{A}(T) = \{a_1, \ldots, a_{|\mathcal{A}(T)|}\}$ with:

$$\begin{aligned}\forall u \in \{2, \ldots, |\mathcal{A}(T)|\} : T &\xrightarrow{a_u} T_u\\ a_1 &= \overline{a}\end{aligned}$$

From Lemma 23 and Lemma 24, we have:

$$\begin{aligned}\mu(T, a_1) = \mu\left(\overline{T}\right) &\xrightarrow[v(T)\to\infty]{\text{a.s}} \mathcal{V}^*\left(\overline{T}\right)\\ \sigma(T, a_1)^2 = \sigma\left(\overline{T}\right)^2 &\xrightarrow[v(T)\to\infty]{\text{a.s}} 0\end{aligned}$$

Moreover, all actions a_u for $u \in \{2, \ldots, |\mathcal{A}(T)|\}$ are split actions, and therefore $\mathcal{S}(T_u) = n$. By the induction hypothesis and Lemma 24 we get:

$$\begin{aligned}\forall u \in \{2, \ldots, |\mathcal{A}(T)|\} : \mu(T, a_u) = -\lambda + \mu(T_u) &\xrightarrow[v(T)\to\infty]{\text{a.s}} -\lambda + \mathcal{V}^*(T_u) = \mathcal{Q}^*(T, a_u)\\ \sigma(T, a_u)^2 = \sigma(T_u)^2 &\xrightarrow[v(T)\to\infty]{\text{a.s}} 0\end{aligned}$$

From Eq. (6.15), we have:

$$\begin{cases}\widehat{\pi}(T) &= \text{Argmax}_{a\in\mathcal{A}(T)}\mu(T, a) \xrightarrow[v(T)\to\infty]{\text{a.s}} \text{Argmax}_{a\in\mathcal{A}(T)}\mathcal{Q}^*(T, a) = \pi^*(T)\\ \mu(T) &= \mu(T, a^*) \xrightarrow[v(T)\to\infty]{\text{a.s}} \mathcal{Q}^*(T, \pi^*(T)) = \mathcal{V}^*(T)\\ \sigma(T) &= \sigma(T, a^*) \xrightarrow[v(T)\to\infty]{\text{a.s}} 0\end{cases}$$

By induction now we deduce that for any state T:

$$\begin{aligned}\mu(T) &\xrightarrow[v(T)\to\infty]{\text{a.s}} \mathcal{V}^*(T)\\ \sigma(T)^2 &\xrightarrow[v(T)\to\infty]{\text{a.s}} 0\\ \widehat{\pi}(T) &\xrightarrow[v(T)\to\infty]{\text{a.s}} \pi^*(T)\end{aligned}$$

Using Corollary 25 yields:

$$\begin{aligned}\mu(T) &\xrightarrow[M\to\infty]{\text{a.s}} \mathcal{V}^*(T)\\ \sigma(T)^2 &\xrightarrow[M\to\infty]{\text{a.s}} 0\\ \widehat{\pi}(T) &\xrightarrow[M\to\infty]{\text{a.s}} \pi^*(T)\end{aligned}$$

Thus concluding our proof.

■

6.7 Appendix: Pseudo-code

Algorithm 3 TSDT

Input: M number of iterations, m number of observed samples per Simulation, $\lambda \geq 0$.
Initialise $\pi(\overline{a}|T) = 1$ and $visited(T) = False$ for all states.
for $t = 1$ to M **do**
  Unroll $\pi : \Omega = T_0 \xrightarrow{a_0} T_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{L-1}} T_L = \overline{T_{L-1}}$ ▷ Selection
  if $visited(T_{L-1})$ **then**
    Observe new m samples in T_L ▷ Simulation
    Update $\theta(T_{L-1}, \overline{a})$ as per (6.8)
  else
    Observe m samples in T_{L-1} ▷ Simulation
    Update $\theta(T_{L-1}, \overline{a})$ as per (6.8)
    for $a \in \mathcal{A}(T_{L-1}) \setminus \{\overline{a}\}$ **do** ▷ Expansion
      Set $T_{L-1} \xrightarrow{a} T$
      Initialise $\theta(T) = \theta(T, \overline{a})$ as per (6.8)
      Update $\theta(T_{L-1}, a) = -\lambda + \theta(T')$ as per (6.9)
    end for
    Update $visited(T_{L-1}) = True$
  end if
  Update $\theta(T_{L-1})$ as per (6.15)
  Update the policy $\pi(a|T_{L-1})$ for all $a \in \mathcal{A}(T_{L-1})$ as per (6.10)
  for $j = L-1$ to 1 **do** ▷ Backpropagation
    Update $\theta(T_{j-1}, a_{j-1}) = -\lambda + \widehat{\mathcal{V}}(T_j)$ as per (6.15)
    Update $\theta(T_{j-1})$ as per (6.15)
    Update the policy $\pi(a|T_{j-1})$ for all $a \in \mathcal{A}(T_{j-1})$ as per (6.10)
  end for
end for
Define the solution $\widehat{\pi}(T) = \text{Argmax}_{a \in \mathcal{A}(T)} \mu(T, a)$ for all states T
Unroll $\widehat{\pi} : \Omega = T_0 \xrightarrow{\pi(T_0)} T_1 \xrightarrow{\pi(T_1)} \ldots \xrightarrow{\pi(T_{L-1})} T_L = \overline{T_{L-1}}$
return T_{L-1}

Chapter 7

Online Decision Trees with Finite-Time Optimality guarantees

Contents

7.1 Introduction 175
7.2 Preliminaries 175
7.3 The Algorithm: Online-Branches 176
7.4 Theoretical Analysis 177
7.5 Experiments 180
7.6 Conclusion 182
7.7 Appendix: Proofs 188

7.1 Introduction

In Chapter 5 and Chapter 6, we introduced UCDT, EGDT and TSDT, three MCTS algorithms that seek optimal sparse DT classifiers from data streams. The motivation behind their MCTS formulations pertains to solving the suboptimality issue from which the state of the art online DT methods suffer. An issue that is mainly due to the greedy top-down construction method, as we showed especially on the XOR experiments (see Section 5.5 and Section 6.3). However, when developing our previous algorithms, we could only derive asymptotic convergence guarantees, as outlined in Section 5.2.2. These guarantees only ensure that the algorithm will retrieve the optimal DT given an infinite stream of data, they do not inform us about the dependency between optimality and the number of observed data. As such, it is hard to compare these algorithms from a theoretical standpoint. Moreover, UCDT, EGDT and TSDT are based upon DT-Miner's MDP, introduced in Section 3.2.3, and we showed in Part I that this MDP does not benefit from all properties of the problem. This motivated us to refine it into Branches's MDP, introduced in Section 4.2.1, which yielded a substantial gain in performance. It is therefore natural to reconsider this MDP for the online setting.

In this chapter, we extend Branches to handling data streams and we call the induced algorithm Online-Branches. Essentially, Online-Branches stores and updates a sliding window of fixed size containing data from the stream. With each iteration, this window is updated with the newly observed data, and it plays the role of the batch dataset $\mathcal{D}$ for Branches. The interesting feature of this algorithm is that it is amenable to a finite time analysis, which comes in the form of a high-probability bound on the optimality of the algorithm depending on the number of observed samples so far. The downside however, is that Online-Branches does not satisfy an asymptotic convergence guarantee. Indeed, Online-Branches does not run forever, it terminates, and once it does, it will be optimal with a high probability, but there will always be a small probability that it is suboptimal, unlike UCDT, EGDT and TSDT, which can run forever and are guaranteed to converge optimally in the limit. Nonetheless, this suboptimality probability is small and decreases with the size of the sliding window. Furthermore, Online-Branches is almost always optimal in our experiments, without reaching timeout. We summarise this chapter's contributions as follows:

- Extending Branches to handling data streams through Online-Branches.
- Deriving a finite-time guarantee for Online-Branches.
- Showing empirically that Online-Branches outperforms all the greedy methods VFDT, HAT, EFDT and SGT along with our MCTS methods UCDT, EGDT and TSDT.

We note that the theoretical analysis we perform in this chapter applies to using Branches with a single batch of data from the stream. This is due to the i.i.d. assumption. The reason we augmented Branches to handling a sliding window is to be inline with the data stream learning framework and prepare it for a future version that handles concept drift.

7.2 Preliminaries

The state and action spaces are the same as Branches' (see Section 4.2.1). The reward for any split action is also the same. However, for any unit-state (branch) l, the value $\mathcal{H}(l)$ is no longer available. One way to alleviate this issue would be to consider $r(l, \overline{a}) = \mathcal{H}(l)$ and then estimate these quantities in similar fashion to what we did in Chapter 5 and Chapter 6. Nonetheless, we opted for a different approach in this chapter, where we rather directly define $r(l, \overline{a})$ as the

estimator. Indeed, we observe a fixed number n of examples from the stream $\{(X_t, Y_t)\}_{t=1}^n$ and we define:

$$r(l, \overline{a}) = \frac{1}{n} \max_{1 \leq k \leq K} n_k(l) = \frac{1}{n} \max_{1 \leq k \leq K} \sum_{t=1}^{n} l(X_t) \mathbb{1}\{Y_t = k\} \tag{7.1}$$

The idea behind this definition pertains to estimating:

$$\mathcal{H}(l) = \mathbb{P}[l(X) = 1, Y_t = k^*(l)] = \max_{1 \leq k \leq K} \mathbb{E}[l(X) \mathbb{1}\{Y = k\}] \tag{7.2}$$

However, we note that $r(l, \overline{a})$ in Eq. (7.1) is a biased estimator of $\mathcal{H}(l)$ in Eq. (7.2) due to Jensen's inequality and the convexity of the max function as shown below.

$$\mathbb{E}[r(l, \overline{a})] = \frac{1}{n} \mathbb{E}\left[\max_{1 \leq k \leq K} \sum_{t=1}^{n} l(X_t) \mathbb{1}\{Y_t = k\}\right] \geq \frac{1}{n} \max_{1 \leq k \leq K} \mathbb{E}\left[\sum_{t=1}^{n} l(X_t) \mathbb{1}\{Y_t = k\}\right] = \mathcal{H}(l) \tag{7.3}$$

Nonetheless, $r(l, \overline{a})$ concentrates around $\mathcal{H}(l)$ for large values of n, as we will show in Lemma 26. Note that this definition of the rewards is different from the earlier definitions in the sense that our rewards are stochastic instead of being deterministic.

Consider a policy π and a non-absorbing state T, we recall that the return of π from T, as defined in Eq. (2.1) is:

$$\mathcal{R}^{\pi}(T) = \sum_{t=0}^{\infty} r(T_t, \pi(T_t))$$

where $T_0 = T$ and $\forall t \geq 0 : T_t \xrightarrow{\pi(T_t)} T_{t+1}$. Due to the stochasticity of the rewards, the value and the return are no longer equal:

$$\forall \pi : \mathcal{V}^{\pi}(T) = \mathbb{E}[\mathcal{R}^{\pi}(T)] \neq \mathcal{R}^{\pi}(T)$$

Moreover, Proposition 6 is no longer satisfied because of the bias of the rewards as shown in (7.3). Nonetheless, we will seek an optimal policy with respect to the return function, i.e.

$$\widehat{\pi} = \text{Argmax}_{\pi} \mathcal{R}^{\pi}(\Omega)$$

The proposed solution $\widehat{\pi}$ is random due to the randomness of the return itself, hence why we denote it with the hat symbol. In Section 7.4, we will prove, with high probability, that Online-Branches is optimal after observing a finite number of samples from the stream.

7.3 The Algorithm: Online-Branches

Online-Branches is analogous to Branches. The difference pertains to the estimates $\mathcal{Q}(l, a)$, which are no longer calculated based on the fixed dataset $\mathcal{D}$. Instead, we store a sliding window of samples $\{(X_t, Y_t)\}_{t=1}^n$ with fixed size n, that gets updates from the stream at each iteration by adding the new sample and discarding the oldest one. We use this window to initialise or update the different estimates $\mathcal{Q}(l, a)$ during Expansion and Backpropagation. In Section 7.4, we analyse the influence of n on optimality. We also note that the caching procedure of Branches' Dynamic Programming and its Branch & Bound pruning remain the same for Online-Branches.

Since the estimates $\mathcal{Q}(l,a)$ are now stochastic, it is possible to prune portions of the search space that include branches from T^*, it is also possible to poorly estimate $\mathcal{H}(l)$ for some branches in T^*. These situations could result in suboptimality, i.e. $\widehat{\pi} \neq \pi^*$, where we recall that π^* is the policy inducing T^*, i.e. $T^{\pi^*} = T^*$. These unfortunate situations cannot be always avoided in this setting, thus Online-Branches is not guaranteed to be always optimal when it terminates. Nevertheless, we shall see in Section 7.4 that Online-Branches achieves optimality with high-probability, depending on the size of the sliding window.

7.4 Theoretical Analysis

The main question of this section is: **What sort of theoretical results guarantee the optimality of Online-Branches? More precisely, we seek a high-probability guarantee that $\widehat{\pi} = \pi$.**

Looking at our reward function reveals that the source of stochasticity is due to the terminal action. Indeed, for any branch l the reward $r(l,\overline{a})$, defined in Eq. (7.1), is a random variable estimating $\mathcal{H}(l)$. Therefore, the first step in our analysis is to guarantee that these estimates are *accurate*. In Section 7.2 we showed that $r(l,\overline{a})$ is biased. However, it still concentrates around $\mathcal{H}(l)$ in a $\mathcal{O}\left(\sqrt{\frac{1}{n}}\right)$ rate as per Lemma 26.

Lemma 26. *Let l be a branch and denote the probabilities:*

$$\forall 1 \leq k \leq K : p(l,k) = \mathbb{P}\left[l(X) = 1, Y = k\right]$$

Then for any $0 < \delta < 1$, $r(l,\overline{a})$ satisfies the following concentration inequality:

$$\mathbb{P}\left[\left|r(l,\overline{a}) - \mathcal{H}(l)\right| \leq \sqrt{\frac{\log\frac{2}{\delta}}{2n}}\right] \geq 1 - \delta - 2\sum_{k \neq k^*(l)} \exp\left\{-\frac{n\left[p(l,k^*(l)) - p(l,k)\right]^2}{2}\right\}$$

The next step in our analysis pertains to proving that the return $\mathcal{R}^{\pi}(\Omega)$ of any policy π concentrates around $\mathcal{H}_{\lambda}(T^{\pi})$. Corollary 27 is straightforward to derive from Lemma 26.

Corollary 27. *Let π be a policy and l a branch, then for any $0 < \delta < 1$, there exists $n_0 \geq 1$ sufficiently large such that for any $n \geq n_0$ the return $\mathcal{R}^{\pi}(l)$ concentrates around $\mathcal{H}_{\lambda}(T^{\pi}(l))$ as follows:*

$$\mathbb{P}\left[\left|\mathcal{R}^{\pi}(l) - \mathcal{H}_{\lambda}(T^{\pi}(l))\right| \leq \sqrt{\frac{|T^{\pi}(l)|^2 \log\frac{2|T^{\pi}(l)|}{\delta}}{2n}}\right] \geq 1 - \delta -$$

$$2\sum_{l \in T^{\pi}(l)} \sum_{k \neq k^*(l)} \exp\left\{-\frac{n\left[p(l,k^*(l)) - p(l,k)\right]^2}{2}\right\}$$

Lemma 26 guarantees that, with high probability, the reward $r(l,\overline{a})$ is within a close vicinity of the true value $\mathcal{H}(l)$. On the other hand, if these rewards of the terminal action are very close to the true values for all possible branches, then the returns $\mathcal{R}^{\pi}(\Omega)$ would also be very close to the values $\mathcal{H}_{\lambda}(T^{\pi})$ for all policies (as demonstrated in Corollary 27). As a consequence, solving $\text{Argmax}_T \mathcal{H}_{\lambda}(T)$ becomes equivalent to solving $\text{Argmax}_{\pi} \mathcal{R}^{\pi}(\Omega)$, and thus

Online-Branches would satisfy the same optimality guarantee as a hypothetical application of Branches to a setting where we have access to $r(l, \overline{a}) = \mathcal{H}(l)$. In this case, optimality would be guaranteed by Theorem 8.

We note that in our reasoning, this optimality happens when all rewards $r(l, \overline{a})$ are very close to $\mathcal{H}(l)$, thus a first natural approach to analyse the optimality of Online-Branches is to control the probability $\mathbb{P}\left[\forall l : \left|r(l, \overline{a}) - \mathcal{H}(l)\right| \leq \epsilon\right]$ for some adequately chosen $\epsilon > 0$. However, it is not clear how to choose $\epsilon > 0$ in this case. An intuitive value would be $\epsilon = \min_{l,l':r(l,\overline{a})\neq r(l',\overline{a})} \frac{1}{2}|r(l, \overline{a}) - r(l', \overline{a})|$, but there are reasons to suggest the failure of this proposal because it does not involve the penalty parameter λ. Indeed, consider a classification problem with only one binary feature, this means that $T^* = \{\Omega\}$ or $T^* = \{l_1, l_2\}$ where $\text{Ch}(\Omega, 1) = \{l_1, l_2\}$. Suppose that $\mathcal{H}(\Omega) = 0.9, \mathcal{H}(l_1) = 0.5, \mathcal{H}(l_2) = 0.5$ and $\lambda = 0.05$. Further suppose that the optimal sparse DT here is in fact $T^* = \{l_1, l_2\}$ and that it is pure, i.e. $\mathcal{H}(T^*) = 1$. In this case we have $\mathcal{H}_\lambda(\Omega) = 0.9$ and $\mathcal{H}_\lambda(\{l_1, l_2\}) = 0.95$. Moreover $\epsilon = \frac{1}{2}(0.9 - 0.5) = 0.2$, thus even if we enforce $\forall l : |r(l, \overline{a}) - \mathcal{H}(l)| \leq \epsilon$, which translates to:

$$\begin{cases} 0.7 \leq r(\Omega, \overline{a}) \leq 1.1 \\ 0.3 \leq r(l_1, \overline{a}) \leq 0.7 \\ 0.3 \leq r(l_2, \overline{a}) \leq 0.7 \end{cases}$$

we could get the following samples satisfying these requirements:

$$\begin{cases} r(\Omega, \overline{a}) = 0.9 \\ r(l_1, \overline{a}) = 0.4 \\ r(l_2, \overline{a}) = 0.4 \end{cases}$$

leading to $\mathcal{R}^*(\Omega) = -\lambda + r(l_1, \overline{a}) + r(l_2, \overline{a}) = 0.6 \leq 0.9 = r(\Omega, \overline{a}) = \mathcal{R}^\pi(\Omega)$ where π is the policy taking the terminal action at the root, i.e. $\pi(\Omega) = \overline{a}$. In this case, even though we enforced:

$$\begin{cases} \forall l : |r(l, \overline{a}) - \mathcal{H}(l)| \leq \epsilon \\ \epsilon = \min_{l,l':r(l,\overline{a})\neq r(l',\overline{a})} \frac{1}{2}|r(l, \overline{a}) - r(l', \overline{a})| \end{cases}$$

we still got $\widehat{\pi} = \pi \neq \pi^*$.

The difficulty of choosing ϵ is not the only issue with this reasoning. Even assuming that we had found an adequate ϵ, how can we derive a lower bound on $\mathbb{P}\left[\forall l : \left|r(l, \overline{a}) - \mathcal{H}(l)\right| \leq \epsilon\right]$? Using a union bound yields:

$$\mathbb{P}\left[\forall l : \left|r(l, \overline{a}) - \mathcal{H}(l)\right| \leq \epsilon\right] \geq 1 - \sum_l \mathbb{P}\left[\left|r(l, \overline{a}) - \mathcal{H}(l)\right| > \epsilon\right]$$

then with Lemma 26, we bound the terms in the sum from below. However, the issue here pertains to summing over all the possible branches l, which is an immense quantity that would render the bound too loose, even if each term in the sum decreases exponentially with n. We need to design a more careful reasoning to avoid both the shortcoming of looking for an adequate ϵ and the shortcoming related to the immense quantity over which we sum.

When such careful analysis is undertaken, we derive Theorem 28. The main takeaway from this result is that we derive an upper bound on the number of observed samples, this upper bound is smaller than $n + (C+1)^q$ and can be in fact significantly smaller depending on the

problem's parameters $q, C, \lambda, \mathcal{S}(T^*), \mathcal{H}(T^*)$ and the gaps in $\mathcal{H}_\lambda$ between T^* and the other DTs. We show that the probability that Online-Branches is optimal while observing a number of samples smaller than our upper bound converges to 1 exponentially in n. We provide a proof sketch below. For the full technical proof, please refer to Section 7.7.

Theorem 28. *For an instance of the classification problem with $q \geq 2$ features, $0 < \lambda \leq 1$ the penalty parameter, $C \geq 2$ the number of categories per feature and $n \geq 1$ the size of the sliding window, let $N(q, C, \lambda, n)$ be the total number of observed samples by Online-Branches when it terminates. Then there exists problem-dependent constants $\epsilon > 0, \delta > 0$ such that:*

$$\mathbb{P}\left[\widehat{\pi} = \pi^*, N(q, C, \lambda, n) \leq n + \psi(\kappa, q)\right] \geq 1 - 2\psi(\kappa, q)\left\{2\exp\left(-2n\epsilon^2\right) + K\exp\left(-\frac{n\delta^2}{2}\right)\right\}$$
$$\xrightarrow[n\to\infty]{} 1$$

with the following quantities:

$$\psi(\kappa, q) = \sum_{h=0}^{\kappa} C^h \binom{q}{h} \leq (C+1)^q$$
$$\kappa = \min\left\{\left\lfloor \frac{(\lambda + (C-1)\epsilon)\mathcal{S}(T^*) + 1 - \lambda - \mathcal{H}(T^*) + 2\epsilon}{\lambda - (C-1)\epsilon}\right\rfloor + 1, q\right\}$$
$$\epsilon = \frac{1}{2}\min\left\{\min_{T \neq T^*} \frac{\mathcal{H}_\lambda(T^*) - \mathcal{H}_\lambda(T)}{2 + (C-1)\left[\mathcal{S}(T) + \mathcal{S}(T^*)\right]}, \frac{\lambda}{C-1}\right\}$$
$$\delta = \min_{l, k \neq k^*(l)} \left\{p(l, k^*(l)) - p(l, k)\right\}$$

Proof Sketch: For any branch l and $\epsilon > 0$, consider the event:

$$\mathcal{A}(l, \epsilon) = \left\{\left| r(l, \overline{a}) - \mathcal{H}(l) \right| \leq \epsilon, \left| \frac{n(l)}{n} - p(l) \right| \leq \epsilon\right\}$$

This event describes the concentration of the $r(l, \overline{a})$ and $\frac{n(l)}{n}$ around $\mathcal{H}(l)$ and $p(l)$ respectively. The idea is to find some fixed set of branches $\mathcal{B}'(\epsilon)$ such that the event:

$$\Gamma(\epsilon) = \bigcap_{l \in \mathcal{B}'(\epsilon)} \mathcal{A}(l, \epsilon)$$

would imply optimality $\widehat{\pi} = \pi^*$ and some upper bound on the number of observed samples after termination. Intuitively, if $\mathcal{A}(l, \epsilon)$ happens for all the expanded branches and their children for some very small $\epsilon > 0$ (well concentrated) then the algorithm should behave in similar fashion to Branches and terminate. As such, we seek $\mathcal{B}'(\epsilon)$ such that $\Gamma(\epsilon) = \bigcap_{l \in \mathcal{B}'(\epsilon)} \mathcal{A}(l, \epsilon)$ would imply the concentration of the expanded branches and their children. To seek this, let us start from the opposite. We show that when the event $\mathcal{A}(l, \epsilon)$ happens for all expanded branches and for all branches on the way to T^*, then a branch l can only be expanded if it satisfies:

$$\mathcal{S}(l) \leq \frac{(\lambda + (C-1)\epsilon)\mathcal{S}(T^*) + 1 - \lambda - \mathcal{H}(T^*) + 2\epsilon}{\lambda - (C-1)\epsilon}$$

Thus a candidate set to consider is the set of these branches and their children:

$$\mathcal{B}(\epsilon) = \left\{ l : \mathcal{S}(l) \leq \frac{(\lambda + (C-1)\epsilon)\,\mathcal{S}(T^*) + 1 - \lambda - \mathcal{H}(T^*) + 2\epsilon}{\lambda - (C-1)\epsilon} \right\}$$

$$\mathcal{B}'(\epsilon) = \mathcal{B}(\epsilon) \bigcup_{l \in \mathcal{B}(\epsilon), i \in \mathcal{A}(l) \setminus \{\overline{a}\}} \mathrm{Ch}(l, i)$$

However, even if $\Gamma(\epsilon) = \bigcap_{l \in \mathcal{B}'(\epsilon)} \mathcal{A}(l, \epsilon)$ happens, this does not guarantee a priori that all expanded branches and their children are well concentrated. Fortunately, this guarantee turns out to be true as we prove in Lemma 32. This in turn implies that the maximum number of expanded branches cannot exceed $|\mathcal{B}(\epsilon)|$ under the event $\Gamma(\epsilon)$. On the other hand, we first observe n samples from the stream, then at each Expansion step we observe one more sample and update the sliding window. Therefore, the total number of observed samples is equal to n plus the number of expanded branches. Hence, under event $\Gamma(\epsilon)$ we have:

$$N(q, C, \lambda, n) \leq n + |\mathcal{B}(\epsilon)|$$

Moreover, for an adequate choice of ϵ the event $\Gamma(\epsilon)$ also implies optimality as we prove in Lemma 33. Therefore:

$$\Gamma(\epsilon) \implies \widehat{\pi} = \pi^* \text{ and } N(q, C, \lambda, n) \leq n + |\mathcal{B}(\epsilon)|$$

This in turn implies that:

$$\begin{aligned}
\mathbb{P}\left[\widehat{\pi} = \pi^*, N(q, C, \lambda, n) \leq n + |\mathcal{B}(\epsilon)|\right] &\geq \mathbb{P}\left[\Gamma(\epsilon)\right] \\
&\geq \mathbb{P}\left[\bigcap_{l \in \mathcal{B}'(\epsilon)} \mathcal{A}(l, \epsilon)\right] \\
&\geq 1 - \sum_{l \in \mathcal{B}'(\epsilon)} \mathbb{P}\left[\mathcal{A}(l, \epsilon)^c\right]
\end{aligned}$$

where the last inequality is due to a Union Bound. Using Hoeffding's inequality, Lemma 26, and calculating $|\mathcal{B}(\epsilon)|$ yield Theorem 28.

7.5 Experiments

We evaluate the empirical performance of Online-Branches on two sets of experiments. The first experimental setup is similar to the one we employed in Chapter 5 and Chapter 6. We compare Online-Branches with VFDT, HAT, EFDT and SGT on the data streams introduced in Table 5.1. This time however, we fix the size of the sliding window n for each stream according to Table 7.1. Moreover, since Online-Branches terminates, we also report the number of samples and the time it took to reach optimality. These are summarised in Table 7.2, Fig. 7.1 and Fig. 7.2; Table 7.3 limits the comparison to the MCTS methods. In our second set of experiments, we analyse the dependence of optimality, in terms of the accuracy of the final proposed DT solution and its number of splits, on the window size n. Fig. 7.3 and Fig. 7.4 provide quartile plots (also commonly referred to as box plots) of these quantities.

Table 7.2, Fig. 7.1 and Fig. 7.2 show that Online-Branches outperforms the greedy algorithms on all the experiments. It reaches optimality using far fewer samples than the 2×10^5 limit and in small execution times as well. The slowest displayed run pertains to Random-Trees 3 and XOR-4, which is natural since these settings are very challenging even for the MCTS methods,

Table 7.1: Number of features q, number of categories per feature C, number of classes K and penalty parameter λ for the different data streams we used.

Stream	q	C	K	λ	n
led	7	2	10	0.01	1000
led-irrelevant	24	2	10	0.01	1000
random-tree-1	10	2	2	0.001	10000
random-tree-2	10	4	2	0.001	10000
random-tree-3	20	5	2	0.001	10000
stagger	3	3	2	0.001	1000
xor-1	50	2	2	0.01	10000
xor-2	100	2	2	0.01	10000
xor-3	200	2	2	0.01	10000
xor-4	500	2	2	0.01	10000

Table 7.2: Comparing Online-Branches with VFDT, HAT, EFDT and SGT on several data streams from River.

Stream	**VFDT**		**EFDT**		**SGT**		**HAT**		**Online-Branches**			
	accuracy	splits	accuracy	splits	accuracy	splits	accuracy	splits	accuracy	splits	samples	time (s)
led	**1**	**9**	0.94	8	0.14	6	**1**	**9**	**1**	**9**	1233	0.09
led-irrelevant	**1**	**9**	**1**	**9**	0.69	10	**1**	**9**	**1**	**9**	1730	1.24
random-tree-1	**1**	35	0.93	27	0.13	598	**1**	35	**1**	**21**	11996	2.17
random-tree-2	0.89	75	0.95	117	0.77	10	0.91	87	**0.89**	**70**	25114	16.69
random-tree-3	0.84	115	0.85	115	0.65	20	0.79	5	**0.88**	**50**	132595	215
stagger	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	1002	0.01
xor-1	**1**	**3**	0.45	7	0.57	21	0.61	64	**1**	**3**	10053	2.91
xor-2	0.45	67	0.8	5	0.48	31	0.54	65	**1**	**3**	10103	8.23
xor-3	0.50	61	0.55	3	0.53	32	0.51	63	**1**	**3**	10202	27.77
xor-4	0.54	31	0.49	3	0.51	16	0.45	31	**1**	**3**	10503	154

as depicted in Table 7.3. The advantages of Online-Branches over the greedy methods are clear but it still suffers from a similar drawback to MCTS, namely the slow treatment of data compared to the greedy methods. In the following points, we compare Online-Branches with our previous MCTS algorithms:

- In terms of optimality of the proposed solution, Table 7.3 shows that Online-Branches outperforms or performs at least as good as the MCTS methods.
- Online-Branches never reaches the timeout limit of 5 minutes, while UCDT, EGDT and TSDT do. However, we cannot really compare the runtimes of these methods since the MCTS algorithms do not have a termination condition. A tempting test would be to compare the time it takes each one of these methods to reach the optimal solution for the first time. However, there is a significant difference between reaching the optimal solution for the first time, and proving its optimality. Indeed, as argued by Hu et al. (2019), an algorithm could quickly stumble upon the optimal solution, if only by chance, yet it would

Table 7.3: Comparing Online-Branches with our MCTS methods UCDT, EGDT and TSDT.

Stream	UCDT		EGDT		TSDT		Online-Branches			
	accuracy	splits	accuracy	splits	accuracy	splits	accuracy	splits	samples	time (s)
led	**1**	**9**	**1**	**9**	**1**	**9**	**1**	**9**	1233	0.09
led-irrelevant	**1**	**9**	**1**	**9**	**1**	**9**	**1**	**9**	1730	1.24
random-tree-1	0.873	13	0.881	13	0.960	23	**1**	**21**	11996	2.17
random-tree-2	0.847	5	0.841	4	0.850	12	**0.89**	**70**	25114	16.69
random-tree-3	0.760	4	0.787	6	0.740	4	**0.88**	**50**	132595	215
stagger	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**	1002	0.01
xor-1	**1**	**3**	**1**	**3**	**1**	**3**	**1**	**3**	10053	2.91
xor-2	**1**	**3**	**1**	**3**	**1**	**3**	**1**	**3**	10103	8.23
xor-3	**1**	**3**	**1**	**3**	**1**	**3**	**1**	**3**	10202	27.77
xor-4	**1**	**3**	0.49	3	0.440	3	**1**	**3**	10503	154

take a substantial amount of time proving optimality. Such comparison would therefore be irrelevant.

- On one hand Online-Branches requires fixing a new hyperparameter, the size n of the sliding window. While this seems like a drawback compared to the MCTS methods, we recall that the latter require defining the exploration-exploitation hyperparameter γ, unlike Online-Branches. We argue that setting a value for n is a much easier task than setting a value for γ. From a pure optimality perspective, the larger n is, the higher the probability that Online-Branches terminates optimally. The only downside of large n values is that they make the algorithm more computationally costly. These considerations are not available for γ on the other hand.

Our second set of experiments analyses the dependence of optimality and the number of samples to reach it on n. From Fig. 7.3 and Fig. 7.4, it is evident that $n = 10000$ is optimal significantly more often than the smaller n values, with the smallest value $n = 100$ exhibiting the widest range of uncertainty about its performance. An analysis that is limited to optimality would always favour larger values of n. However, when the number of consumed samples (or equivalently the execution times) is taken into consideration, then medium values of n strike a better balance between this metric and optimality. On many data streams, $n = 1000$ displays a similar optimality performance as $n = 10000$ all while requiring significantly less observed samples, even on the hard XOR-4 setting. Nonetheless, the Random-Trees experiments pose a difficult challenge for $n = 1000$, where its performance, in terms of optimality, is far less satisfactory than that of $n = 10000$. Thus, depending on the problem at hand, a practitioner should choose the largest value of n they can tolerate, then slowly change it to smaller values if the algorithm consumes more data than anticipated or tolerated.

7.6 Conclusion

In our earlier attempts at deriving optimal sparse DTs in the online setting, we derived UCDT, EGDT and TSDT, three MCTS algorithms. These methods enjoy an optimal asymptotic convergence property, and they have been shown to outperform the state of the art online DTs from

the River library. Yet, as we have discussed in Chapter 5 and Chapter 6, UCDT, EGDT and TSDT lack a finite-time analysis for assessing their performances given a set number of observed samples.

Performing such finite-time analysis was the main objective of this chapter. We could not derive these results for our earlier MCTS methods, and for this reason, we devised a new method that generalises Branches to handling data streams based on a sliding window of fixed size. When it terminates, the induced algorithm, Online-Branches, returns the true optimal sparse DT with a high probability that vanishes with the size of the sliding window. These theoretical findings are validated through an empirical evaluation where we notice that, indeed, larger sizes of the sliding window induce higher frequencies of optimal termination. Moreover, we also showed that Online-Branches outperforms UCDT, EGDT and TSDT.

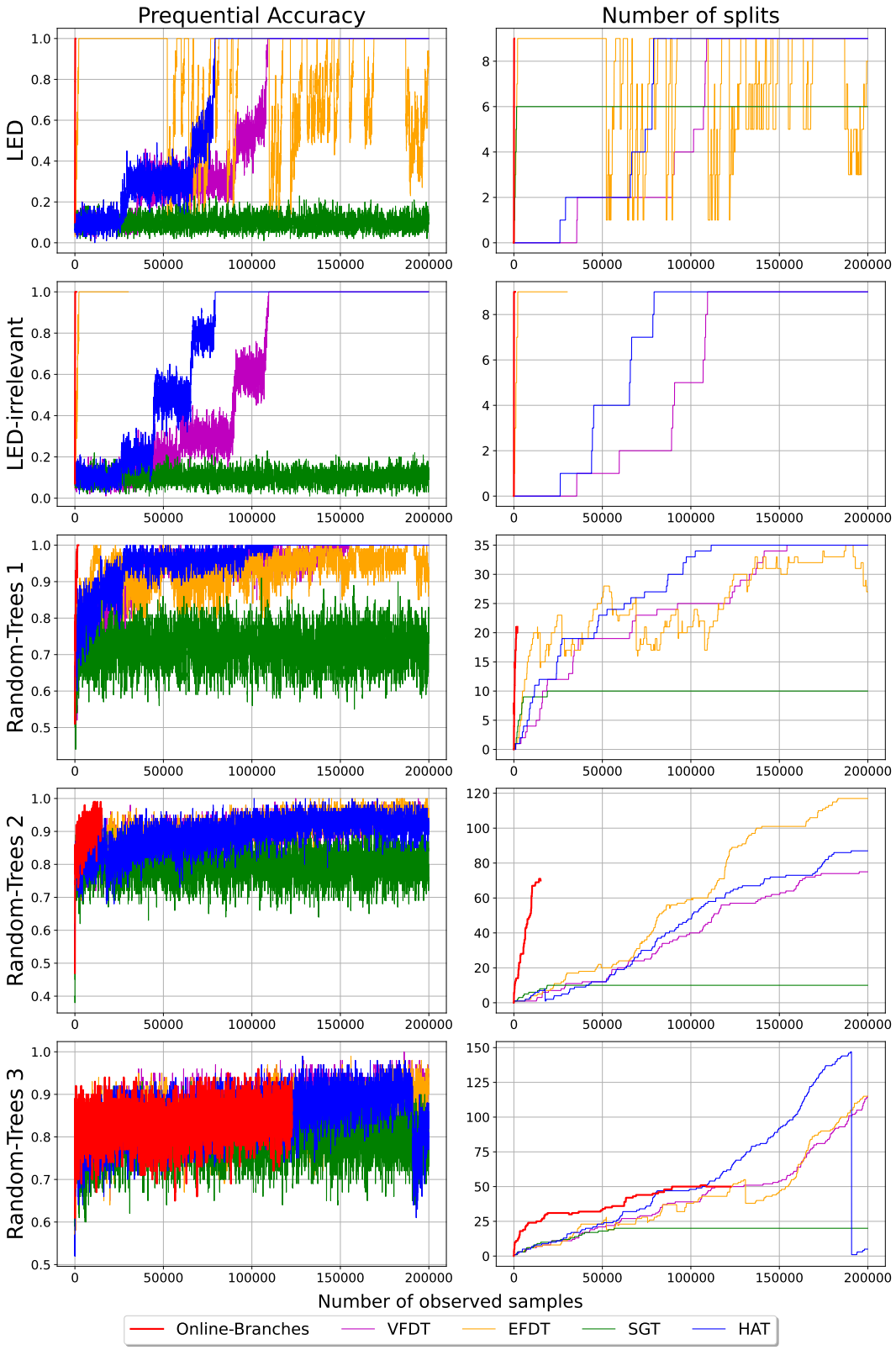


Figure 7.1: The left column reports the prequential accuracy of each method as a function of the number of observed samples so far. The right column reports the number of splits of the solution for each algorithm as a function of the number of observed samples so far.

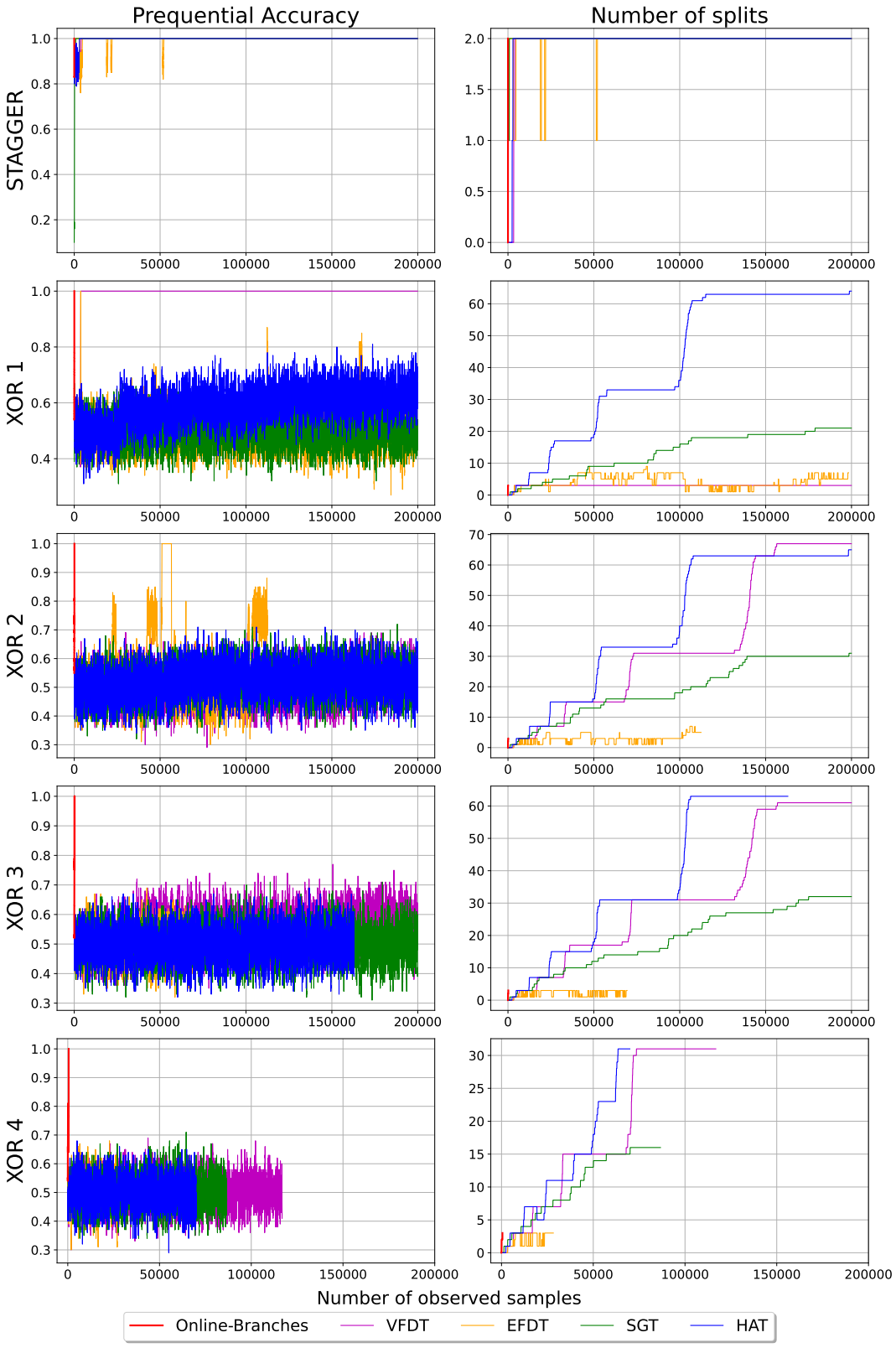


Figure 7.2: The remaining experimental results.

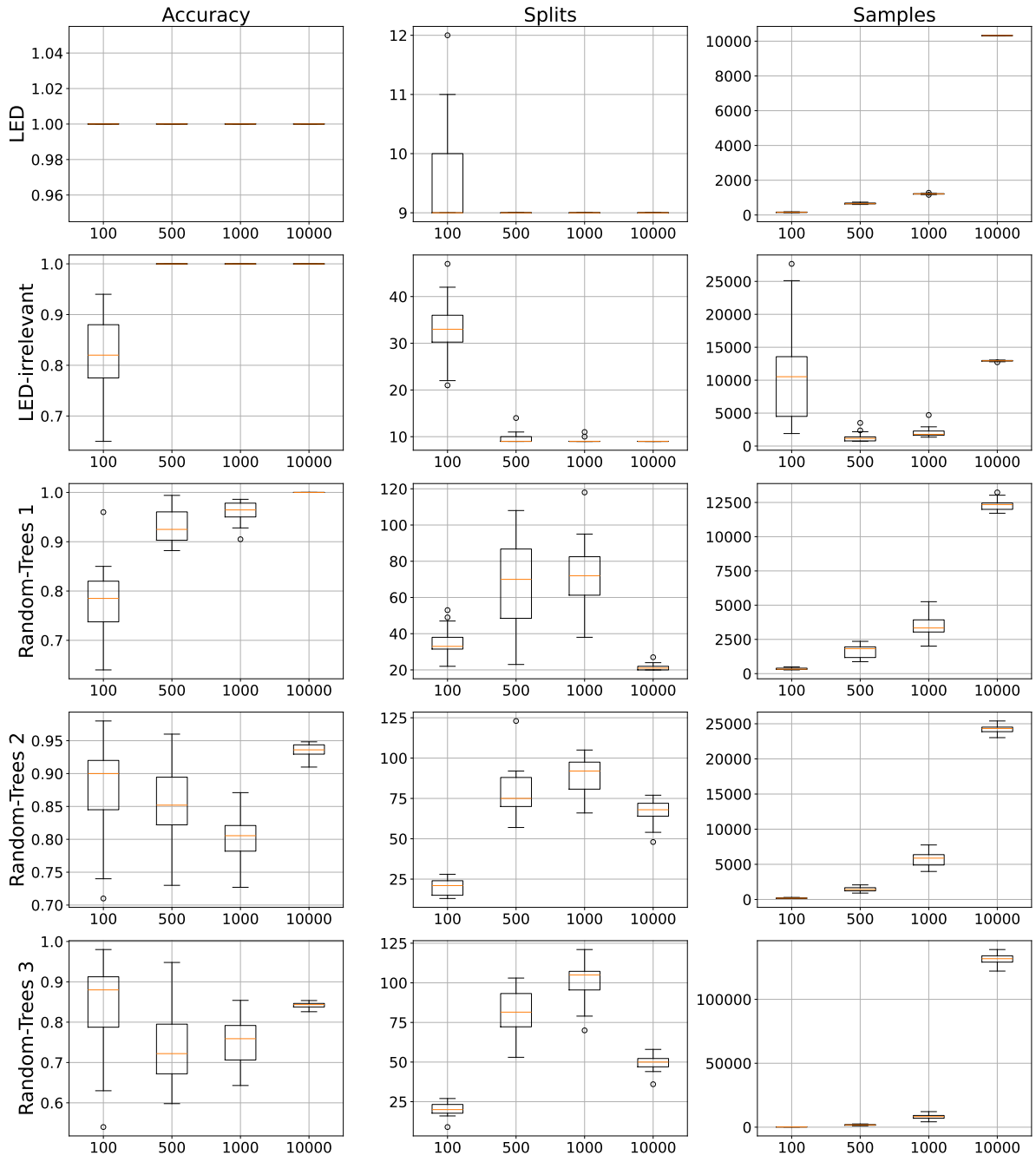


Figure 7.3: Quartile plots of the accuracy (of the final proposed solution), the number of splits and the number of samples it took to terminate, for different values of the window size $n \in \{100, 500, 1000, 10000\}$.

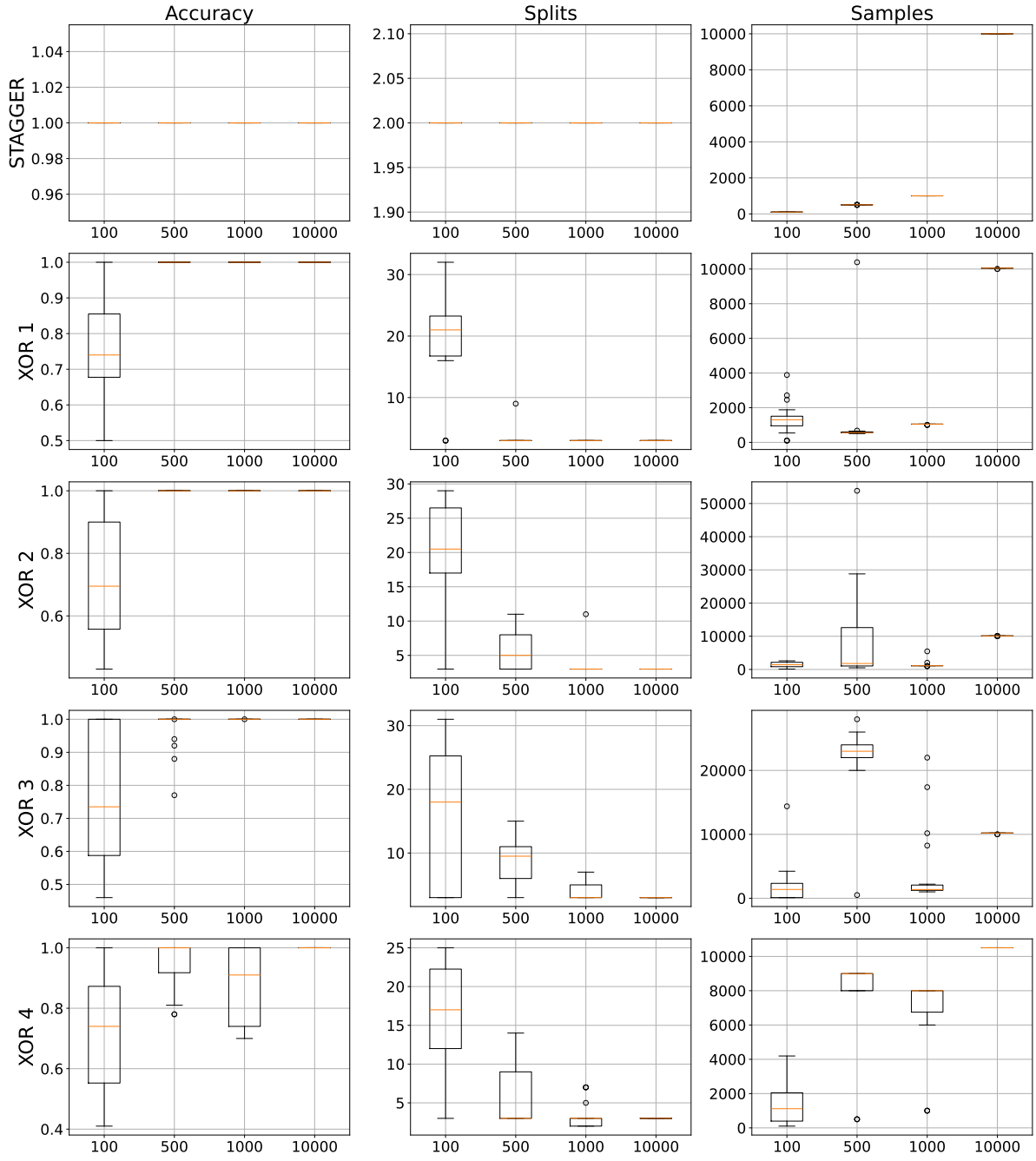


Figure 7.4: The remaining quartile plots.

7.7 Appendix: Proofs

Lemma 26. *Let l be a branch and denote the probabilities:*

$$\forall 1 \leq k \leq K : p(l,k) = \mathbb{P}[l(X) = 1, Y = k]$$

Then for any $0 < \delta < 1$, $r(l,\overline{a})$ satisfies the following concentration inequality:

$$\mathbb{P}\left[\left|r(l,\overline{a}) - \mathcal{H}(l)\right| \leq \sqrt{\frac{\log\frac{2}{\delta}}{2n}}\right] \geq 1 - \delta - 2\sum_{k \neq k^*(l)} \exp\left\{-\frac{n\left[p(l,k^*(l)) - p(l,k)\right]^2}{2}\right\}$$

Proof We first note that:

$$\begin{aligned} & k^*(l) = \text{Argmax}_{1\leq k\leq K}\mathbb{P}[Y = k | l(X) = 1] \\ \implies & k^*(l) = \text{Argmax}_{1\leq k\leq K}\mathbb{P}[Y = k | l(X) = 1]\,\mathbb{P}[l(X) = 1] \\ \implies & k^*(l) = \text{Argmax}_{1\leq k\leq K} p(l,k) \end{aligned}$$

Now let $\epsilon > 0$ and let us analyse the term $\mathbb{P}\left[\left|r(l,\overline{a}) - \mathcal{H}(l)\right| \leq \epsilon\right]$:

$$\mathbb{P}\left[\left|r(l,\overline{a}) - \mathcal{H}(l)\right| \leq \epsilon\right] = \mathbb{P}\left[\left|\frac{1}{n}\max_{1\leq k\leq K}\sum_{t=1}^{n} l(X_t)\mathbb{1}\{Y_t = k\} - \mathcal{H}(l)\right| \leq \epsilon\right]$$

To make the notation lighter, let us denote $\forall 1 \leq k \leq K : Z_k^{(n)} = \sum_{t=1}^{n} l(X_t)\mathbb{1}\{Y_t = k\}$

$$\begin{aligned} \mathbb{P}\left[\left|r(l,\overline{a}) - \mathcal{H}(l)\right| \leq \epsilon\right] &= \mathbb{P}\left[\left|\frac{1}{n}\max_{1\leq k\leq K} Z_k^{(n)} - \mathcal{H}(l)\right| \leq \epsilon\right] \\ &\geq \mathbb{P}\left[\left|\frac{1}{n}\max_{1\leq k\leq K} Z_k^{(n)} - \mathcal{H}(l)\right| \leq \epsilon, k^*(l) = \text{Argmax}_{1\leq k\leq K} Z_k^{(n)}\right] \\ &\geq \mathbb{P}\left[\left|\frac{Z_{k^*(l)}^{(n)}}{n} - \mathcal{H}(l)\right| \leq \epsilon, k^*(l) = \text{Argmax}_{1\leq k\leq K} Z_k^{(n)}\right] \\ \implies \mathbb{P}\left[\left|r(l,\overline{a}) - \mathcal{H}(l)\right| \leq \epsilon\right] &\geq 1 - \mathbb{P}\left[\left|\frac{Z_{k^*(l)}^{(n)}}{n} - \mathcal{H}(l)\right| > \epsilon\right] - \mathbb{P}\left[k^*(l) \neq \text{Argmax}_{1\leq k\leq K} Z_k^{(n)}\right] \end{aligned} \tag{7.4}$$

Let us analyse the term $\mathbb{P}\left[\left|\frac{Z_{k^*(l)}^{(n)}}{n} - \mathcal{H}(l)\right| > \epsilon\right]$:

$$\begin{aligned} \mathbb{E}\left[\frac{Z_{k^*(l)}^{(n)}}{n}\right] &= \mathbb{E}[l(X)\mathbb{1}\{Y = k^*(l)\}] = \mathbb{E}[\mathbb{E}[l(X)\mathbb{1}\{Y = k^*(l)\}|X]] \\ &= \mathbb{P}[l(X) = 1]\,\mathbb{P}[Y = k^*(l)\,|l(X) = 1] = \mathcal{H}(l) \end{aligned}$$

On the other hand, since the variables $\{l(X_t)\mathbb{1}\{Y_t = k^*(l)\}\}_{t=1}^n$ are independent, we use Ho-

effding's inequality to get:

$$\begin{aligned}
&\mathbb{P}\left[\left|\frac{Z_{k^*(l)}^{(n)}}{n}-\mathcal{H}\left(l\right)\right|>\epsilon\right]=\mathbb{P}\left[\left|\frac{Z_{k^*(l)}^{(n)}}{n}-\mathbb{E}\left[\frac{Z_{k^*(l)}^{(n)}}{n}\right]\right|>\epsilon\right]\\
\implies &\mathbb{P}\left[\left|\frac{Z_{k^*(l)}^{(n)}}{n}-\mathcal{H}\left(l\right)\right|>\epsilon\right]\leq 2\exp\left(-2n\epsilon^2\right)
\end{aligned}\tag{7.5}$$

Now let us analyse the second term $\mathbb{P}\left[k^*\left(l\right)\neq \text{Argmax}_{1\leq k\leq K}Z_k^{(n)}\right]$:

$$\mathbb{P}\left[k^*\left(l\right)\neq \text{Argmax}_{1\leq k\leq K}Z_k^{(n)}\right]\leq \sum_{k\neq k^*(l)}\mathbb{P}\left[k\neq \text{Argmax}_{1\leq j\leq K}Z_j^{(n)}\right]$$

Let $k\neq k^*\left(l\right)$, we have the following:

$$\mathbb{P}\left[k\neq \text{Argmax}_{1\leq j\leq K}Z_j^{(n)}\right]\leq \mathbb{P}\left[Z_k^{(n)}\geq Z_{k^*(l)}^{(n)}\right]$$

Consider a positive number $\epsilon_k>0$, we have:

$$\begin{aligned}
\mathbb{P}\left[Z_k^{(n)}\geq Z_{k^*(l)}^{(n)}\right]&=\mathbb{P}\left[Z_k^{(n)}\geq Z_{k^*(l)}^{(n)},\frac{Z_{k^*(l)}}{n}-p\left(l,k^*\left(l\right)\right)\geq-\epsilon_k\right]\\
&+\mathbb{P}\left[Z_k^{(n)}\geq Z_{k^*(l)}^{(n)},\frac{Z_{k^*(l)}}{n}-p\left(l,k^*\left(l\right)\right)<-\epsilon_k\right]\\
&\leq\mathbb{P}\left[Z_{k^*(l)}^{(n)}\leq Z_k^{(n)},\frac{Z_{k^*(l)}}{n}-p\left(l,k^*\left(l\right)\right)\geq-\epsilon_k\right]\\
&+\mathbb{P}\left[\frac{Z_{k^*(l)}}{n}-p\left(l,k^*\left(l\right)\right)<-\epsilon_k\right]
\end{aligned}$$

Take $\epsilon_k=\frac{p(l,k^*(l))-p(l,k)}{2}>0$, this yields:

$$\begin{aligned}
\mathbb{P}\left[Z_k^{(n)}\geq Z_{k^*(l)}^{(n)}\right]&\leq\mathbb{P}\left[p\left(l,k^*\left(l\right)\right)-\epsilon_k\leq\frac{Z_k^{(n)}}{n}\right]+\mathbb{P}\left[\frac{Z_{k^*(l)}^{(n)}}{n}\leq p\left(l,k^*\left(l\right)\right)-\epsilon_k\right]\\
&\leq\mathbb{P}\left[p\left(l,k\right)+\epsilon_k\leq\frac{Z_k^{(n)}}{n}\right]+\mathbb{P}\left[\frac{Z_{k^*(l)}^{(n)}}{n}\leq p\left(l,k^*\left(l\right)\right)-\epsilon_k\right]
\end{aligned}$$

By definition, we have $\forall 1\leq j\leq K:p\left(l,j\right)=\mathbb{E}\left[l\left(X\right)\mathbb{1}\{Y=j\}\right]$, therefore, using Hoeffding's inequality, we get:

$$\begin{aligned}
&\mathbb{P}\left[Z_k^{(n)}\geq Z_{k^*(l)}^{(n)}\right]\leq 2\exp\left(-2n\epsilon_k^2\right)\\
\implies&\mathbb{P}\left[Z_k^{(n)}\geq Z_{k^*(l)}^{(n)}\right]\leq 2\exp\left(-\frac{n\left[p\left(l,k^*\left(l\right)\right)-p\left(l,k\right)\right]^2}{2}\right)\\
\implies&\mathbb{P}\left[k\neq \text{Argmax}_{1\leq j\leq K}Z_j^{(n)}\right]\leq 2\sum_{k\neq k^*(l)}\exp\left(-\frac{n\left[p\left(l,k^*\left(l\right)\right)-p\left(l,k\right)\right]^2}{2}\right)
\end{aligned}\tag{7.6}$$

From Eq. (7.4), Eq. (7.5) and Eq. (7.6) we get:

$$\mathbb{P}\left[\left|r\left(l,\overline{a}\right)-\mathcal{H}\left(l\right)\right|\leq\epsilon\right]\geq 1-2\exp\left(-2n\epsilon^2\right)-2\sum_{k\neq k^*(l)}\exp\left(-\frac{n\left[p\left(l,k^*\left(l\right)\right)-p\left(l,k\right)\right]^2}{2}\right)$$

By setting $\delta = 2\exp\left(-2n\epsilon^2\right) \in]0,1[$, we get the statement of the Lemma. ■

Corollary 27. *Let π be a policy and l a branch, then for any $0 < \delta < 1$, there exists $n_0 \geq 1$ sufficiently large such that for any $n \geq n_0$ the return $\mathcal{R}^\pi(l)$ concentrates around $\mathcal{H}_\lambda(T^\pi(l))$ as follows:*

$$\mathbb{P}\left[\left|\mathcal{R}^\pi(l) - \mathcal{H}_\lambda(T^\pi(l))\right| \leq \sqrt{\frac{|T^\pi(l)|^2 \log\frac{2|T^\pi(l)|}{\delta}}{2n}}\right] \geq 1 - \delta -$$
$$2\sum_{l\in T^\pi(l)}\sum_{k\neq k^*(l)} \exp\left\{-\frac{n\left[p(l,k^*(l)) - p(l,k)\right]^2}{2}\right\}$$

Proof Let $T_0 = l$ and $\forall t \geq 0 : T_t \xrightarrow{\pi(T_t)} T_{t+1}$, then the return of π from l is:

$$\mathcal{R}^\pi(l) = \sum_{t=0}^{\infty} r(T_t, \pi(T_t)) = -\lambda\mathcal{S}(T^\pi(l)) + \sum_{l\in T^\pi(l)} r(l,\overline{a})$$
$$\implies \left|\mathcal{R}^\pi(l) - \mathcal{H}_\lambda(T^\pi(l))\right| = \left|\sum_{l\in T^\pi(l)}\left[r(l,\overline{a}) - \mathcal{H}(l)\right]\right|$$
$$\leq \sum_{l\in T^\pi(l)}\left|r(l,\overline{a}) - \mathcal{H}(l)\right|$$

For any $\epsilon > 0$ we have:

$$\mathbb{P}\left[\left|\mathcal{R}^\pi(l) - \mathcal{H}_\lambda(T^\pi(l))\right| \leq \epsilon\right] \geq \mathbb{P}\left[\sum_{l\in T^\pi(l)}\left|r(l,\overline{a}) - \mathcal{H}(l)\right| \leq \epsilon\right]$$
$$\geq \mathbb{P}\left[\forall l \in T^\pi(l) : \left|r(l,\overline{a}) - \mathcal{H}(l)\right| \leq \frac{\epsilon}{|T^\pi(l)|}\right]$$
$$\geq 1 - \sum_{l\in T^\pi(l)}\left\{2\exp\left(\frac{-2n\epsilon^2}{|T^\pi(l)|^2}\right) + \right.$$
$$\left. 2\sum_{k\neq k^*(l)}\exp\left(-\frac{n\left[p(l,k^*(l)) - p(l,k)\right]^2}{2}\right)\right\}$$
$$\geq 1 - 2|T^\pi(l)|\exp\left(\frac{-2n\epsilon^2}{|T^\pi(l)|^2}\right) -$$
$$2\sum_{l\in T^\pi(l)}\sum_{k\neq k^*(l)}\exp\left(-\frac{n\left[p(l,k^*(l)) - p(l,k)\right]^2}{2}\right)$$

The third line is due to a union bound and Lemma 26. Now, by setting $\delta = 2|T^\pi(l)|\exp\left(\frac{-2n\epsilon^2}{|T^\pi(l)|^2}\right)$ and taking $n_0 = \left\lceil\frac{|T^\pi(l)|^2\log 2|T^\pi(l)|}{2\epsilon^2}\right\rceil$ we get the result of the Theorem. ■

Lemma 29. *We introduce the following notation:*

$$\begin{cases} \Omega = \widetilde{T}_0 \xrightarrow{\pi^*} \ldots \xrightarrow{\pi^*} \widetilde{T}_{\tau^*} = T^* \\ \forall 0 \leq t \leq \tau^* : \widetilde{T}_t = \left\{ l_{t,1}, \ldots, l_{t,|\widetilde{T}_t|} \right\} \end{cases}$$

A branch l is expanded only if there exists a DT T and a DT $\widetilde{T} = \left\{ \widetilde{l}_1, \ldots, \widetilde{l}_{|\widetilde{T}|} \right\}$ such that:

$l \in T$

$\forall 1 \leq v \leq |\widetilde{T}|, \exists 0 \leq t \leq \tau^*, 1 \leq u \leq |\widetilde{T}_t| : \widetilde{l}_v = \widetilde{l}_{t,u}$

$$-\lambda \mathcal{S}(T) + \sum_{l' \in T} \max \left\{ r(l', \overline{a}), -\lambda + \frac{n(l')}{n} \right\} \geq -\lambda \mathcal{S}\left(\widetilde{T}\right) + \sum_{l' \in \widetilde{T}} \max \left\{ r(l', \overline{a}), -\lambda + \frac{n(l')}{n} \right\}$$

Proof Let π be the Selection policy, i.e. for any unit-state l:

$$\pi(l) = \begin{cases} \overline{a} \text{ If } l \text{ has not been expanded yet.} \\ \text{Argmax}_{a \in \mathcal{A}(l)} \mathcal{Q}(l, a) \text{ Otherwise.} \end{cases}$$

At the current iteration, let $\mathcal{E}$ be the set of branches that have been expanded and:

$$\mathcal{E}' = \mathcal{E} \bigcup_{l \in \mathcal{E}, i \in \mathcal{A}(l) \setminus \{\overline{a}\}} \text{Ch}(l, i)$$

$\mathcal{E}'$ is the set of expanded branches at the current iteration, along with their children. This means that, at the current iteration, only branches in $\mathcal{E}' \setminus \mathcal{E}$ can be considered for expansion. Consider a branch $l \in \mathcal{E}' \setminus \mathcal{E}$. By the definition of π, l can only be expanded if $l \in T^\pi$. Moreover, for any DT T constructed with branches in $\mathcal{E}' \setminus \mathcal{E}$, we have:

$$\mathcal{R}(T) \leq \mathcal{R}(T^\pi)$$

Take $\widetilde{T} = \left(\bigcup_{0 \leq t \leq \tau^*, 1 \leq u \leq |\widetilde{T}_t|} \{l_{t,u}\} \right) \cap (\mathcal{E}' \setminus \mathcal{E})$ the DT constructed with branches leading to T^* that are also in $\mathcal{E}' \setminus \mathcal{E}$. Since $\widetilde{T}$ is constructed with branches in $\mathcal{E}' \setminus \mathcal{E}$, then we have:

$$\begin{aligned} & \mathcal{R}\left(\widetilde{T}\right) \leq \mathcal{R}(T^\pi) \\ \implies & -\lambda \mathcal{S}(T^\pi) + \sum_{l' \in T^\pi} \max \left\{ r(l', \overline{a}), -\lambda + \frac{n(l')}{n} \right\} \geq -\lambda \mathcal{S}\left(\widetilde{T}\right) + \sum_{l' \in \widetilde{T}} \max \left\{ r(l', \overline{a}), -\lambda + \frac{n(l')}{n} \right\} \end{aligned}$$

■

Lemma 30. *For a classification problem where all features have the same number of possible categories $C \geq 2$, any DT T satisfies the following:*

$$|T| = \mathcal{S}(T)(C - 1) + 1$$

Proof Let $f(n)$ be the number of branches in a DT with n splits. A DT with no split has only one branch, thus $f(0) = 1$. On the other hand, when splitting a branch in a DT with n splits,

the resulting DT has $n+1$ splits, C additional branches and one less (the branch that has been split). Thus $f(n+1) = f(n) - 1 + C$, which leads to:

$$\begin{aligned} f(n) - f(0) &= \sum_{k=1}^{n} \Big\{ f(k) - f(k-1) \Big\} = n(C-1) \\ \implies f(n) &= n(C-1) + 1 \end{aligned}$$

Writing this down with our conventional notation, we get:

$$|T| = \mathcal{S}(T)(C-1) + 1$$

■

Lemma 31. *For any branch l and $\epsilon > 0$, we define the following event:*

$$\mathcal{A}(l,\epsilon) = \left\{ \left| r(l,\overline{a}) - \mathcal{H}(l) \right| \leq \epsilon, \left| \frac{n(l)}{n} - p(l) \right| \leq \epsilon \right\}$$

Let $\mathcal{E}$ be the set of all expanded branches and:

$$\mathcal{E}' = \mathcal{E} \bigcup_{l \in \mathcal{E}, i \in \mathcal{A}(l) \setminus \{\overline{a}\}} Ch(l,i)$$

If the event $\bigcap_{l \in \mathcal{E}'} \mathcal{A}(l,\epsilon) \bigcap_{0 \leq t \leq \tau^*, 1 \leq u \leq |\widetilde{T}_t|} \mathcal{A}(l_{t,u},\epsilon)$ happens, then a branch l can be expanded only if:

$$\mathcal{S}(l) \leq \frac{(\lambda + (C-1)\epsilon)\mathcal{S}(T^*) + 1 - \lambda - \mathcal{H}(T^*) + 2\epsilon}{\lambda - (C-1)\epsilon}$$

where $l_{t,u}$ are defined in Lemma 29 and $0 < \epsilon < \frac{\lambda}{C-1}$.

Proof Suppose the event $\bigcap_{l \in \mathcal{E}'} \mathcal{A}(l,\epsilon) \bigcap_{0 \leq t \leq \tau^*, 1 \leq u \leq |\widetilde{T}_t|} \mathcal{A}(l_{t,u},\epsilon)$ happens, and let l be a branch that is expanded. According to Lemma 29, there exists a DT T such that $l \in T$ and:

$$-\lambda \mathcal{S}(T) + \sum_{l' \in T} \max \left\{ r(l',\overline{a}), -\lambda + \frac{n(l')}{n} \right\} \geq -\lambda \mathcal{S}\left(\widetilde{T}\right) + \sum_{l' \in \widetilde{T}} \max \left\{ r(l',\overline{a}), -\lambda + \frac{n(l')}{n} \right\}$$

Define $L = \left\{ l' \in T : r(l',\overline{a}) \geq -\lambda + \frac{n(l')}{n} \right\}$, then we have:

$$-\lambda \mathcal{S}(T) + \sum_{l' \in L} r(l',\overline{a}) + \sum_{l' \in T \setminus L} \left[-\lambda + \frac{n(l')}{n} \right] \geq -\lambda \mathcal{S}\left(\widetilde{T}\right) + \sum_{l' \in \widetilde{T}} \max \left\{ r(l',\overline{a}), -\lambda + \frac{n(l')}{n} \right\}$$

Moreover, from the proof of Lemma 29, T is constructed with branches in $\mathcal{E}'$. Therefore, under

the event $\bigcap_{l\in\mathcal{E}'}\mathcal{A}(l,\epsilon)\bigcap_{0\leq t\leq\tau^*,1\leq u\leq|\widetilde{T}_t|}\mathcal{A}(l_{t,u},\epsilon)$, we get the following:

$$
\begin{aligned}
&-\lambda\mathcal{S}(T)+\sum_{l'\in L}\mathcal{H}(l')-\lambda|T\setminus L|+\sum_{l'\in T\setminus L}p(l')+|T|\epsilon\\
&\geq-\lambda\mathcal{S}\left(\widetilde{T}\right)+\sum_{l'\in\widetilde{T}}\max\{\mathcal{H}(l'),-\lambda+p(l')\}-|\widetilde{T}|\epsilon\\
&\implies-\lambda\mathcal{S}(T)+\sum_{l'\in L}p(l')-\lambda|T\setminus L|+\sum_{l'\in T\setminus L}p(l')+|T|\epsilon\geq-\lambda\mathcal{S}(T^*)+\mathcal{H}(T^*)-|\widetilde{T}|\epsilon\\
&\implies-\lambda\mathcal{S}(T)+1-\lambda+|T|\epsilon\geq-\lambda\mathcal{S}(T^*)+\mathcal{H}(T^*)-|T^*|\epsilon
\end{aligned}
$$

The first implication comes from the following facts:

- $\forall l'\in L:\mathcal{H}(l')=\mathbb{P}[l'(X)=1,k^*(l')=Y]\leq\mathbb{P}[l'(X)=1]=p(l')$
- $-\lambda\mathcal{S}\left(\widetilde{T}\right)+\sum_{l'\in\widetilde{T}}\max\{\mathcal{H}(l'),-\lambda+p(l')\}\geq-\lambda\mathcal{S}(T^*)+\mathcal{H}(T^*)$, this is due to the Purification bound introduced in Proposition 7.
- $|\widetilde{T}|\leq|T^*|$ by the definition of $\widetilde{T}$.

The second implication is due to $\sum_{l'\in T}p(l')=1$ and $|T\setminus L|\geq 1$ because $l\in T\setminus L$. Indeed, $l\in T\setminus L$ because l is expanded, if it had been in L then it would not be expanded. Now, we express $|T|$ and $|T^*|$ in terms of $\mathcal{S}(T)$ and $\mathcal{S}(T^*)$ and we get:

$$
\begin{aligned}
&-\lambda\mathcal{S}(T)+1-\lambda+(\mathcal{S}(T)(C-1)+1)\epsilon\geq-\lambda\mathcal{S}(T^*)+\mathcal{H}(T^*)-(\mathcal{S}(T^*)(C-1)+1)\epsilon\\
&\implies\mathcal{S}(l)\leq\mathcal{S}(T)\leq\frac{(\lambda+(C-1)\epsilon)\mathcal{S}(T^*)+1-\lambda-\mathcal{H}(T^*)+2\epsilon}{\lambda-(C-1)\epsilon}
\end{aligned}
$$

■

Lemma 32. *For any $0<\epsilon<\frac{\lambda}{C-1}$ Define the following:*

$$
\begin{aligned}
\mathcal{B}(\epsilon)&=\left\{l:\mathcal{S}(l)\leq\frac{(\lambda+(C-1)\epsilon)\mathcal{S}(T^*)+1-\lambda-\mathcal{H}(T^*)+2\epsilon}{\lambda-(C-1)\epsilon}\right\}\\
\mathcal{B}'(\epsilon)&=\mathcal{B}(\epsilon)\bigcup_{l\in\mathcal{B}(\epsilon),i\in\mathcal{A}(l)\setminus\{\overline{a}\}}Ch(l,i)\\
\Gamma(\epsilon)&=\bigcap_{l\in\mathcal{B}'(\epsilon)}\mathcal{A}(l,\epsilon)
\end{aligned}
$$

where $\mathcal{A}(l,\epsilon)$ is defined in Lemma 31. Moreover, define $N(q,C,\lambda,n)$ the total number of observed samples by Online-Branches (as defined in Theorem 28). Then we have the following:

$$\Gamma(\epsilon)\implies N(q,C,\lambda,n)\leq n+|\mathcal{B}(\epsilon)|$$

Proof We proceed by showing that $\forall l\in\mathcal{B}'(\epsilon)\setminus\mathcal{B}(\epsilon)$ l cannot be expanded. By contradiction, suppose that $\exists l\in\mathcal{B}'(\epsilon)\setminus\mathcal{B}(\epsilon):l$ is expanded. Let $l\in\mathcal{B}'(\epsilon)\setminus\mathcal{B}(\epsilon)$ be the first branch we encounter of this sort and let π be the Selection policy defined in the proof of Lemma 29. Then $T^\pi\in\mathcal{B}'(\epsilon)$ and $l\in T^\pi$. Now, according to Lemma 29, we have:

$$
-\lambda\mathcal{S}(T^\pi)+\sum_{l'\in T^\pi}\max\left\{r(l',\overline{a}),-\lambda+\frac{n(l')}{n}\right\}\geq-\lambda\mathcal{S}\left(\widetilde{T}\right)+\sum_{l'\in\widetilde{T}}\max\left\{r(l',\overline{a}),-\lambda+\frac{n(l')}{n}\right\}
$$

Under the event $\Gamma(\epsilon)$, we get:

$$\begin{aligned}
&-\lambda \mathcal{S}(T^{\pi}) + \sum_{l' \in T^{\pi}} \max\left\{\mathcal{H}(l'), -\lambda + p(l')\right\} + |T^{\pi}|\epsilon \\
\geq &-\lambda \mathcal{S}\left(\widetilde{T}\right) + \sum_{l' \in \widetilde{T}} \max\left\{\mathcal{H}(l'), -\lambda + p(l')\right\} - |\widetilde{T}|\epsilon \\
\geq &-\lambda \mathcal{S}(T^{*}) + \mathcal{H}(T^{*}) - |\widetilde{T}|\epsilon
\end{aligned}$$

The second inequality comes from the definition of $\widetilde{T}$ and the purification bound. With a similar reasoning as the one we followed in the proof of Lemma 31, we get:

$$\mathcal{S}(T) \leq \frac{(\lambda + (C-1)\epsilon)\mathcal{S}(T^{*}) + 1 - \lambda - \mathcal{H}(T^{*}) + 2\epsilon}{\lambda - (C-1)\epsilon}$$

However, since $l \in T^{\pi}$, then $\mathcal{S}(l) \leq \mathcal{S}(T)$. On the other hand, $l \notin \mathcal{B}(\epsilon)$, therefore:

$$\mathcal{S}(T) \geq \mathcal{S}(l) > \frac{(\lambda + (C-1)\epsilon)\mathcal{S}(T^{*}) + 1 - \lambda - \mathcal{H}(T^{*}) + 2\epsilon}{\lambda - (C-1)\epsilon}$$

which is a contradiction. We have just proven that:

$$\Gamma(\epsilon) \implies \forall l \in \mathcal{B}'(\epsilon) \setminus \mathcal{B}(\epsilon) : l \text{ cannot be expanded}$$

We recall that $\mathcal{B}'(\epsilon)$ is the set of all branches with a threshold on their number of splits, along with their children. Thus, for any branch $l \notin \mathcal{B}'(\epsilon)$ to be expanded, its ancestor in $\mathcal{B}'(\epsilon) \setminus \mathcal{B}(\epsilon)$ has to be expanded first. However, we have just shown that this is not possible under event $\Gamma(\epsilon)$, hence yielding:

$$\Gamma(\epsilon) \implies \forall l \notin \mathcal{B}(\epsilon) : l \text{ cannot be expanded}$$

which means that the total number of expanded branches cannot exceed $|\mathcal{B}(\epsilon)|$. Now let us discuss the total number of observed samples $N(q, C, \lambda, n)$. At the start of the algorithm, we initialise the sliding window by observing n samples, then at each Expansion step, we observe a new sample from the stream and update the sliding window. As such, $N(q, C, \lambda, n)$ is always smaller than n plus the total number of expanded branches. As a consequence, we get:

$$\Gamma(\epsilon) \implies N(q, C, \lambda, n) \leq n + |\mathcal{B}(\epsilon)|$$

■

Lemma 33. *Let $\Gamma(\epsilon)$ be the event we defined in Lemma 32. For the choice:*

$$\epsilon = \frac{1}{2}\min\left\{\min_{T \neq T^{*}} \frac{\mathcal{H}_{\lambda}(T^{*}) - \mathcal{H}_{\lambda}(T)}{2 + (C-1)\left[\mathcal{S}(T) + \mathcal{S}(T^{*})\right]}, \frac{\lambda}{C-1}\right\}$$

the event $\Gamma(\epsilon)$ implies optimality, i.e.

$$\Gamma(\epsilon) \implies \widehat{\pi} = \pi^{*}$$

Proof According to Lemma 32, all the proposed solutions have branches in $\mathcal{B}'(\epsilon)$. Let T be such a solution such that $T \neq T^*$, then we have:

$$-\lambda \mathcal{S}(T) + \mathcal{H}(T) + |T|\epsilon \geq -\lambda \mathcal{S}\left(\widetilde{T}\right) + \sum_{l' \in \widetilde{T}} \max\left\{\mathcal{H}(l'), -\lambda + p(l')\right\} - |\widetilde{T}|\epsilon$$

This is because $\mathcal{A}(l, \epsilon)$ happens for all branches in $\widetilde{T}$ and in T due to event $\Gamma(\epsilon)$. Again, by the definition of $\widetilde{T}$ and the Purification bound, we get:

$$\begin{aligned} -\lambda \mathcal{S}(T) + \mathcal{H}(T) + |T|\epsilon &\geq -\lambda \mathcal{S}(T^*) + \mathcal{H}_\lambda(T^*) - |T^*|\epsilon \\ \implies \epsilon &\geq \frac{\mathcal{H}_\lambda(T^*) - \mathcal{H}_\lambda(T)}{|T| + |T^*|} = \frac{\mathcal{H}_\lambda(T^*) - \mathcal{H}_\lambda(T)}{2 + (C-1)\left[\mathcal{S}(T) + \mathcal{S}(T^*)\right]} \end{aligned}$$

Thus by taking:

$$\epsilon < \frac{\mathcal{H}_\lambda(T^*) - \mathcal{H}_\lambda(T)}{2 + (C-1)\left[\mathcal{S}(T) + \mathcal{S}(T^*)\right]}$$

There could be no such suboptimal solution as T. Hence by choosing:

$$\epsilon = \frac{1}{2} \min\left\{\min_{T \neq T^*} \frac{\mathcal{H}_\lambda(T^*) - \mathcal{H}_\lambda(T)}{2 + (C-1)\left[\mathcal{S}(T) + \mathcal{S}(T^*)\right]}, \frac{\lambda}{C-1}\right\}$$

we get:

$$\Gamma(\epsilon) \implies \widehat{\pi} = \pi^*$$

■

Theorem 28. *For an instance of the classification problem with $q \geq 2$ features, $0 < \lambda \leq 1$ the penalty parameter, $C \geq 2$ the number of categories per feature and $n \geq 1$ the size of the sliding window, let $N(q, C, \lambda, n)$ be the total number of observed samples by Online-Branches when it terminates. Then there exists problem-dependent constants $\epsilon > 0, \delta > 0$ such that:*

$$\mathbb{P}\left[\widehat{\pi} = \pi^*, N(q, C, \lambda, n) \leq n + \psi(\kappa, q)\right] \geq 1 - 2\psi(\kappa, q)\left\{2\exp\left(-2n\epsilon^2\right) + K\exp\left(-\frac{n\delta^2}{2}\right)\right\}$$

$$\xrightarrow[n \to \infty]{} 1$$

with the following quantities:

$$\begin{aligned} \psi(\kappa, q) &= \sum_{h=0}^{\kappa} C^h \binom{q}{h} \leq (C+1)^q \\ \kappa &= \min\left\{\left\lfloor \frac{(\lambda + (C-1)\epsilon)\mathcal{S}(T^*) + 1 - \lambda - \mathcal{H}(T^*) + 2\epsilon}{\lambda - (C-1)\epsilon} \right\rfloor + 1, q\right\} \\ \epsilon &= \frac{1}{2} \min\left\{\min_{T \neq T^*} \frac{\mathcal{H}_\lambda(T^*) - \mathcal{H}_\lambda(T)}{2 + (C-1)\left[\mathcal{S}(T) + \mathcal{S}(T^*)\right]}, \frac{\lambda}{C-1}\right\} \\ \delta &= \min_{l, k \neq k^*(l)} \left\{p(l, k^*(l)) - p(l, k)\right\} \end{aligned}$$

Proof For the choice:

$$\epsilon = \frac{1}{2}\min\left\{\min_{T\neq T^*}\frac{\mathcal{H}_\lambda\left(T^*\right)-\mathcal{H}_\lambda\left(T\right)}{2+\left(C-1\right)\left[\mathcal{S}\left(T\right)+\mathcal{S}\left(T^*\right)\right]},\frac{\lambda}{C-1}\right\}$$

Lemma 32 and Lemma 33 yield:

$$\Gamma\left(\epsilon\right)\implies\left\{\widehat{\pi}=\pi^*,N\left(q,C,\lambda,n\right)\leq n+\left|\mathcal{B}\left(\epsilon\right)\right|\right\}$$

Which means that:

$$\mathbb{P}\left[\widehat{\pi}=\pi^*,N\left(q,C,\lambda,n\right)\leq n+\left|\mathcal{B}\left(\epsilon\right)\right|,N\left(q,C,\lambda,n\right)\leq n+\left|\mathcal{B}\left(\epsilon\right)\right|\right]\geq\mathbb{P}\left[\Gamma\left(\epsilon\right)\right]\tag{7.7}$$

Therefore, let us lower bound $\mathbb{P}\left[\Gamma\left(\epsilon\right)\right]$:

$$\begin{aligned}\mathbb{P}\left[\Gamma\left(\epsilon\right)\right]&=\mathbb{P}\left[\bigcap_{l\in\mathcal{B}'\left(\epsilon\right)}\mathcal{A}\left(l,\epsilon\right)\right]\\&\geq 1-\sum_{l\in\mathcal{B}'\left(\epsilon\right)}\mathbb{P}\left[\mathcal{A}\left(l,\epsilon\right)^c\right]\end{aligned}$$

the second line is due to employing a Union bound. We recall that:

$$\mathcal{A}\left(l,\epsilon\right)=\left\{\left|r\left(l,\overline{a}\right)-\mathcal{H}\left(l\right)\right|\leq\epsilon,\left|\frac{n\left(l\right)}{n}-p\left(l\right)\right|\leq\epsilon\right\}$$

Using a Union bound, we get:

$$\mathbb{P}\left[\mathcal{A}\left(l,\epsilon\right)^c\right]\leq\mathbb{P}\left[\left|\frac{n\left(l\right)}{n}-p\left(l\right)\right|>\epsilon\right]+\mathbb{P}\left[\left|r\left(l,\overline{a}\right)-\mathcal{H}\left(l\right)\right|>\epsilon\right]$$

Using Hoeffding's inequality and Lemma 26, we get:

$$\mathbb{P}\left[\mathcal{A}\left(l,\epsilon\right)^c\right]\leq 4\exp\left(-2n\epsilon^2\right)+2\sum_{k\neq k^*\left(l\right)}\exp\left\{-\frac{n\left[p\left(l,k^*\left(l\right)\right)-p\left(l,k\right)\right]^2}{2}\right\}$$

Take:

$$\delta=\min_{l,k\neq k^*\left(l\right)}\left\{p\left(l,k^*\left(l\right)\right)-p\left(l,k\right)\right\}>0$$

This yields:

$$\mathbb{P}\left[\mathcal{A}\left(l,\epsilon\right)^c\right]\leq 4\exp\left(-2n\epsilon^2\right)+2\left(K-1\right)\exp\left\{-\frac{n\delta^2}{2}\right\}$$

As a consequence, we get:

$$\mathbb{P}\left[\Gamma\left(\epsilon\right)\right]\geq 1-4\left|\mathcal{B}'\left(\epsilon\right)\right|\exp\left(-2n\epsilon^2\right)-2\left(K-1\right)\left|\mathcal{B}'\left(\epsilon\right)\right|\exp\left\{-\frac{n\delta^2}{2}\right\}\tag{7.8}$$

Let us now calculate $\left|\mathcal{B}'\left(\epsilon\right)\right|$, we recall that:

$$\begin{aligned}\mathcal{B}'\left(\epsilon\right)&=\mathcal{B}\left(\epsilon\right)\bigcup_{l\in\mathcal{B}\left(\epsilon\right),i\in\mathcal{A}\left(l\right)\setminus\left\{\overline{a}\right\}}\mathrm{Ch}\left(l,i\right)\\\mathcal{B}\left(\epsilon\right)&=\left\{l:\mathcal{S}\left(l\right)\leq\frac{\left(\lambda+\left(C-1\right)\epsilon\right)\mathcal{S}\left(T^*\right)+1-\lambda-\mathcal{H}\left(T^*\right)+2\epsilon}{\lambda-\left(C-1\right)\epsilon}\right\}\\\implies\mathcal{B}'\left(\epsilon\right)&=\left\{l:\mathcal{S}\left(l\right)\leq\frac{\left(\lambda+\left(C-1\right)\epsilon\right)\mathcal{S}\left(T^*\right)+1-\lambda-\mathcal{H}\left(T^*\right)+2\epsilon}{\lambda-\left(C-1\right)\epsilon}+1\right\}\end{aligned}$$

Let us calculate $\left|\mathcal{B}'\left(\epsilon\right)\right|$. For each $l\in\mathcal{B}\left(\epsilon\right)$

- We choose $h = \mathcal{S}(l)$ in $\{0, \dots, \kappa\}$ where:

$$\kappa = \min \left\{ \left\lfloor \frac{(\lambda + (C-1)\,\epsilon)\,\mathcal{S}(T^*) + 1 - \lambda - \mathcal{H}(T^*) + 2\epsilon}{\lambda - (C-1)\,\epsilon} \right\rfloor + 1, q \right\}$$

- For each choice $h \in \{0, \dots, \kappa\}$, there are $\binom{q}{h}$ possible choices of features defining the clauses of l.
- For each feature among the h features constituting the clauses of l, there are C possible values.

Therefore we have:

$$|\mathcal{B}'(\epsilon)| = \sum_{h=0}^{\kappa} C^h \binom{q}{h} = \psi(\kappa, q)$$

Going back to (7.8), we get:

$$\mathbb{P}[\Gamma(\epsilon)] \geq 1 - 2\psi(\kappa, q) \left\{ 2\exp\left(-2n\epsilon^2\right) + K\exp\left(-\frac{n\delta^2}{2}\right) \right\}$$

Going back to (7.7), we get the result of the Theorem. ■

Chapter 8

Conclusion

Our main objective in this thesis was to develop novel and more efficient algorithms seeking optimal sparse Decision Trees in the batch and online settings. To this aim, we formulated the problem within a Reinforcement Learning framework and developed our methods accordingly. This research is important because of the high interpretability of sparse Decision Trees, which is an essential property to have in sensitive domains.

In the batch setting, there have been recent breakthroughs that leverage Dynamic Programming and Branch & Bound techniques, and they indeed provided the first practical algorithms for optimal sparse DTs. Nevertheless, these methods still suffer from significant shortcomings, their search strategies lack in efficiency which induces a poor scalability to large datasets, and they necessitate a preliminary binary encoding of the data, thus accentuating the scalability issue as we have demonstrated in Chapter 4. Moreover, the theoretical analyses of these methods often forgo analysing the number of *computational steps* before terminating, which is an important indicator of computational complexity.

We addressed the concerns above in Chapter 4 where we introduce Branches, the main contribution of this thesis. Central to Branches' efficiency is its AO* search strategy coupled with the Purification Bound and several other heuristics we use in its Selection and Backpropagation procedures. These techniques prune large portions of the search space and focus the search effort on relevant regions. We analysed Branches' computational complexity by bounding the number of branch evaluations it performs before termination, and we compared this bound with a similar result from the literature (Hu et al., 2019, Theorem E.2) showing a large difference in favour of our result. We validated these findings through an extensive empirical analysis showing that Branches outperforms the state of the art frequently in terms of runtime, number of iterations, and anytime behaviour. In addition, we showed both theoretically and empirically how binary encodings can significantly harm runtime performance compared to ordinal encoding. As a consequence, Branches' support for ordinal encoding confers it a significant advantage in terms of scalability, which we illustrated experimentally through substantially smaller runtimes. Branches offers a novel and efficient search strategy for seeking optimal sparse Decision Trees, and future work based on Branches (of which we summarised several ideas in Section 4.9) can potentially lead to breakthroughs in the scalability and practicality, thus advancing the field of Interpretable Machine Learning.

Unlike the batch setting, optimal sparse Decision Trees have rarely been discussed in the online setting, with most popular works adapting greedy DT construction methods to handling data streams. As a consequence, these approaches inherit the suboptimality limitation of their

greedy batch counterparts. We addressed this issue in Part II of the thesis by developing several algorithms that specifically seek optimal sparse DTs online. Chapter 5 and Chapter 6 introduce three Monte Carlo Tree Search (MCTS) algorithms UCDT, EGDT and TSDT, which are guaranteed to converge to the optimal solution given an infinite data stream. In our experiments, we demonstrated that these MCTS methods successfully retrieve the optimal sparse DT in settings where the state of the art methods are practically guaranteed to be suboptimal. Nevertheless, while an asymptotic optimality guarantee is important, it is insufficient to assess the search efficiency. For this reason, we developed Online-Branches in Chapter 7, an algorithm that is amenable to a finite-time analysis. Unlike UCDT, EGDT and TSDT, Online-Branches does not satisfy asymptotic optimality, but its finite-time guarantee is more insightful, and our empirical analysis shows Online-Branches to be superior to our previous methods. Our work on the online setting can draw the attention of the online Decision Trees community to the importance of seeking optimal sparse Decision Trees. Furthermore, it provides a basis for future works on this topic, of which the most straightforward and promising idea, in our opinion, is to modify the MDP of the MCTS methods to Branches' MDP as it is more efficient. The work we developed in Part II can accelerate this adaptation by providing insights into how to solve several problems that will arise such as Weights Degeneracy (see Section 5.4).

Overall we believe the work we conducted in this thesis to be very important for the field of Interpretable Machine Learning. It advances the state of the art significantly in some aspects like search efficiency with Branches, and it provides valuable insights to mitigating some of the most important issues facing optimal sparse Decision Tree methods like scalability; our investigation of the drawbacks of binary encoding compared to ordinal encoding is a good example of such insight. This thesis also emphasizes optimality concerns in the online setting where it is rarely discussed, and it lays a foundational basis for future works.

Bibliography

Aghaei, S., Gómez, A., and Vayanos, P. (2024). Strong optimal classification trees. *Operations Research*.

Aglin, G., Nijssen, S., and Schaus, P. (2020). Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3146–3153.

Aglin, G., Nijssen, S., and Schaus, P. (2021). Pydl8. 5: a library for learning optimal decision trees. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 5222–5224.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A. I., et al. (1996). Fast discovery of association rules. *Advances in knowledge discovery and data mining*, 12(1):307–328.

Agrawal, S. and Goyal, N. (2012). Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on learning theory*, pages 39–1. JMLR Workshop and Conference Proceedings.

Agrawal, S. and Goyal, N. (2013). Further optimal regret bounds for thompson sampling. In Carvalho, C. M. and Ravikumar, P., editors, *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 99–107, Scottsdale, Arizona, USA. PMLR.

Angelino, E., Larus-Stone, N., Alabi, D., Seltzer, M., and Rudin, C. (2018). Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research*, 18(234):1–78.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256.

Avellaneda, F. (2020). Efficient inference of optimal decision trees. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3195–3202.

Bai, A., Wu, F., and Chen, X. (2013). Bayesian mixture modelling and inference based thompson sampling in monte-carlo tree search. *Advances in neural information processing systems*, 26.

Bai, A., Wu, F., Zhang, Z., and Chen, X. (2014). Thompson sampling based monte-carlo planning in pomdps. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, pages 29–37.

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial intelligence*, 72(1-2):81–138.

Basharin, G. P. (1959). On a statistical estimate for the entropy of a sequence of independent random variables. *Theory of Probability & Its Applications*, 4(3):333–336.

Bellman, R. E. and Dreyfus, S. E. (2015). *Applied dynamic programming*, volume 2050. Princeton university press.

Bennett, K. P. (1992). Decision tree construction via linear programming. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Bennett, K. P. (1994). Global tree optimization: A non-greedy decision tree algorithm. *Computing Science and Statistics*, pages 156–156.

Bennett, K. P. and Blue, J. A. (1996). Optimal decision trees. *Rensselaer Polytechnic Institute Math Report*, 214(24):128.

Bertsimas, D. and Dunn, J. (2017). Optimal classification trees. *Machine Learning*, 106:1039–1082.

Bessiere, C., Hebrard, E., and O'Sullivan, B. (2009). Minimising decision tree size as combinatorial optimisation. In *International Conference on Principles and Practice of Constraint Programming*, pages 173–187. Springer.

Biere, A., Heule, M., and van Maaren, H. (2009). *Handbook of satisfiability*, volume 185. IOS press.

Bifet, A. and Gavalda, R. (2009). Adaptive learning from evolving data streams. In *International Symposium on Intelligent Data Analysis*, pages 249–260. Springer.

Bifet, A. and Kirkby, R. (2009). Data stream mining: a practical approach. *The university of Waikato*.

Blake, C. and Ntoutsi, E. (2018). Reinforcement learning based decision tree induction over data streams with concept drifts. In *2018 IEEE International Conference on Big Knowledge (ICBK)*, pages 328–335. IEEE.

Blanquero, R., Carrizosa, E., Molero-Río, C., and Morales, D. R. (2021). Optimal randomized classification trees. *Computers & Operations Research*, 132:105281.

Bonet, B. and Geffner, H. (1998). Learning sorting and decision trees with pomdps. In *ICML*, pages 73–81. Citeseer.

Boutilier, J. J., Michini, C., and Zhou, Z. (2022). Shattering inequalities for learning optimal decision trees. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 74–90. Springer.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Aaai*, volume 94, pages 1023–1028.

Chaouki, A., Read, J., and Bifet, A. (2023). Online decision tree construction with deep reinforcement learning. In *Sixteenth European Workshop on Reinforcement Learning*.

Chaouki, A., Read, J., and Bifet, A. (2024). Online learning of decision trees with Thompson sampling. In Dasgupta, S., Mandt, S., and Li, Y., editors, *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 2944–2952. PMLR.

Chaouki, A., Read, J., and Bifet, A. (2025). Branches: Efficiently seeking optimal sparse decision trees with ao*.

Clark, C. E. (1961). The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162.

Demirović, E., Hebrard, E., and Jean, L. (2023). Blossom: an anytime algorithm for computing optimal decision trees. In *International Conference on Machine Learning*, pages 7533–7562. PMLR.

Demirović, E., Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., Ramamohanarao, K., and Stuckey, P. J. (2022). Murtree: Optimal decision trees via dynamic programming and search. *The Journal of Machine Learning Research*, 23(1):1169–1215.

Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80.

Esmeir, S. and Markovitch, S. (2007). Anytime learning of decision trees. *Journal of Machine Learning Research*, 8(5).

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Garlapati, A., Raghunathan, A., Nagarajan, V., and Ravindran, B. (2015). A reinforcement learning approach to online learning of decision trees. *arXiv preprint arXiv:1507.06923*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Gouk, H., Pfahringer, B., and Frank, E. (2019). Stochastic gradient trees. In *Asian conference on machine learning*, pages 1094–1109. PMLR.

Günlük, O., Kalagnanam, J., Li, M., Menickelly, M., and Scheinberg, K. (2021). Optimal decision trees for categorical data via integer programming. *Journal of global optimization*, 81:233–260.

Halford, M., Bolmier, G., Sourty, R., Vaysse, R., and Zouitine, A. (2019). creme, a Python library for online machine learning.

Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. *ACM sigmod record*, 29(2):1–12.

Harris, B. (1975). The statistical estimation of entropy in the non-parametric case topics in information theory ed i csiszar.

Hasselt, H. (2010). Double q-learning. *Advances in neural information processing systems*, 23.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

Hu, H., Siala, M., Hebrard, E., and Huguet, M.-J. (2020). Learning optimal decision trees with maxsat and its integration in adaboost. In *IJCAI-PRICAI 2020, 29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence*.

Hu, X., Rudin, C., and Seltzer, M. (2019). Optimal sparse decision trees. *Advances in Neural Information Processing Systems*, 32.

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. (2017). Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*.

Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106.

Jin, R. and Agrawal, G. (2003). Efficient decision tree construction on streaming data. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 571–576.

Kaufmann, E., Korda, N., and Munos, R. (2012). Thompson sampling: An asymptotically optimal finite-time analysis. In *International conference on algorithmic learning theory*, pages 199–213. Springer.

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Kocsis, L., Szepesvári, C., and Willemson, J. (2006). Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1:1–22.

Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, 12.

Lattimore, T. and Szepesvári, C. (2020). *Bandit algorithms*. Cambridge University Press.

Laurent, H. and Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17.

Lawler, E. L. and Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719.

Le Berre, D. and Parrain, A. (2010). The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–64.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, J., Zhong, C., Hu, D., Rudin, C., and Seltzer, M. (2020). Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning*, pages 6150–6160. PMLR.

Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1995). Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. Elsevier.

Lomax, S. and Vadera, S. (2013). A survey of cost-sensitive decision tree induction algorithms. *ACM Computing Surveys (CSUR)*, 45(2):1–35.

Manapragada, C., Webb, G. I., and Salehi, M. (2018). Extremely fast decision tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1953–1962.

Martelli, A. and Montanari, U. (1978). Optimizing decision trees through heuristically guided search. *Communications of the ACM*, 21(12):1025–1039.

McDiarmid, C. et al. (1989). On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188.

McTavish, H., Zhong, C., Achermann, R., Karimalis, I., Chen, J., Rudin, C., and Seltzer, M. (2022). Fast sparse decision tree optimization via reference ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9604–9613.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

Montanari, U. et al. (1975). From dynamic programming to search algorithms with functional costs.

Montiel, J., Halford, M., Mastelini, S. M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H. M., Read, J., Abdessalem, T., et al. (2021). River: machine learning for streaming data in python.

Montiel, J., Read, J., Bifet, A., and Abdessalem, T. (2018a). Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19(72):1–5.

Montiel, J., Read, J., Bifet, A., and Abdessalem, T. (2018b). Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19(72):1–5.

Morris, J. X., Lifland, E., Yoo, J. Y., Grigsby, J., Jin, D., and Qi, Y. (2020). Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. *arXiv preprint arXiv:2005.05909*.

Murthy, S. and Salzberg, S. (1995). Lookahead and pathology in decision tree induction. In *IJCAI*, pages 1025–1033. Citeseer.

Narodytska, N., Ignatiev, A., Pereira, F., Marques-Silva, J., and Ras, I. (2018). Learning optimal decision trees with sat. In *Ijcai*, pages 1362–1368.

Nijssen, S. and Fromont, E. (2007). Mining optimal decision trees from itemset lattices. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 530–539.

Nijssen, S. and Fromont, E. (2010). Optimal constraint-based decision tree induction from itemset lattices. *Data Mining and Knowledge Discovery*, 21:9–51.

Nilsson, N. J. (2014). *Principles of artificial intelligence*. Morgan Kaufmann.

Norouzi, M., Collins, M., Johnson, M. A., Fleet, D. J., and Kohli, P. (2015). Efficient non-greedy optimization of decision trees. *Advances in neural information processing systems*, 28.

Norton, S. W. (1989). Generating better decision trees. In *IJCAI*, volume 89, pages 800–805.

Nunes, C., De Craene, M., Langet, H., Camara, O., and Jonsson, A. (2018). A monte carlo tree search approach to learning decision trees. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 429–435. IEEE.

Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc.

Peters, J. and Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190.

Preda, M. (2007). Adaptive building of decision trees by reinforcement learning. In *Proceedings of the 7th Conference on 7th WSEAS International Conference on Applied Informatics and Communications*, volume 7, pages 34–39. Citeseer.

Quinlan, J. (2014). *C4. 5: programs for machine learning*. Elsevier.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

Rudin, C., Chen, C., Chen, Z., Huang, H., Semenova, L., and Zhong, C. (2022). Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistic Surveys*, 16:1–85.

Russo, D. and Van Roy, B. (2016). An information-theoretic analysis of thompson sampling. *Journal of Machine Learning Research*, 17(68):1–30.

Rutkowski, L., Jaworski, M., Pietruczuk, L., and Duda, P. (2013). Decision trees for mining data streams based on the Gaussian approximation. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):108–119.

Rutkowski, L., Pietruczuk, L., Duda, P., and Jaworski, M. (2012). Decision trees for mining data streams based on the mcdiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1272–1279.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017a). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017b). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Sinha, D., Zhou, H., and Shenoy, N. V. (2007). Advances in computation of the maximum of a set of gaussian random variables. *IEEE Transactions on Computer-Aided design of integrated circuits and systems*, 26(8):1522–1533.

Sondik, E. J. (1971). *The optimal control of partially observable Markov processes*. Stanford University.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tarjan, R. (1971). Depth-first search and linear graph algorithms. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pages 114–121.

Tesauro, G., Rajan, V., and Segal, R. (2012). Bayesian inference in monte-carlo tree search. *arXiv preprint arXiv:1203.3519*.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294.

van der Linden, J., de Weerdt, M., and Demirović, E. (2024). Necessary and sufficient conditions for optimal decision trees using dynamic programming. *Advances in Neural Information Processing Systems*, 36.

Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C.-G., and Schaus, P. (2020). Learning optimal decision trees using constraint programming. *Constraints*, 25:226–250.

Verwer, S. and Zhang, Y. (2017). 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*, pages 94–103. Springer.

Verwer, S. and Zhang, Y. (2019). Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1625–1632.

Wächter, A. and Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8:279–292.

Wen, G. and Wu, K. (2021). Building decision tree for imbalanced classification via deep reinforcement learning. In *Asian Conference on Machine Learning*, pages 1645–1659. PMLR.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

Xiong, Z., Zhang, W., and Zhu, W. (2017). Learning decision trees with reinforcement learning. In *NIPS Workshop on Meta-Learning*.

Zaki, M. J., Parthasarathy, S., Ogihara, M., Li, W., et al. (1997). New algorithms for fast discovery of association rules. In *KDD*, volume 97, pages 283–286.

Zhu, H., Murali, P., Phan, D., Nguyen, L., and Kalagnanam, J. (2020). A scalable mip-based method for learning optimal multivariate decision trees. *Advances in neural information processing systems*, 33:1771–1781.

ECOLE DOCTORALE

Titre : Arbres de décision optimaux via la recherche : un cadre d'apprentissage par renforcement

Mots clés : Arbres de décision optimaux, Processus de décision markovien, Programmation dynamique, Séparation et évaluation, Recherche arborescente Monte-Carlo.

Résumé : Les arbres de décision (DTs) sont des modèles importants en raison de leur interprétabilité, en particulier ceux ayant une petite complexité (ou taille). Cette thèse apporte des contributions à l'optimisation des arbres de décision en pénalisant la complexité. Nous formulons le problème comme un processus de décision markovien (MDP), et nous divisons la thèse en deux parties : **(1)** Apprentissage des arbres de décision optimaux étant donné un ensemble de données fixe, et **(2)** Apprentissage des arbres de décision optimaux étant donné un flux de données. Notre contribution principale dans **(1)** est le développement de l'algorithme Branches. Branches utilise une procédure efficace de programmation dynamique, ainsi qu'une stratégie d'élagage par la méthode séparation et évaluation basée sur une nouvelle borne analytique que nous appelons borne de purification. Nous démontrons théoriquement qu'à la fin de l'exécution de Branches, celui-ci renvoie toujours l'arbre de décision optimal. De plus, nous analysons sa complexité en bornant le nombre de ses étapes d'évaluation, et nous montrons numériquement que cette borne est nettement inférieure aux bornes de complexité fournies dans la littérature. Du point de vue empirique, Branches surpasse l'état de l'art en termes de nombre d'itérations et de temps d'exécution sur de nombreux ensembles de données standards. Branches peut encore être amélioré par une implémentation en C++, d'autant plus qu'il se prête à une formulation pour le calcul parallèle. Dans **(2)**, nous proposons deux approches. La première est basée sur une recherche arborescente Monte Carlo, aboutissant à trois algorithmes UCDT, EGDT et TSDT qui trouvent asymptotiquement l'arbre optimal presque sûrement. La seconde approche, Online-Branches, est une généralisation de Branches pour les flux de données, elle est basée sur une fenêtre glissante de taille fixe. Nous démontrons que sa probabilité d'optimalité, après l'observation d'un certain nombre de données, converge vers 1 exponentiellement avec la taille de la fenêtre. Toutes ces méthodes surpassent les arbres de décision en ligne existants en précision et qualité du modèle final proposé.

Title : Optimal Decision Trees via Search: A Reinforcement Learning framework

Keywords : Decision Trees, Markov Decision Process, Dynamic Programming, Branch & Bound, Monte Carlo Tree Search.

Abstract : Decision Trees (DTs) are important models due to their interpretability, especially those with small complexity (or size). This thesis brings forth contributions to the Decision Tree optimisation problem under a soft constraint on complexity. We formulate the problem as a Markov Decision Process (MDP), and we divide the thesis into two parts: **(1)** Learning optimal DTs in a batch setting where a fixed dataset is provided, and **(2)** Learning optimal DTs in an online setting where we observe a stream of data. Our main contribution in **(1)** is the development of the algorithm Branches. Branches employs an efficient Dynamic Programming procedure, along with a Branch & Bound pruning strategy that is based on a novel analytical bound we call Purification Bound. We prove theoretically that once Branches terminates, it always returns the optimal DT. Moreover, we analyse its complexity by deriving an upper bound on the number of its evaluation steps, and we show numerically that it is significantly lower than complexity bounds provided in the literature. From the empirical standpoint, Branches outperforms the state of the art, in terms of number of iterations and execution time, on a variety of standard datasets. Branches can be further improved through a C++ implementation as it is amenable to parallel computing. In **(2)**, we provide two new approaches. The first approach is based on a Monte Carlo Tree Search formulation and yields three algorithms UCDT, EGDT and TSDT. These three algorithms satisfy asymptotic convergence guarantees; given an infinite stream of data, they find the optimal DT almost surely. On the other hand, we developed Online-Branches, a generalisation of Branches to the online setting based on a sliding window of fixed size. We show that its probability of optimality, after observing a certain number of samples, converges to 1 exponentially with the size of the sliding window. All the methods we introduced in **(2)** outperform the state of the art online DTs in terms of prequential accuracy and quality of the final proposed DT.

Institut Polytechnique de Paris
91120 Palaiseau, France